invenio-previewer Documentation

Release 1.2.1

CERN

May 07, 2020

Contents

Invenio module for previewing files.

Features:

- File previewing.
- Support for PDF, ZIP, CSV, MARKDOWN, XML, JSON, PNG, JPG and GIF out-of-the-box.
- Extensible API to create new previewers.

Further documentation available: https://invenio-previewer.readthedocs.io/

CHAPTER 1

User's Guide

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

1.1 Installation

Invenio-Previewer is on PyPI so all you need is:

```
pip install invenio-previewer
```

Invenio-Previewer depends on Invenio-Assets for assets bundling and Invenio-PidStore and Invenio-Records-UI for record integration.

You will normally use it in combination with files. You can install the extra Invenio modules Invenio-Files-REST and Invenio-Records-Files by specifying the `files` key in extras:

```
pip install invenio-previewer[files]
```

1.2 Usage

Invenio module for previewing files.

Invenio-Previewer provides extensible file previewers for Invenio. It can be easily integrated with Invenio-Records-UI to serve the file preview under a simple URL path like `/records/<pid_value>/preview`.

It includes previewers for the following file types:

- PDF (using PDF.js)
- ZIP
- CSV (using d3.js)
- Markdown (using Mistune library)

- XML and JSON (using Prism.js)
- Simple images (PNG, JPG, GIF)
- Jupyter Notebooks

Invenio-Previewer only provides the front-end layer for displaying previews of files. Specifically, Invenio-Previewer does not take care of generating derived formats such thumbnails etc. This could be done by Invenio-IIIF.

1.2.1 Initialization

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

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

Configuration

Invenio-Previewer is enabled by adding a `invenio_records_ui.config.RECORDS_UI_ENDPOINTS` endpoint with a custom view function set to *`invenio_previewer.views.preview`*

```
>>> app.config.update(
...     SQLALCHEMY_TRACK_MODIFICATIONS=False,
...     RECORDS_UI_DEFAULT_PERMISSION_FACTORY=None,
...     RECORDS_UI_ENDPOINTS=dict(
...         recid=dict(
...             pid_type='recid',
...             route='/records/<pid_value>',
...             template='invenio_records_ui/detail.html',
...         ),
...         recid_previewer=dict(
...             pid_type='recid',
...             route='/records/<pid_value>/preview/<filename>',
...             view_imp='invenio_previewer.views:preview',
...             record_class='invenio_records_files.api:Record',
...         ),
...     )
... )
```

Here, we configure the URL route to `/records/<pid_value>/preview`, but you can set it to whatever you like. Records-UI takes care resolving the URL to a record and checks access control for the record's file.

Extensions

Now that we have configured the Flask application, let's initialize all dependent Invenio extensions:

```
>>> from flask_babelex import Babel
>>> from invenio_assets import InvenioAssets
>>> from invenio_db import InvenioDB, db
>>> from invenio_records import InvenioRecords
>>> from invenio_records_ui import InvenioRecordsUI
>>> from invenio_files_rest import InvenioFilesREST
>>> from invenio_records_ui.views import create_blueprint_from_app
>>> ext_babel = Babel(app)
```

(continues on next page)

(continued from previous page)

```
>>> ext_assets = InvenioAssets(app)
>>> ext_db = InvenioDB(app)
>>> ext_records = InvenioRecords(app)
>>> ext_records_ui = InvenioRecordsUI(app)
>>> ext_files_rest = InvenioFilesREST(app)
>>> ext_blueprints = app.register_blueprint(create_blueprint_from_app(app))
```

The above modules provide the following features to Invenio-Previewer:

- Invenio-Assets: JavaScript/CSS bundling for interactive previewers.
- Invenio-Records-UI: Retrieval of record and access control.
- Invenio-Files-REST: Access to local files (required only if the previewer plugin needs access to the content of a file).

