## Contents

1 User’s Guide 3  
  1.1 Installation ............................................. 3  
  1.2 Configuration ........................................... 3  
  1.3 Usage .................................................. 5  
  1.4 Example application ................................. 9  

2 API Reference 11  
  2.1 API Docs .................................................. 11  

3 Additional Notes 15  
  3.1 Contributing ............................................ 15  
  3.2 Changes ................................................ 17  
  3.3 License .................................................. 17  
  3.4 Contributors ............................................ 17  

Python Module Index 19
User interface for Invenio-Records.

Further documentation is available on https://invenio-records-ui.readthedocs.io/.
This part of the documentation will show you how to get started in using Invenio-Records-UI.

1.1 Installation

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

```bash
$ pip install invenio-records-ui
```

Invenio-Records-UI depends on Invenio-Records, Invenio-PIDStore and Invenio-DB.

1.2 Configuration

Flask extension for Invenio-Records-UI.

```python
invenio_records_ui.config.RECORDS_UI_DEFAULT_PERMISSION_FACTORY = None
    Configure the default permission factory.

invenio_records_ui.config.RECORDS_UI_ENDPOINTS = {'recid': {'pid_type': 'recid', 'route': '/records/<pid_value>', 'template': ...
     Default UI endpoints.

     This option can be overwritten to describe the endpoints of the different record types.
     Each element on the dictionary represent a independent endpoint.

     The structure of the dictionary is as follows:

     ```python
     def my_view(pid, record, template=None):
         return render_template(template, pid=pid, record=record)

     def my_permission_factory(record, *args, **kwargs):
     ```
```
def can(self):
    rec = Record.get_record(record.id)
    return rec.get('access', '') == 'open'
return type('MyPermissionChecker', (), {'can': can})()

RECORDS_UI_ENDPOINTS = {
    "<endpoint-name>": {
        "pid_type": "<record-pid-type>",
        "route": "/records/<pid_value>",
        "template": "invenio_records_ui/detail.html",
        "permission_factory_imp": "my_permission_factory",
        "view_imp": my_view,
        "record_class": "invenio_records.api:Record",
        "methods": ["GET", "POST", "PUT", "DELETE"],
    },
    ...
}

Parameters

• pid_type – Persistent identifier type for endpoint. Required.
• route – URL route (must include <pid_value> pattern). Required.
• template – Template to render. (Default: invenio_records_ui/detail.html)
• permission_factory_imp – Import path to factory that creates a permission object for a given record. If the value is None, then no access control is done. (Default: None)
• view_imp – Import path to view function. (Default: None)
• record_class – Import path of record class. (Default: invenio_records.api:Record)
• methods – List of methods supported. (Default: ['GET'])

invenio_records_ui.config.RECORDS_UI_EXPORT_FORMATS = {}

Default record serialization views.

The structure of the dictionary is as follows:

RECORDS_UI_EXPORT_FORMATS = {
    "<pid-type>": {
        "<format-slug>": {
            "title": "<export format title>",
            "serializer": "<object or import path to record serializer>",
            "order": 1,
        },
        ...
    },
    ...
}

invenio_records_ui.config.RECORDS_UI_LOGIN_ENDPOINT = 'security.login'

Endpoint where redirect the user if login is required.

invenio_records_ui.config.RECORDS_UI_TOMBSTONE_TEMPLATE = 'invenio_records_ui/tombstone.html'

Configure the tombstone template.
1.3 Usage

Module for displaying records.

Invenio-Records-UI is a core component of Invenio which provides configurable views for display records. It uses Invenio-PIDStore to resolve an external persistent identifier into an internal record object. It also has support for displaying tombstones for deleted records, as well as redirecting an external persistent identifier to another in case e.g. records are merged.

In simple terms, Records-UI works by creating one or more endpoints for displaying records. You can e.g. have one endpoint (/records/) for displaying bibliographic records and another endpoint (/authors/) for displaying author records.

One endpoint is bound to one and only one specific persistent identifier type. For instance, /records/ could be bound to integer record identifiers, while /authors/ could be bound to ORCID identifiers.

