invenio-pidstore Documentation

Release 1.0.0

CERN

Dec 04, 2018

Contents

1 User's Guide **3**
- 1.1 Installation . . . 3
- 1.2 Configuration . . . 3
- 1.3 Usage . . . 3
- 1.4 Example application . . . 9

2 API Reference **11**
- 2.1 API Docs . . . 11

3 Additional Notes **23**
- 3.1 Contributing . . . 23
- 3.2 Changes . . . 25
- 3.3 License . . . 25
- 3.4 Contributors . . . 25

Python Module Index **27**

Invenio module that mints, stores, registers and resolves persistent identifiers.

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

CHAPTER 1

User's Guide

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

1.1 Installation

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

```
$ pip install Invenio-PIDStore
```

Invenio-PIDStore depends on Invenio-DB.

1.2 Configuration

Invenio-PIDStore configuration.

invenio_pidstore.config.**PIDSTORE_RECID_FIELD = 'control_number'**
Default record id field inside the json data.

This name will be used by the fetcher, to retrieve the record ID value from the JSON, and by the minter, to store it inside the JSON.

1.3 Usage

Module that mints, stores, registers and resolves persistent identifiers.

Invenio-PIDStore is a core component of Invenio responsible for managing a mapping from persistent identifiers to internal objects. The mapping is used to resolve requests for e.g. an external integer record identifier to an internal metadata record.

PIDStore consists of:

- *Persistent identifier API* - for working with persistent identifiers (create, reserve, register, delete, and redirect).
- *Resolver* - given a persistent identifier retrieve the assigned internal object.
- *Providers* - wrappers around a persistent identifier to provide extra functionality (e.g. interaction with remote services).
- *Minters* - small functions that are responsible for minting a specific persistent identifier type for a specific internal object type in as automatic way as possible.
- *Fetchers* - small functions that are responsible for returning a minted persistent identifier.

1.3.1 Initialization

First create a Flask application with a Click support (Flask version 0.11+):

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

You initialize PIDStore like a normal Flask extension, however Invenio-PIDStore is dependent on Invenio-DB so you need to initialize both extensions:

```
>>> from invenio_db import InvenioDB
>>> from invenio_pidstore import InvenioPIDStore
>>> ext_db = InvenioDB(app)
>>> ext_pidstore = InvenioPIDStore(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()
```

1.3.2 Persistent identifiers

Persistent identifiers map an external persistent identifier into an internal object.

- A persistent identifier has **zero or one** assigned internal object.
- An internal object has **one or many** persistent identifiers.
- PIDStore can only map to internal objects which are **identified by UUIDs**.
- Persistent identifiers have a **type** (scheme) which is identified by a 6 character string.
- All persistent identifier values are stored as string values, even if they may have a data type such as integer or UUID.

Persistent identifiers have a **state**:

- New: The persistent identifier is new.
- Reserved: The persistent identifier is reserved.

- Registered: The persistent identifier is registered.
- Redirected: The persistent identifier is redirected to another persistent identifier.
- Deleted: The persistent identifier is deleted/inactivated.

The different states might not all be relevant for different types of identifiers. In addition persistent identifiers may be associated with a specific **provider** (see below).

Creating PIDs

You can create a persistent identifier like this:

```
>>> from invenio_pidstore.models import PersistentIdentifier
>>> pid = PersistentIdentifier.create('doi', '10.1234/foo')
```

By default, persistent identifiers are created in the *new* state, and has no object assigned:

```
>>> from invenio_pidstore.models import PIDStatus
>>> pid.status == PIDStatus.NEW
True
>>> pid.has_object()
False
```

Assign objects

You can either assign objects after having created the persistent identifier:

```
>>> import uuid
>>> pid.assign('rec', uuid.uuid4())
True
```

or alternatively simply assign the object while you create the persistent identifier (and also set the correct status immediately):

```
>>> pid = PersistentIdentifier.create('recid', '10',
...     object_type='rec', object_uuid=uuid.uuid4(),
...     status=PIDStatus.REGISTERED)
>>> pid.has_object()
True
```

