invenio-sequencegenerator Documentation

Release 1.0.0a2

CERN

Aug 28, 2017

Contents

Invenio module for generating sequences.

This is an experimental developer preview release.

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

CHAPTER 1

User's Guide

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

Installation

```
$ pip install -e .[all]
```

Invenio-SequenceGenerator depends on Invenio-DB.

Usage

Invenio module for generating sequences.

Invenio-SequenceGenerator is a secondary component of Invenio, responsible for automatically generating identifiers in a safe manner (i.e. making sure that every time a client requests a new identifier of a specific form, the server will provide him with a new *unique* identifier).

Conventions

Conventions regarding template strings of sequences:

- They must contain exactly one placeholder for the counter, named 'counter', which possibly contains conversion and/or format options.

Correct	False
`'File {counter}'`	`'File {count}'`
`'{counter}: File'`	`'File'`
`'{counter:02d}: File'`	`'File {counter:invalid_format}'`

Note: You can format the counter (and all other placeholders) with all the formatting options provided by Python strings.

- They can contain an optional placeholder for a parent sequence, named as the parent sequence, defined when the *Sequence.create* API function was called.

Correct	False
PL = `'Playlist {counter}'`	`'{PL}-{FL} {counter}'`
FL = `'{PL}: {counter}'`	`'{INVALID}: {counter}'`

  For a more in-depth example, see *Hierarchical identifiers*.

- They can contain an arbitrary number of user-defined placeholders, which must be always passed as keyword arguments on each instantiation. No exception is raised in case of redundant keyword arguments.

  You can find more examples in *User-defined keywords*.

Initialization

First create a Flask application:

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

Then initialise the Invenio-DB extension:

```
>>> from invenio_db import InvenioDB
>>> ext_db = InvenioDB(app)
```

Also, initialise the Invenio-SequenceGenerator extension:

```
>>> from invenio_sequencegenerator.ext import InvenioSequenceGenerator
>>> ext_seq = InvenioSequenceGenerator(app)
```

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

```
>>> 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):

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

Template definition

The first thing to do is to define your sequence (i.e. the specific form of your identifiers):

```
>>> from invenio_sequencegenerator.api import Template
>>> template = Template.create('ID', '{counter}-file')
```

Simple counters

Next, you have to get a sequence based on the aforementioned template. During this tutorial, this will be referred to as ‘instantiation’.

```
>>> from invenio_sequencegenerator.api import Sequence
>>> seq = Sequence(template)
```

Then you can request new identifiers from the sequence defined above:

```
>>> seq.next()
'0-file'
>>> seq.next()
'1-file'
>>> seq.next()
'2-file'
```

Note: By default, counters start from 0 and are incremented by 1.

Advanced counters

You can also specify the initial counter and increment:

```
>>> tpl = Template.create('ID2', '{counter:02d}-file', start=10, step=10)
>>> seq = Sequence(tpl)
>>> seq.next()
'10-file'
>>> seq.next()
'20-file'
>>> seq.next()
'30-file'
```

User-defined keywords

Consider the case where you need to generate identifiers for files of different categories. Thus, your template definition should look like this:

```
>>> cat = Template.create('KW', '{category}: File {counter:03d}', start=1)
```

Next, instantiate the template for specific categories:

Note: Keyword arguments must be always specified on instantiation.

```
>>> photos = Sequence(cat, category='PHOTOS')
>>> photos.next()
'PHOTOS: File 001'
>>> photos.next()
'PHOTOS: File 002'
>>> videos = Sequence(cat, category='VIDEOS')
>>> videos.next()
'VIDEOS: File 001'
```

```
>>> videos.next()
'VIDEOS: File 002'
```

```
>>> Sequence('invalid')
Traceback (most recent call last):
 ...
SequenceNotFound
```

```
>>> invalid = Sequence(cat, invalid='PHOTOS')
>>> invalid.next()
Traceback (most recent call last):
 ...
KeyError
```

Hierarchical identifiers

It is possible to have nested templates (i.e. templates depending on another). This is achieved by placing a placeholder with the name of the parent template and making sure a valid identifier of the parent is passed on each instantiation.

Consider the example of playlists of audio files for each year. We need two template definitions, one for playlists and one for files:

```
>>> pl = Template.create('PL', '{year}: Playlist {counter}', start=1)
>>> fl = Template.create('FL', '{PL} > Audio File {counter:02d}', start=1)
```

Let's get some playlists for different years:

```
>>> pl15 = Sequence(pl, year=2015)
>>> pl15.next()
'2015: Playlist 1'
>>> pl15.next()
'2015: Playlist 2'
>>> pl16 = Sequence(pl, year=2016)
>>> pl16.next()
'2016: Playlist 1'
```

Now let's get some files inside the playlists generated above:

```
>>> fl15 = Sequence(fl, PL='2015: Playlist 2')
>>> fl15.next()
'2015: Playlist 2 > Audio File 01'
>>> fl15.next()
'2015: Playlist 2 > Audio File 02'
>>> fl16 = Sequence(fl, PL='2016: Playlist 1')
>>> fl16.next()
'2016: Playlist 1 > Audio File 01'
```

Bulk generations

As Sequence is a proper Python iterator, you can use all Python methods that consume them (mainly found on the itertools library). Picking up from the last example with playlists, one might want to generate multiple files for a specific playlist, all at once:

```
>>> from itertools import islice
>>> list(islice(fl15, 5))
['2016: Playlist 2 > Audio File 03', '2016: Playlist 2 > Audio File 04',
'2016: Playlist 2 > Audio File 05', '2016: Playlist 2 > Audio File 06',
'2016: Playlist 2 > Audio File 07']
```

Command-line interface

It is possible to use the command-line interface, instead of using the API directly. As you would expect, one can define new templates and get new identifiers from sequences.

To get started, we need to first setup the example app:

```
$ cd examples
$ export FLASK_APP=app.py
$ flask db init
$ flask db create
```

Let's see the example introduced in *Simple counters*, using the CLI this time:

```
$ flask sequences create ID '{counter}-file'
$ flask sequences next ID
0-file
$ flask sequences next ID
1-file
$ flask sequences next ID
2-file
```

Now let's see a more complicated example, such as the one shown in *Hierarchical identifiers*:

```
$ flask sequences create PL '{year}: Playlist {counter}' --start 1
$ flask sequences create FL '{PL} > Audio File {counter:02d}' --start 1
$ flask sequences next PL year=2015
2015: Playlist 1
$ flask sequences next PL year=2015
2015: Playlist 2
$ flask sequences next PL year=2016
2016: Playlist 1
$ flask sequences next FL PL='2015: Playlist 2'
2015: Playlist 2 > Audio File 01
$ flask sequences next FL PL='2015: Playlist 2'
2015: Playlist 2 > Audio File 02
$ flask sequences next FL PL='2016: Playlist 1'
2016: Playlist 1 > Audio File 01
```

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.

API Docs

Models

CHAPTER 3

Additional Notes

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

Contributing

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

Types of Contributions

Report Bugs

Report bugs at https://github.com/inveniosoftware/invenio-pidstore-reportnumber/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-PIDStore-ReportNumber could always use more documentation, whether as part of the official Invenio-PIDStore-ReportNumber 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-pidstore-reportnumber/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 :)

Get Started!

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

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

```
$ git clone git@github.com:your_name_here/invenio-pidstore-reportnumber.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-pidstore-reportnumber
$ cd invenio-pidstore-reportnumber/
$ 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.

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.org/inveniosoftware/invenio-pidstore-reportnumber/pull_requests and make sure that the tests pass for all supported Python versions.

Changes

Version 1.0.0a2 (released August 18, 2017)

- Initial public release.

License

Authors

Invenio module for generating sequences.

- Esteban J. G. Gabancho <egabancho@gmail.com>
- Javier Martin Montull <javier.martin.montull@cern.ch>
- Jiri Kuncar <jiri.kuncar@cern.ch>
- Lars Holm Nielsen <lars.holm.nielsen@cern.ch>
- Leonardo Rossi <leonardo.r@cern.ch>
- Orestis Melkonian <melkon.or@gmail.com>
- Samuele Kaplun <samuele.kaplun@cern.ch>
- Tibor Simko <tibor.simko@cern.ch>

Python Module Index

Index