1.3.1 Initialization

First create a Flask application (Flask-CLI is not needed for Flask version 1.0+):

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

You initialize Records-UI like a normal Flask extension, however Invenio-Records-UI is dependent on Invenio-Records, Invenio-PIDStore and Invenio-DB so you need to initialize these extensions first:

```python
>>> from invenio_db import InvenioDB
>>> ext_db = InvenioDB(app)
>>> from invenio_records import InvenioRecords
>>> from invenio_pidstore import InvenioPIDStore
>>> from invenio_records_ui import InvenioRecordsUI
>>> ext_pidstore = InvenioPIDStore(app)
>>> ext_records = InvenioRecords(app)
```

Configuration

Before we initialize the InvenioRecordsUI extension, we need to configure which endpoints we want to expose. Let’s start with one endpoint /records/ which resolves integer record identifiers to internal record objects.

```python
>>> app.config['RECORDS_UI_ENDPOINTS'] = dict(
...     recid=dict(
...         pid_type='recid',
...         route='/records/<pid_value>',
...         template='invenio_records_ui/detail.html',
...     )
... )
```

Here we create a single endpoint named recid (note, the naming of endpoints are important - see the section on redirection for details). The endpoint resolves recid persistent identifiers (see pid_type key), on the URL route /records/<pid_value>.

You are free to choose any URL route as long as it includes the <pid_value> URL pattern in the route.

The created endpoint will render the template invenio_records_ui/detail.html which will be passed the following two variables in the template context:
Installing endpoints

We have now configured which endpoint we want, so we can go ahead and install the extension and register the endpoints (note, in this example we switch off the permission checking capabilities by setting `RECORDS_UI_DEFAULT_PERMISSION_FACTORY` to `None`):

```python
>>> from invenio_records_ui.views import create_blueprint_from_app
>>> app.config['RECORDS_UI_DEFAULT_PERMISSION_FACTORY'] = None
>>> ext_records_ui = InvenioRecordsUI(app)
>>> app.register_blueprint(create_blueprint_from_app(app))
```

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

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

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

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

1.3.2 Displaying a record

Before we can display a record, we need a persistent identifier and a record, so let’s create them:

```python
>>> from uuid import uuid4
>>> from invenio_db import db
>>> from invenio_records.api import Record
>>> from invenio_pidstore.providers.recordid import RecordIdProvider

rec_uuid = uuid4()
rec = Record.create({'title': 'My title'}, id_=rec_uuid)
provider = RecordIdProvider.create(object_type='rec', object_uuid=rec_uuid)
db.session.commit()
```

In above example we use the `RecordIdProvider` to create a `recid` persistent identifier type:

```python
>>> print(provider.pid.pid_type)
recid
>>> print(provider.pid.pid_value)
1
```

We can now access the record:

```python
>>> with app.test_client() as client:
...     res = client.get('/records/1')
>>> res.status_code
200
```

If you try to access a non-existing record, you will naturally receive a not found page:
It is important to note that only persistent identifiers in registered state will resolve. If you have a persistent identifier in new state, it will return a 404 error code.

### 1.3.3 Tombstones

If you need to delete a record for some reason, it’s possible to display a tombstone for the record, so that any external links to the record may still see a landing page with information on why a given record was removed.

First, let’s create a persistent identifier and delete it:

```python
>>> provider = RecordIdProvider.create()
>>> print(provider.pid.pid_value)
2
>>> provider.pid.delete()
True
>>> db.session.commit()
```

If we now try to access the same record as before, we will receive a 410 error code:

```python
>>> with app.test_client() as client:
...    res = client.get('/records/2')
>>> res.status_code
410
```

The template being rendered is `invenio_records_ui/tombstone.html`, which you can change using the `RECORDS_UI_TOMBSTONE_TEMPLATE` configuration variable:

```python
>>> app.config['RECORDS_UI_TOMBSTONE_TEMPLATE']
'invenio_records_ui/tombstone.html'
```

The template will receive two variables in the template context:
- `pid` - the persistent identifier
- `record` - the internal record object or an empty dict in case no record was assigned to the persistent identifier.

### 1.3.4 Redirection

You can redirect one persistent identifier to another persistent identifier. This can be useful in cases where you need to e.g. merge two records.