Note, that once an object has been assigned you cannot assign a new object unless you use the `overwrite` parameter:

```
>>> pid.assign('rec', uuid.uuid4())
Traceback (most recent call last):
...
invenio_pidstore.errors.PIDObjectAlreadyAssigned: ...
>>> pid.assign('rec', uuid.uuid4(), overwrite=True)
True
```

Modifying status

You can modify the state of a persistent identifier:

```
>>> pid = PersistentIdentifier.create('doi', '10.1234/bar')
>>> pid.reserve()
True
>>> pid.register()
True
>>> pid.delete()
True
```

Note that if you delete a persistent identifier in state *new* it will be completely removed from the database. Also, certain state changes are not allowed (e.g. here trying to reserve a deleted persistent identifier):

```
>>> pid.reserve()
Traceback (most recent call last):
...
invenio_pidstore.errors.PIDInvalidAction: ...
```

Redirecting

You can redirect a registered persistent identifier to another persistent identifier. This can be useful in cases where you merge two underlying records or you have multiple identifier schemes but one is preferred over another.

```
>>> pid = PersistentIdentifier.create('recid', '11',
...     status=PIDStatus.REGISTERED)
>>> pid.redirect(PersistentIdentifier.get('recid', '10'))
True
```

This allows you easily retrieve the persistent identifier being redirected to:

```
>>> pid.get_redirect()
<PersistentIdentifier recid:10 / rec:... (R)>
```

1.3.3 Resolver

The resolver is responsible for retrieving an internal object for a given persistent identifier.

Following is an example for creating a record resolver.

```
>>> from invenio_pidstore.resolver import Resolver
>>> records = {}
>>> resolver = Resolver(pid_type='recid', object_type='rec',
...    getter=records.get)
```

Above creates a resolver that will translate `recid` persistent identifiers to internal records. The getter argument must be a callable that takes one argument which is the persistent identifiers object UUID.

Before we use the resolver, let's first create record and a persistent identifier that we can use to test the resolver with:

```
>>> my_object_uuid = uuid.uuid4()
>>> records[my_object_uuid] = {'title': 'PIDStore test'}
>>> pid = PersistentIdentifier.create(
...     'recid', '12', object_type='rec', object_uuid=my_object_uuid,
...     status=PIDStatus.REGISTERED)
```

Using the resolver, we can now retrieve the record like this:

```
>>> pid, record = resolver.resolve('12')
>>> pid
<PersistentIdentifier recid:12 / rec:... (R)>
>>> record
{'title': 'PIDStore test'}
```

The resolver will only resolve **registered** persistent identifiers. Any non-resolvable value will raise an exception. E.g. trying to resolve a reserved persistent identifier will throw a *invenio_pidstore.errors.PIDUnregistered* exception:

```
>>> pid = PersistentIdentifier.create(
...     'recid', '13', object_type='rec', object_uuid=my_object_uuid,
...     status=PIDStatus.RESERVED)
>>> resolver.resolve('13')
Traceback (most recent call last):
...
invenio_pidstore.errors.PIDUnregistered: ...
```

1.3.4 Providers

Providers adds extra functionality persistent identifiers. Use cases for this includes automatically creating the persistent identifier or retrieving the persistent identifier from an external service.

PIDStore comes by default with a *invenio_pidstore.providers.recordid.RecordIdProvider* which will create Invenio legacy integer record identifiers:

```
>>> from invenio_pidstore.providers.recordid import RecordIdProvider
>>> provider = RecordIdProvider.create()
>>> provider.pid.pid_type
'recid'
>>> provider.pid.pid_value
'1'
```

Creating your own provider

In order to create your own provider, you need to inherit from *invenio_pidstore.providers.base.BaseProvider* and override the methods you want to customize.

Here's a minimal example:

```
>>> from invenio_pidstore.providers.base import BaseProvider
>>> class MyProvider(BaseProvider):
...     pid_type = 'my'
...     pid_provider = 'test'
...     default_status = PIDStatus.RESERVED
```

1.3.5 Minters

Minters are small functions that are responsible for minting a specific persistent identifier type for a specific internal object type in as automatic way as possible.

A minter takes two arguments:

- an *object uuid* of the internal object which the persistent identifier should be assigned to.