Lastly, initialize Invenio-Previewer:

```
>>> from invenio_previewer import InvenioPreviewer
>>> ext_previewer = InvenioPreviewer(app)
```

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

```
>>> app.app_context().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):

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

1.2.2 Previewing files

Invenio-Previewer looks into record's metadata to find which files can be previewed. First, we need to create a record and attach a file to it.

Creating a record

When creating a record, by default a bucket is created and associated to the record. Therefore, before creating the record, we need to initialize a default location:

```
>>> import tempfile
>>> from six import BytesIO
>>> from invenio_files_rest.models import Bucket, Location, \
...     ObjectVersion
>>> from invenio_records_files.api import RecordsBuckets
>>> tmpdir = tempfile.mkdtemp()
>>> loc = Location(name='local', uri=tmpdir, default=True)
>>> db.session.add(loc)
>>> db.session.commit()
```

Now we can create the record and its persistent identifier:

```
>>> from uuid import uuid4
>>> from invenio_pidstore.providers.recordid import RecordIdProvider
>>> rec_uuid = uuid4()
>>> provider = RecordIdProvider.create(
...     object_type='rec', object_uuid=rec_uuid)
>>> from invenio_records_files.api import Record
>>> data = {
...     'pid_value': provider.pid.pid_value,
... }
>>> record = Record.create(data, id_=rec_uuid)
>>> db.session.commit()
```

Adding files

We can then add a few demo files into the record:

```
>>> import os
>>> demo_files_path = 'examples/demo_files'
>>> demo_files = (
...     'markdown.md',
...     'csvfile.csv',
...     'zipfile.zip',
...     'jsonfile.json',
...     'xmlfile.xml',
...     'notebook.ipynb',
...     'jpgfile.jpg',
...     'pngfile.png',
... )
```

```
>>> for f in demo_files:
...     with open(os.path.join(demo_files_path, f), 'rb') as fp:
...         record.files[f] = fp
```

```
>>> record.files.flush()
>>> _ = record.commit()
>>> db.session.commit()
```

Previewing a file

Let's try to preview the MarkDown file. What we expect is that the previewer will be able to convert the MarkDown to HTML on the fly and we will see a nice MarkDown preview:

```
>>> with app.test_client() as client:
...     res = client.get('/records/1/preview/markdown.md')
```

Here a more detailed explanation:

1. When calling the url `/records/1/preview/markdown.md`, Invenio-Records-UI resolves the record from the PID 1 and it passes it to Invenio-Previewer.
2. Invenio-Previewer retrieves the file object by the filename `markdown.md` using the `invenio_previewer.ext.record_file_factory` property (implemented by Invenio-Records-Files if installed).
3. With the file object, Invenio-Previewer iterates the list of configured previewers until it finds the first that is able to preview the file.

4. The preview plugin finally takes care of rendering the file.

1.2.3 Bundled previewers

This module contains several previewers out-of-the-box:

- `Markdown`: Previews a markdown file. It is based on python mistune library.
- `JSON/XML`: Previews JSON and XML files. It pretty-prints the contents and applies syntax highlighting using the Prism.js library. You can also configure the maximum file size in order to avoid client and server freezes. By default it is set to 1MB.
- `CSV` - Previews *CSV* files but it can actually work with any other tabular data format in plain text based on the idea of separated values due to it is detecting the delimiter between the characters automatically. On the client side, the file is previewed using d3.js library.
- `PDF` - Previews a PDF file in your browser using PDF.js library.
- `Simple Images` - Previews simple images. Supported formats are JPG, PNG and GIF. There is also a configurable file size limit, which by default is set to 512KB.
- `ZIP` - Previews file tree inside the archive. You can specify a files limit to avoid a temporary lock in both of client and server side when you are dealing with large ZIP files. By default, this limit is set 1000 files.
- `Jupyter Notebook` - Previews a Jupyter Notebook in your browser using Jupyter notebook python converter.
- `Default` - This previewer is intended to be a fallback previewer to those cases when there is no previewer to deal with some file type. It is showing a simple message that the file cannot be previewed.

Local vs. remote files

Some of the bundled previewers are only working with locally managed files (i.e. files stored in Invenio-Files-REST, which supports many different storage backends). This is the case for JSON, XML, CSV, Markdown and ZIP previewers. The PDF and Image previewer doesn't need to have the files stored locally.

1.2.4 Override default previewer

The default previewer for a file can be overridden in two ways:

1. By providing a `previewer` key in the file object, with the name of the previewer to use. This works on a per-file basis.