Let’s create a persistent identifier and redirect it:

```python
>>> from invenio_pidstore.models import PersistentIdentifier, 
...     PIDStatus
>>> provider = RecordIdProvider.create(
...     status=PIDStatus.REGISTERED)
>>> provider.pid.redirect(
...     PersistentIdentifier.get('recid', '1'))
True
>>> db.session.commit()
```
If you now try to access the redirected persistent identifier, you will be redirected:

```python
>>> with app.test_client() as client:
...    res = client.get('/records/3')
>>> res.status_code
302
>>> print(res.location)
http://localhost/records/1
```

### Naming of endpoints

For redirection to work for a given persistent identifier type, you must provide exactly one endpoint with the name of the type. For instance, in the redirection above, we redirected a `recid` persistent identifier type to another `recid` persistent identifier type. This redirect works because we named the endpoint `recid`:

```python
>>> app.config['RECORDS_UI_ENDPOINTS']
{'recid': ...}
```

Had we instead named the endpoint e.g. `records`, the redirect would not have worked.

### 1.3.5 Signals

Every time a record is viewed a signal is sent. This allows you to e.g. track viewing events. In this example, let’s just create a signal receiver which prints the persistent identifier which was viewed:

```python
>>> from invenio_records_ui.signals import record_viewed
>>> def receiver(sender, record=None, pid=None):
...    print("Viewed record {0}".format(pid.pid_value))
```

If we now try to access the record

```python
>>> with record_viewed.connected_to(receiver):
...    with app.test_client() as client:
...        res = client.get('/records/1')
Viewed record 1
>>> res.status_code
200
```

### 1.3.6 Access control

Invenio-Records-UI is integrated with Flask-Principal to provide access control to records. To protect access to a record you must provide a permission factory. A permission factory is a simple method which takes a record and returns an permission instance:

```python
>>> from flask_principal import Permission, RoleNeed
>>> def perm_factory(record):
...    return Permission(RoleNeed('admin'))
```

This allows the permission factory to make use of any information inside and outside of the record in order to create a permission to protect it. This allows very fine-grained control with who can access which record and how you protect it.

The permission factory you can apply globally to all endpoints by setting `RECORDS_UI_DEFAULT_PERMISSION_FACTORY` to the import path of the permission factory:
Alternatively you can also apply a permission factory to only a specific endpoint by passing the `permission_factory_imp` argument:

```
>>> app.config["RECORDS_UI_ENDPOINTS"] = dict(
    ...    recid=dict(
        ...        pid_type='recid',
        ...        route='/records/<pid_value>',
        ...        template='invenio_records_ui/detail.html',
        ...        permission_factory_imp=perm_factory,
        ...    ),
    ...)
```

### 1.3.7 Custom view functions

Invenio-Records-UI also has support for custom view functions, in case you need more customization than what is possible with templates.

The custom view function must take a persistent identifier and a record as arguments as well as keyword arguments. The view function is called at the very end after the persistent identifier have been resolved to a record and any access control have been checked as well.

```
>>> def custom_view(pid, record, template=None, **kwargs):
    ...    return "CUSTOM: {0}".format(pid.pid_value)
```

You configure the endpoint to use your custom view by passing the `view_imp` argument to your endpoints configuration:

```
>>> app.config["RECORDS_UI_ENDPOINTS"] = dict(
    ...    recid=dict(
        ...        pid_type='recid',
        ...        route='/records/<pid_value>',
        ...        view_imp='mymodule.custom_view'
        ...    ),
    ...)
```

### 1.4 Example application

#### 1.4.1 Simple example

Run the Redis server.

Run example development server:

```
$ pip install -e .[all]
$ cd examples
$ export FLASK_APP=app.py
$ ./app-setup.sh
$ ./app-fixtures.sh
```

Run example development server:
$ flask run --debugger -p 5000

View some records in your browser:

http://localhost:5000/records/1
http://localhost:5000/records/2
http://localhost:5000/records/3
http://localhost:5000/records/4
http://localhost:5000/records/5
http://localhost:5000/records/6
http://localhost:5000/records/7
http://localhost:5000/records/8

To be able to uninstall the example app:

$ ./app-teardown.sh

1.4.2 With permission support

Run the Redis server.