- a *dictionary object*, which the minter may extract information from, and/or store extra information in. For instance, if a record object is passed, it might contain the DOI which a provider should mint, or alternatively, if the persistent identifier is generated, the minter can store the generated value into the record.

Below is an example minter which mints external UUIDs (not to be confused with the internal object UUIDs), and stores the generated UUID into the dictionary object (which could be e.g. a record):

```
>>> import uuid
>>> def my_uuid_minter(object_uuid, data):
...     pid = PersistentIdentifier.create('myuuid', str(uuid.uuid4()))
...     data['uuid'] = str(pid.object_uuid)
...     return pid
```

Registering minters

Minters are usually used by other modules and thus registered on the Flask application.

First import the proxy to the current application's PIDStore:

```
>>> from invenio_pidstore import current_pidstore
```

Next, register a minter:

```
>>> current_pidstore.register_minter('my_uuid_minter', my_uuid_minter)
```

You can retrieve registered minters from the extension as well:

```
>>> current_pidstore.minters['my_uuid_minter']
<function my_uuid_minter at ...>
```

1.3.6 Fetchers

Fetchers are small functions that are responsible for returning a minted persistent identifier.

A fetcher takes two arguments:

- an *object uuid* of the internal object which the persistent identifier should be assigned to.
- a *dictionary object*, which the fetcher may extract information from. For instance, if a record object is passed, it might contain a previously minted DOI.

Below is an example fetcher which extracts external UUIDs previously minted in the dictionary object (which could be e.g. a record):

```
>>> import uuid
>>> def my_uuid_fetcher(object_uuid, data):
...     return data['uuid']
```

Registering fetchers

Fetchers are usually used by other modules and thus registered on the Flask application.

First import the proxy to the current application's PIDStore:

```
>>> from invenio_pidstore import current_pidstore
```

Next, register a fetcher:

```
>>> current_pidstore.register_fetcher('my_uuid_fetcher', my_uuid_fetcher)
```

You can retrieve registered fetchers from the extension as well:

```
>>> current_pidstore.fetchers['my_uuid_fetcher']
<function my_uuid_fetcher at ...>
```

Entry points loading

PIDStore will automatically register minters and fetchers defined by the entry point groups `invenio_pidstore.minters` and `invenio_pidstore.fetchers`.

Example:

```
# setup.py
setup(
    # ...
    entry_points={
        'invenio_pidstore.minters': [
            'recid = invenio_pidstore.minters:recid_minter',
        ],
        'invenio_pidstore.fetchers': [
            'recid = invenio_pidstore.fetchers:recid_fetcher',
        ]})
```

Above is equivalent to:

```
from invenio_pidstore.minters import recid_minter
from invenio_pidstore.fetchers import recid_fetcher
current_pidstore.register_minter('recid', recid_minter)
current_pidstore.register_fetcher('recid', recid_fetcher)
```

1.4 Example application

First install Invenio-PIDStore, setup the application and load fixture data by running:

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

Next, start the development server:

```
$ export FLASK_APP=app.py FLASK_DEBUG=1
$ flask run
```

Open the admin page:

```
$ open http://127.0.0.1:5000/admin/recordmetadata/
```

Login with:

username: admin@inveniosoftware.org

password: 123456

To reset the example application run:

```
$ ./app-teardown.sh
```

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

2.1.1 Extension

Invenio module that stores and registers persistent identifiers.

class invenio_pidstore.ext.**InvenioPIDStore**(*app=None*, *minters_entry_point_group='invenio_pidstore.minters'*, *fetchers_entry_point_group='invenio_pidstore.fetchers'*)

Invenio-PIDStore extension.

Extension initialization.

Parameters

- **minters_entry_point_group** – The entrypoint for minters. (Default: *invenio_pidstore.minters*).
- **fetchers_entry_point_group** – The entrypoint for fetchers. (Default: *invenio_pidstore.fetchers*).

init_app(*app*, *minters_entry_point_group=None*, *fetchers_entry_point_group=None*)

Flask application initialization.

Initialize:

- The CLI commands.
- Initialize the logger (Default: *app.debug*).
- **Initialize the default admin object link endpoint.** (Default: *{"rec": "recordmetadata.details_view"}* if *invenio-records* is installed, otherwise *{}*).
- Register the *pid_exists* template filter.