```
fileobj['previewer'] = 'zip'
```

2. By listing the previewer plugin search order in `PREVIEWER_PREFERENCES`. The first item in the list is the most prioritized previewer in case of collision.

1.2.5 Custom previewer

The implementation of a custom previewer is an easy process. You need to implement the preview function, declare the previewer in the related entry points and define its priority.

Let's try to create a `txt` file previewer. We need to implement:

1. `can_preview()` method: called in order to check if a given file can be previewed and should return a boolean.

2. `preview()` method: called to render the preview.
3. `previewable_extensions` property: string list of previewer's supported file extensions.

Here an example code:

```
>>> class TxtPreviewer(object):
...     previewable_extensions = ['txt']
...     def can_preview(file):
...         return file.file['uri'].endswith('.txt')
...     def preview(file):
...         with file.open() as fp:
...             content = fp.read().decode('utf-8')
...         return content
>>> txt_previewer = TxtPreviewer()
```

Once we have our code, we have to register the previewer in the previewer entry points:

```
>>> entry_points = {
...   'invenio_previewer.previewers': [
...     'tex_previewer = \
...         myproject.modules.previewer.extensions.txt_previewer',
...   ]}
```

Now define the priority for all previewers by adding the newly created `txt_previewer` to the 1st position, so it has the highest priority:

```
>>> PREVIEWER_PREVIEWERS_ORDER = [
...     'invenio_previewer.extensions.txt_previewer',
...     'invenio_previewer.extensions.csv_dthreejs',
...     'invenio_previewer.extensions.json_prismjs',
...     'invenio_previewer.extensions.xml_prismjs',
...     'invenio_previewer.extensions.simple_image',
...     'invenio_previewer.extensions.mistune',
...     'invenio_previewer.extensions.pdfjs',
...     'invenio_previewer.extensions.zip',
...     'invenio_previewer.extensions.default',
... ]
```

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

Invenio module for previewing files.

class invenio_previewer.ext.**InvenioPreviewer**(*app*, ***kwargs*)
Invenio-Previewer extension.

Extension initialization.

init_app(*app*, *entry_point_group='invenio_previewer.previewers'*)
Flask application initialization.

init_config(*app*)
Initialize configuration.

invenio_previewer.ext.**load_or_import_from_config**(*key*, *app=None*, *default=None*)
Load or import value from config.

invenio_previewer.ext.**obj_or_import_string**(*value*, *default=None*)
Import string or return object.

2.1.1 API

File reader utility.

class invenio_previewer.api.**PreviewFile**(*pid*, *record*, *fileobj*)
Preview file default implementation.

Default constructor.

Parameters **file** – ObjectVersion instance from Invenio-Files-REST.

bucket
Get bucket.

filename
Get filename.

has_extensions(**exts*)
Check if file has one of the extensions.

is_local()
Check if file is local.

open()
Open the file.

size
Get file size.

uri
Get file download link.

> **Note:** The URI generation assumes that you can download the file using the view `invenio_records_ui.<pid_type>_files`.

2.1.2 Proxies

Proxy for current previewer.

invenio_previewer.proxies.**current_previewer = <LocalProxy unbound>**
Proxy object to the current previewer extension.

2.1.3 Utils

Invenio Previewer Utilities.

invenio_previewer.utils.**detect_encoding**(*fp*, *default=None*)
Detect the character encoding of a file.

Parameters

- **fp** – Open Python file pointer.
- **default** – Fallback encoding to use.

Returns The detected encoding.

> **Note:** The file pointer is returned at its original read position.

2.1.4 Views

View method for Invenio-Records-UI for previewing files.

invenio_previewer.views.**blueprint = <flask.blueprints.Blueprint object>**
Blueprint used to register template and static folders.

invenio_previewer.views.**is_previewable**(*extension*)
Test if a file can be previewed checking its extension.

invenio_previewer.views.**preview**(*pid*, *record*, *template=None*, ***kwargs*)
Preview file for given record.

Plug this method into your RECORDS_UI_ENDPOINTS configuration:

```
RECORDS_UI_ENDPOINTS = dict(
    recid=dict(
        # ...
        route='/records/<pid_value/preview/<path:filename>',
        view_imp='invenio_previewer.views.preview',
        record_class='invenio_records_files.api:Record',
    )
)
```

2.1.5 Webpack

2.1.6 Previewers

CSV (d3.js)

Render a CSV file using d3.js.

invenio_previewer.extensions.csv_dthreejs.**can_preview**(*file*)
Determine if the given file can be previewed.

invenio_previewer.extensions.csv_dthreejs.**preview**(*file*)
Render the appropriate template with embed flag.

invenio_previewer.extensions.csv_dthreejs.**validate_csv**(*file*)
Return dialect information about given csv file.

Default

Default rendering returning a default web page.

invenio_previewer.extensions.default.**can_preview**(*file*)
Return if file type can be previewed.

invenio_previewer.extensions.default.**preview**(*file*)
Return the appropriate template and passes the file and embed flag.

JSON (prism.js)

Previews a JSON file.

invenio_previewer.extensions.json_prismjs.**can_preview**(*file*)
Determine if the given file can be previewed.

invenio_previewer.extensions.json_prismjs.**preview**(*file*)
Render the appropriate template with embed flag.

invenio_previewer.extensions.json_prismjs.**render**(*file*)
Pretty print the JSON file for rendering.

invenio_previewer.extensions.json_prismjs.**validate_json**(*file*)
Validate a JSON file.

Markdown

Markdown rendering using mistune library.

invenio_previewer.extensions.mistune.**can_preview**(*file*)
Determine if file can be previewed.

invenio_previewer.extensions.mistune.**preview**(*file*)
Render Markdown.

invenio_previewer.extensions.mistune.**render**(*file*)
Render HTML from Markdown file content.

PDF (pdf.js)

PDF previewer based on pdf.js.

invenio_previewer.extensions.pdfjs.**can_preview**(*file*)
Check if file can be previewed.

invenio_previewer.extensions.pdfjs.**preview**(*file*)
Preview file.

Simple Images

Previews simple image files.

invenio_previewer.extensions.simple_image.**can_preview**(*file*)
Determine if the given file can be previewed.

invenio_previewer.extensions.simple_image.**preview**(*file*)
Render the appropriate template with embed flag.

invenio_previewer.extensions.simple_image.**validate**(*file*)
Validate a simple image file.

XML (prism.js)

Previews an XML file.

invenio_previewer.extensions.xml_prismjs.**can_preview**(*file*)
Determine if the given file can be previewed.

invenio_previewer.extensions.xml_prismjs.**preview**(*file*)
Render appropiate template with embed flag.

invenio_previewer.extensions.xml_prismjs.**render**(*file*)
Pretty print the XML file for rendering.

invenio_previewer.extensions.xml_prismjs.**validate_xml**(*file*)
Validate an XML file.

ZIP

Simple ZIP archive previewer.

invenio_previewer.extensions.zip.**can_preview**(*file*)
Return True if filetype can be previewed.

invenio_previewer.extensions.zip.**children_to_list**(*node*)
Organize children structure.

invenio_previewer.extensions.zip.**make_tree**(*file*)
Create tree structure from ZIP archive.

invenio_previewer.extensions.zip.**preview**(*file*)
Return the appropriate template and pass the file and an embed flag.

2.1.7 Bundles

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-previewer/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-Previewer could always use more documentation, whether as part of the official Invenio-Previewer 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-previewer/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-previewer.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-previewer
$ cd invenio-previewer/
$ 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-previewer/pull_requests and make sure that the tests pass for all supported Python versions.

3.2 Changes

3.2.1 Version 1.2.1 (Release 2020-05-07)

- set Sphinx <3 because of errors related to application context
- stop using example app

3.2.2 Version 1.2.0 (Release 2020-03-13)

- Change flask dependency to centrally managed by invenio-base
- Drop support for Python 2.7

3.2.3 Version 1.1.0 (Release 2019-12-20)

- Changes styling and method signature of file_list macro.

3.2.4 Version 1.0.2 (Release 2019-11-21)

- Removes inline styling from simple image previewer for Content Security Policy compliance

3.2.5 Version 1.0.1 (Release 2019-08-02)

- Removes html sanitization config

3.2.6 Version 1.0.0 (release 2019-07-29)

- Initial public release.

3.3 License

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.

Invenio-Preivewer includes the following external libraries:

- **PDF.js**, Copyright 2012 Mozilla Foundation, licensed under Apache License 2.0. The source code is found under:
  - `/assets/css/pdfjs`
  - `/assets/js/pdfjs`
  - `/static/css/pdfjs`
  - `/static/js/pdfjs`

3.4 Contributors

- Alex Ioannidis
- Alexander Ioannidis
- Alizee Pace
- Aman Jain
- Chris Topaloudis
- Francois DECOURCELLE
- Javier Delgado
- Jiri Kuncar
- Lars Holm Nielsen
- Leonardo Rossi
- Michaël Zasso
- Sami Hiltunen
- Tibor Simko

Python Module Index

i

Index

M

O

P

R

S

U

V