Run the the example development server:

$ pip install -e .[all]
$ cd examples
$ export FLASK_APP=permsapp.py
$ ./app-setup.sh
$ ./app-fixtures.sh

Run example development server:

$ flask run --debugger -p 5000

Try to view record 1:

http://localhost:5000/records/1

Open the record 1:

http://localhost:5000/records/1

Try now to open the record 2:

http://localhost:5000/records/2

As you can see, for this user the action is forbidden.

To be able to uninstall the example app:

$ ./app-teardown.sh
API Reference

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

2.1 API Docs

2.1.1 Flask extension

Flask extension for Invenio-Records-UI.

```python
class invenio_records_ui.ext.InvenioRecordsUI(app=None)
Invenio-Records-UI extension.

The extension takes care of setting default configuration and registering a blueprint with URL routes for the endpoints.

Extension initialization.

Parameters app – The Flask application. (Default: None)

init_app (app)
Flask application initialization.

Parameters app – The Flask application.

init_config (app)
Initialize configuration on application.

Parameters app – The Flask application.
```

class invenio_records_ui.ext._RecordUIState (app)
Record UI state.

Initialize state.

Parameters app – The Flask application.
export_formats \((\text{pid\_type})\)

List of export formats.

permission_factory

Load default permission factory.

### 2.1.2 Views

Factory for creating a blueprint for Invenio-Records-UI.

invenio_records_ui.views.create_blueprint \((\text{endpoints})\)

Create Invenio-Records-UI blueprint.

The factory installs one URL route per endpoint defined, and adds an error handler for rendering tombstones.

**Parameters** endpoints – Dictionary of endpoints to be installed. See usage documentation for further details.

**Returns** The initialized blueprint.

invenio_records_ui.views.create_blueprint_from_app \((\text{app})\)

Create Invenio-Records-UI blueprint from a Flask application.

**Note:** This function assumes that the application has loaded all extensions that want to register REST endpoints via the RECORDS_UI_ENDPOINTS configuration variable.

**Params** app – A Flask application.

**Returns** Configured blueprint.

invenio_records_ui.views.create_url_rule \((\text{endpoint}, \text{route}=\text{None}, \text{pid\_type}=\text{None}, \text{template}=\text{None}, \text{permission\_factory\_imp}=\text{None}, \text{view\_imp}=\text{None}, \text{record\_class}=\text{None}, \text{methods}=\text{None})\)

Create Werkzeug URL rule for a specific endpoint.

The method takes care of creating a persistent identifier resolver for the given persistent identifier type.

**Parameters**

- endpoint – Name of endpoint.
- route – URL route (must include \(<\text{pid\_value}>\) pattern). Required.
- pid_type – Persistent identifier type for endpoint. Required.
- template – Template to render. (Default: invenio_records_ui/detail.html)
- permission_factory_imp – Import path to factory that creates a permission object for a given record.
- view_imp – Import path to view function. (Default: None)
- record_class – Name of the record API class.
- methods – Method allowed for the endpoint.

**Returns** A dictionary that can be passed as keywords arguments to Blueprint.add_url_rule.
invenio_records_ui.views.default_view_method (pid, record, template=None, **kwargs)

Display default view.

Sends record_viewed signal and renders template.

Parameters

• pid – PID object.
• record – Record object.
• template – Template to render.
• **kwargs – Additional view arguments based on URL rule.

Returns

The rendered template.

invenio_records_ui.views.export (pid, record, template=None, **kwargs)

Record serialization view.

Serializes record with given format and renders record export template.

Parameters

• pid – PID object.
• record – Record object.
• template – Template to render.
• **kwargs – Additional view arguments based on URL rule.

Returns

The rendered template.

invenio_records_ui.views.record_view (pid_value=None, resolver=None, template=None, permission_factory=None, view_method=None, **kwargs)

Display record view.

The two parameters resolver and template should not be included in the URL rule, but instead set by creating a partially evaluated function of the view.