- Initialize extension state.

  Parameters

  - **app** – The Flask application
  - **minters_entry_point_group** – The minters entry point group (Default: None).
  - **fetchers_entry_point_group** – The fetchers entry point group (Default: None).

  Returns PIDStore state application.

init_config(*app*)
Initialize configuration.

invenio_pidstore.ext.**pid_exists**(*value*, *pidtype=None*)
Check if a persistent identifier exists.

Parameters

- **value** – The PID value.
- **pidtype** – The pid value (Default: None).

Returns *True* if the PID exists.

2.1.2 Models

Persistent identifier store and registration.

class invenio_pidstore.models.**PersistentIdentifier**(***kwargs*)
Store and register persistent identifiers.

Assumptions:

- Persistent identifiers can be represented as a string of max 255 chars.
- An object has many persistent identifiers.
- A persistent identifier has one and only one 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.

assign(*object_type*, *object_uuid*, *overwrite=False*)
Assign this persistent identifier to a given object.

Note, the persistent identifier must first have been reserved. Also, if an existing object is already assigned to the pid, it will raise an exception unless overwrite=True.

Parameters

- **object_type** – The object type is a string that identify its type.
- **object_uuid** – The object UUID.
- **overwrite** – Force PID overwrites in case was previously assigned.

Raises

- ***invenio_pidstore.errors.PIDInvalidAction*** – If the PID was previously deleted.

- **invenio_pidstore.errors.PIDObjectAlreadyAssigned** – If the PID was previously assigned with a different type/uuid.

Returns *True* if the PID is successfully assigned.

classmethod create(*pid_type*, *pid_value*, *pid_provider=None*, *status=<PIDStatus.NEW: 'N'>*, *object_type=None*, *object_uuid=None*)

Create a new persistent identifier with specific type and value.

Parameters

- **pid_type** – Persistent identifier type.
- **pid_value** – Persistent identifier value.
- **pid_provider** – Persistent identifier provider. (default: None).
- **status** – Current PID status. (Default: *invenio_pidstore.models.PIDStatus.NEW*)
- **object_type** – The object type is a string that identify its type. (default: None).
- **object_uuid** – The object UUID. (default: None).

Returns A *invenio_pidstore.models.PersistentIdentifier* instance.

delete()

Delete the persistent identifier.

If the persistent identifier haven't been registered yet, it is removed from the database. Otherwise, it's marked as *invenio_pidstore.models.PIDStatus.DELETED*.

Returns *True* if the PID is successfully removed.

classmethod get(*pid_type*, *pid_value*, *pid_provider=None*)

Get persistent identifier.

Parameters

- **pid_type** – Persistent identifier type.
- **pid_value** – Persistent identifier value.
- **pid_provider** – Persistent identifier provider. (default: None).

Raises *invenio_pidstore.errors.PIDDoesNotExistError* if no PID is found.

Returns A *invenio_pidstore.models.PersistentIdentifier* instance.

get_assigned_object(*object_type=None*)

Return the current assigned object UUID.

Parameters **object_type** – If it's specified, returns only if the PID object_type is the same, otherwise returns None. (default: None).

Returns The object UUID.

classmethod get_by_object(*pid_type*, *object_type*, *object_uuid*)

Get a persistent identifier for a given object.

Parameters

- **pid_type** – Persistent identifier type.
- **object_type** – The object type is a string that identify its type.
- **object_uuid** – The object UUID.

Raises **invenio_pidstore.errors.PIDDoesNotExistError** – If no PID is found.

Returns A *invenio_pidstore.models.PersistentIdentifier* instance.

get_redirect()
Get redirected persistent identifier.

Returns The *invenio_pidstore.models.PersistentIdentifier* instance.

has_object()
Determine if this PID has an assigned object.

Returns *True* if the PID has a object assigned.

id
Id of persistent identifier entry.

is_deleted()
Return true if the persistent identifier has been deleted.

Returns A boolean value.

is_new()
Return true if the PIDhas not yet been registered or reserved.

Returns A boolean value.

is_redirected()
Return true if the persistent identifier has been registered.

is_registered()
Return true if the persistent identifier has been registered.

Returns A *invenio_pidstore.models.PIDStatus* status.

is_reserved()
Return true if the PID has not yet been reserved.

Returns A boolean value.

object_type
Object Type - e.g. rec for record.

object_uuid
Object ID - e.g. a record id.

pid_provider
Persistent Identifier Provider

pid_type
Persistent Identifier Schema.

pid_value
Persistent Identifier.

redirect(*pid*)
Redirect persistent identifier to another persistent identifier.

Parameters **pid** – The *invenio_pidstore.models.PersistentIdentifier* where redirect the PID.

Raises

- ***invenio_pidstore.errors.PIDInvalidAction*** – If the PID is not registered or is not already redirecting to another PID.
- ***invenio_pidstore.errors.PIDDoesNotExistError*** – If PID is not found.

Returns *True* if the PID is successfully redirect.

register()
Register the persistent identifier with the provider.

Raises ***invenio_pidstore.errors.PIDInvalidAction*** – If the PID is not already registered or is deleted or is a redirection to another PID.

Returns *True* if the PID is successfully register.

reserve()
Reserve the persistent identifier.

Note, the reserve method may be called multiple times, even if it was already reserved.

Raises *invenio_pidstore.errors.PIDInvalidAction* if the PID is not new or is not already reserved a PID.

Returns *True* if the PID is successfully reserved.

status
Status of persistent identifier, e.g. registered, reserved, deleted.

sync_status(*status*)
Synchronize persistent identifier status.

Used when the provider uses an external service, which might have been modified outside of our system.

Parameters **status** – The new status to set.

Returns *True* if the PID is successfully sync.

unassign()
Unassign the registered object.

Note: Only registered PIDs can be redirected so we set it back to registered.

Returns *True* if the PID is successfully unassigned.

class invenio_pidstore.models.**PIDStatus**(*value*)
Constants for possible status of any given PID.

Hack.

DELETED = 'D'
PID has been deleted/inactivated with the service provider.

This should happen very rarely, and must be kept track of, as the PID should not be reused for something else.

NEW = 'N'
PID has *not* yet been registered with the service provider.

REDIRECTED = 'M'
PID has been redirected to another persistent identifier.

REGISTERED = 'R'
PID has been registered with the service provider.

RESERVED = 'K'
PID reserved in the service provider but not yet fully registered.

class invenio_pidstore.models.**RecordIdentifier**(***kwargs*)
Sequence generator for integer record identifiers.

The sole purpose of this model is to generate integer record identifiers in sequence using the underlying database's auto increment features in a transaction friendly manner. The feature is primarily provided to support legacy Invenio instances to continue their current record identifier scheme. For new instances we strong encourage to not use auto incrementing record identifiers, but instead use e.g. UUIDs as record identifiers.

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 insert(*val*)
Insert a record identifier.

Parameters **val** – The *recid* column value to insert.

classmethod max()
Get max record identifier.

classmethod next()
Return next available record identifier.

class invenio_pidstore.models.**Redirect**(***kwargs*)
Redirect for a persistent identifier.

You can redirect a PID to another one.

E.g.

```
pid1 = PersistentIdentifier.get(pid_type="recid", pid_value="1")
pid2 = PersistentIdentifier.get(pid_type="recid", pid_value="2")
pid1.redirect(pid=pid2)
assert pid2.pid_value == pid.get_redirect().pid_value
```

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.

id
Id of redirect entry.

pid
Relationship to persistent identifier.

pid_id
Persistent identifier.

2.1.3 Resolver

Internal resolver for persistent identifiers.

class invenio_pidstore.resolver.**Resolver**(*pid_type=None*, *object_type=None*, *getter=None*)
Persistent identifier resolver.

Helper class for retrieving an internal object for a given persistent identifier.

Initialize resolver.

Parameters

- **pid_type** – Persistent identifier type.
- **object_type** – Object type.
- **getter** – Callable that will take an object id for the given object type and retrieve the internal object.

resolve(*pid_value*)
Resolve a persistent identifier to an internal object.

Parameters **pid_value** – Persistent identifier.

Returns A tuple containing (pid, object).

2.1.4 Providers

Implementations of different PID providers.