The template being rendered is passed two variables in the template context:

• pid
• record

Procedure followed:

1. PID and record are resolved.
2. Permission are checked.
3. view_method is called.

Parameters

• pid_value – Persistent identifier value.
• resolver – An instance of a persistent identifier resolver. A persistent identifier resolver takes care of resolving persistent identifiers into internal objects.
• template – Template to render.
• permission_factory – Permission factory called to check if user has enough power to execute the action.
• view_method – Function that is called.
Returns Tuple (pid object, record object).

2.1.3 Signals

Record module signals.

invenio_records_ui.signals.record_viewed = <blinker.base.NamedSignal object at 0x7fdb33fb0450>

Signal sent when a record is viewed on any endpoint.

Parameters:

• sender - a Flask application object.
• pid - a persistent identifier instance.
• record - a record instance.

Example receiver:

```python
def receiver(sender, record=None, pid=None):
    # ...
```

Note, the signal is always sent in a request context, thus it is safe to access the current request and/or current user objects inside the receiver.
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


If you are reporting a bug, please include:

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

Fix Bugs

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

Implement Features

Look through the GitHub issues for features. Anything tagged with “feature” is open to whoever wants to implement it.
Write Documentation

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

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/inveniosoftware/invenio-records-ui/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-records-ui for local development.

1. Fork the invenio-records-ui repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/invenio-records-ui.git
   ```

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

   ```
   $ mkvirtualenv invenio-records-ui
   $ cd invenio-records-ui/
   $ 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 "component: title without verbs"
   `-m "* NEW Adds your new feature."
   `-m "* FIX Fixes an existing issue."
   `-m "* BETTER Improves and existing feature."
   `-m "* Changes something that should not be visible in release notes."
   $ 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, 3.5 and 3.6. Check https://travis-ci.org/inveniosoftware/invenio-records-ui/pull_requests and make sure that the tests pass for all supported Python versions.

3.2 Changes

Version 1.0.1 (released 2018-05-25)
• Changes dynamic blueprint registration to after the extension initialization phase (entry point works only with Invenio-Base v1.0.1+).

Version 1.0.0 (released 2018-03-23)
• Initial public release.

3.3 License

MIT License
Copyright (C) 2015-2018 CERN.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

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

3.4 Contributors

• Alexander Ioannidis
• Alizée Pace
• Diego Rodriguez
• Dinos Kousidis
• Harris Tzovanakis
• Jacopo Notarstefano
• Jiri Kuncar
• Lars Holm Nielsen
• Leonardo Rossi
• Nicolas Harraudeau
• Tibor Simko
Symbol Index

_Symbols_
.RecordUIState (class in invenio_records_ui.ext), 11

_C_
create_blueprint() (in module invenio_records_ui.views), 12
create_blueprint_from_app() (in module invenio_records_ui.views), 12
create_url_rule() (in module invenio_records_ui.views), 12

_D_
default_view_method() (in module invenio_records_ui.views), 12

_E_
extport() (in module invenio_records_ui.views), 13
export_formats() (invenio_records_ui.ext._RecordUIState method), 11

_I_
init_app() (invenio_records_ui.ext.InvenioRecordsUI method), 11
init_config() (invenio_records_ui.ext.InvenioRecordsUI method), 11
invenio_records_ui (module), 5
invenio_records_ui.config (module), 3
invenio_records_ui.ext (module), 11
invenio_records_ui.signals (module), 14
invenio_records_ui.views (module), 12
InvenioRecordsUI (class in invenio_records_ui.ext), 11

_P_
permission_factory (invenio_records_ui.ext._RecordUIState attribute), 12

_R_
record_view() (in module invenio_records_ui.views), 13
record_viewed (in module invenio_records_ui.signals), 14
RECORDS_UI_DEFAULT_PERMISSION_FACTORY (in module invenio_records_ui.config), 3
RECORDS_UI_ENDPOINTS (in module invenio_records_ui.config), 3
RECORDS_UI_EXPORT_FORMATS (in module invenio_records_ui.config), 4
RECORDS_UI_LOGIN_ENDPOINT (in module invenio_records_ui.config), 4
RECORDS_UI_TOMBSTONE_TEMPLATE (in module invenio_records_ui.config), 4