Module storing implementations of PID providers.

class invenio_pidstore.providers.base.**BaseProvider**(*pid*)
Abstract class for persistent identifier provider classes.

Initialize provider using persistent identifier.

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

classmethod create(*pid_type=None*, *pid_value=None*, *object_type=None*, *object_uuid=None*, *status=None*, ***kwargs*)
Create a new instance for the given type and pid.

Parameters

- **pid_type** – Persistent identifier type. (Default: None).
- **pid_value** – Persistent identifier value. (Default: None).
- **status** – Current PID status. (Default: *invenio_pidstore.models.PIDStatus.NEW*)
- **object_type** – The object type is a string that identify its type. (Default: None).
- **object_uuid** – The object UUID. (Default: None).

Returns A *invenio_pidstore.providers.base.BaseProvider* instance.

default_status = 'N'
Default status for newly created PIDs by this provider.

delete()
Delete a persistent identifier.

See: *invenio_pidstore.models.PersistentIdentifier.delete()*.

classmethod get(*pid_value*, *pid_type=None*, ***kwargs*)
Get a persistent identifier for this provider.

Parameters

- **pid_type** – Persistent identifier type. (Default: configured *invenio_pidstore.providers.base.BaseProvider.pid_type*)
- **pid_value** – Persistent identifier value.

- **kwargs** – See *invenio_pidstore.providers.base.BaseProvider()* required initialization properties.

Returns A *invenio_pidstore.providers.base.BaseProvider* instance.

pid_provider = None
Persistent identifier provider name.

pid_type = None
Default persistent identifier type.

register()
Register a persistent identifier.

See: *invenio_pidstore.models.PersistentIdentifier.register()*.

reserve()
Reserve a persistent identifier.

This might or might not be useful depending on the service of the provider.

See: *invenio_pidstore.models.PersistentIdentifier.reserve()*.

sync_status()
Synchronize PIDstatus with remote service provider.

update()
Update information about the persistent identifier.

Record ID provider.

class invenio_pidstore.providers.recordid.**RecordIdProvider**(*pid*)
Record 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 record identifier.

Note: if the object_type and object_uuid values are passed, then the PID status will be automatically setted to *invenio_pidstore.models.PIDStatus.REGISTERED*.

Parameters

- **object_type** – The object type. (Default: None.)
- **object_uuid** – The object identifier. (Default: None).
- **kwargs** – You specify the pid_value.

default_status = 'K'
Record IDs are by default registered immediately.

Default: *invenio_pidstore.models.PIDStatus.RESERVED*

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 record ids.

pid_type = 'recid'
Type of persistent identifier.

2.1.5 Minters

Persistent identifier minters.

invenio_pidstore.minters.**recid_minter**(*record_uuid*, *data*)

Mint record identifiers.

This is a minter specific for records. With the help of *invenio_pidstore.providers.recordid.RecordIdProvider*, it creates the PID instance with *rec* as predefined *object_type*.

Procedure followed: (we will use *control_number* as value of *PIDSTORE_RECID_FIELD* for the simplicity of the documentation.)

#. If a *control_number* field is already there, a *AssertionError* exception is raised.

#. The provider is initialized with the help of *invenio_pidstore.providers.recordid.RecordIdProvider*. It's called with default value 'rec' for *object_type* and *record_uuid* variable for *object_uuid*.

1. The new *id_value* is stored inside *data* as *control_number* field.

Parameters

- **record_uuid** – The record UUID.
- **data** – The record metadata.

Returns A fresh *invenio_pidstore.models.PersistentIdentifier* instance.

2.1.6 Fetchers

Persistent identifier fetchers.

A proper fetcher is defined as a function that return a *invenio_pidstore.fetchers.FetchedPID* instance.

E.g.

```
def my_fetcher(record_uuid, data):
    return FetchedPID(
        provider=MyRecordIdProvider,
        pid_type=MyRecordIdProvider.pid_type,
        pid_value=extract_pid_value(data),
    )
```

To see more about providers see *invenio_pidstore.providers*.

class invenio_pidstore.fetchers.**FetchedPID**(*provider*, *pid_type*, *pid_value*)

A pid fetcher.

pid_type
Alias for field number 1

pid_value
Alias for field number 2

provider
Alias for field number 0

invenio_pidstore.fetchers.**recid_fetcher**(*record_uuid*, *data*)

Fetch a record's identifiers.

Parameters

- **record_uuid** – The record UUID.
- **data** – The record metadata.

Returns A *invenio_pidstore.fetchers.FetchedPID* instance.

2.1.7 Exceptions

Errors for persistent identifiers.

exception invenio_pidstore.errors.**PIDAlreadyExists**(*pid_type*, *pid_value*, **args*, ***kwargs*)
Persistent identifier already exists error.

Initialize exception.

exception invenio_pidstore.errors.**PIDDeletedError**(*pid*, *record*, **args*, ***kwargs*)
Persistent identifier is deleted.

Initialize exception.

exception invenio_pidstore.errors.**PIDDoesNotExistError**(*pid_type*, *pid_value*, **args*, ***kwargs*)
PID does not exists error.

Initialize exception.

exception invenio_pidstore.errors.**PIDInvalidAction**
Invalid operation on persistent identifier in current state.

exception invenio_pidstore.errors.**PIDMissingObjectError**(*pid*, **args*, ***kwargs*)
Persistent identifier has no object.

Initialize exception.

exception invenio_pidstore.errors.**PIDObjectAlreadyAssigned**
Persistent identifier is already assigned to another object.

exception invenio_pidstore.errors.**PIDRedirectedError**(*pid*, *dest_pid*, **args*, ***kwargs*)
Persistent identifier is redirected to another pid.

Initialize exception.

exception invenio_pidstore.errors.**PIDUnregistered**(*pid*, **args*, ***kwargs*)
Persistent identifier has not been registered.

Initialize exception.

exception invenio_pidstore.errors.**PIDValueError**(*pid_type*, *pid_value*, **args*, ***kwargs*)
Base class for value errors.

Initialize exception.

exception invenio_pidstore.errors.**PersistentIdentifierError**
Base class for PIDStore errors.

exception invenio_pidstore.errors.**ResolverError**(*pid*, **args*, ***kwargs*)
Persistent identifier does not exists.

Initialize exception.

2.1.8 CLI

Detailed usage documentation is available by running `inveniomanage pid --help`.

Click command-line interface for PIDStore management.

`invenio_pidstore.cli.`**`process_status`**(*ctx*, *param*, *value*)
Return status value.

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-pidstore/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 could always use more documentation, whether as part of the official Invenio-PIDStore 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/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-pidstore* for local development.

1. Fork the *inveniosoftware/invenio-pidstore* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/invenio-pidstore.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
$ cd invenio-pidstore/
$ 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, 3.3, 3.4 and 3.5. Check https://travis-ci.org/inveniosoftware/invenio-pidstore/pull_requests and make sure that the tests pass for all supported Python versions.

3.2 Changes

Version 1.0.0 (released 2018-03-23)

- Initial public release.

3.3 License

3.4 Contributors

- Alexander Ioannidis
- Alizee Pace
- Diego Rodriguez
- Esteban J. G. Gabancho
- Harri Hirvonsalo

- Jiri Kuncar
- Krzysztof Nowak
- Lars Holm Nielsen
- Leonardo Rossi
- Nicolas Harraudeau
- Orestis Melkonian
- Paulina Lach
- Sami Hiltunen
- Tibor Simko

Python Module Index

i

invenio_pidstore, 3
invenio_pidstore.cli, 21
invenio_pidstore.config, 3
invenio_pidstore.errors, 20
invenio_pidstore.ext, 11
invenio_pidstore.fetchers, 19
invenio_pidstore.minters, 19
invenio_pidstore.models, 12
invenio_pidstore.providers, 17
invenio_pidstore.providers.base, 17
invenio_pidstore.providers.recordid, 18
invenio_pidstore.resolver, 16

Index

A

assign() (invenio_pidstore.models.PersistentIdentifier method), 12

B

BaseProvider (class in invenio_pidstore.providers.base), 17

C

create() (invenio_pidstore.models.PersistentIdentifier class method), 13
create() (invenio_pidstore.providers.base.BaseProvider class method), 17
create() (invenio_pidstore.providers.recordid.RecordIdProvider class method), 18

D

default_status (invenio_pidstore.providers.base.BaseProvider attribute), 17
default_status (invenio_pidstore.providers.recordid.RecordIdProvider attribute), 18
delete() (invenio_pidstore.models.PersistentIdentifier method), 13
delete() (invenio_pidstore.providers.base.BaseProvider method), 17
DELETED (invenio_pidstore.models.PIDStatus attribute), 15

F

FetchedPID (class in invenio_pidstore.fetchers), 19

G

get() (invenio_pidstore.models.PersistentIdentifier class method), 13
get() (invenio_pidstore.providers.base.BaseProvider class method), 17
get_assigned_object() (invenio_pidstore.models.PersistentIdentifier method), 13
get_by_object() (invenio_pidstore.models.PersistentIdentifier class method), 13
get_redirect() (invenio_pidstore.models.PersistentIdentifier method), 14

H

has_object() (invenio_pidstore.models.PersistentIdentifier method), 14

I

id (invenio_pidstore.models.PersistentIdentifier attribute), 14
id (invenio_pidstore.models.Redirect attribute), 16
init_app() (invenio_pidstore.ext.InvenioPIDStore method), 11
init_config() (invenio_pidstore.ext.InvenioPIDStore method), 12
insert() (invenio_pidstore.models.RecordIdentifier class method), 16
invenio_pidstore (module), 3
invenio_pidstore.cli (module), 21
invenio_pidstore.config (module), 3
invenio_pidstore.errors (module), 20
invenio_pidstore.ext (module), 11
invenio_pidstore.fetchers (module), 19
invenio_pidstore.minters (module), 19
invenio_pidstore.models (module), 12
invenio_pidstore.providers (module), 17
invenio_pidstore.providers.base (module), 17
invenio_pidstore.providers.recordid (module), 18
invenio_pidstore.resolver (module), 16
InvenioPIDStore (class in invenio_pidstore.ext), 11
is_deleted() (invenio_pidstore.models.PersistentIdentifier method), 14
is_new() (invenio_pidstore.models.PersistentIdentifier method), 14
is_redirected() (invenio_pidstore.models.PersistentIdentifier method), 14
is_registered() (invenio_pidstore.models.PersistentIdentifier method), 14

is_reserved() (invenio_pidstore.models.PersistentIdentifier method), 14

M

max() (invenio_pidstore.models.RecordIdentifier class method), 16

N

NEW (invenio_pidstore.models.PIDStatus attribute), 15
next() (invenio_pidstore.models.RecordIdentifier class method), 16

O

object_type (invenio_pidstore.models.PersistentIdentifier attribute), 14
object_uuid (invenio_pidstore.models.PersistentIdentifier attribute), 14

P

PersistentIdentifier (class in invenio_pidstore.models), 12
PersistentIdentifierError, 20
pid (invenio_pidstore.models.Redirect attribute), 16
pid_exists() (in module invenio_pidstore.ext), 12
pid_id (invenio_pidstore.models.Redirect attribute), 16
pid_provider (invenio_pidstore.models.PersistentIdentifier attribute), 14
pid_provider (invenio_pidstore.providers.base.BaseProvider attribute), 18
pid_provider (invenio_pidstore.providers.recordid.RecordIdProvider attribute), 18
pid_type (invenio_pidstore.fetchers.FetchedPID attribute), 19
pid_type (invenio_pidstore.models.PersistentIdentifier attribute), 14
pid_type (invenio_pidstore.providers.base.BaseProvider attribute), 18
pid_type (invenio_pidstore.providers.recordid.RecordIdProvider attribute), 18
pid_value (invenio_pidstore.fetchers.FetchedPID attribute), 19
pid_value (invenio_pidstore.models.PersistentIdentifier attribute), 14
PIDAlreadyExists, 20
PIDDeletedError, 20
PIDDoesNotExistError, 20
PIDInvalidAction, 20
PIDMissingObjectError, 20
PIDObjectAlreadyAssigned, 20
PIDRedirectedError, 20
PIDStatus (class in invenio_pidstore.models), 15
PIDSTORE_RECID_FIELD (in module invenio_pidstore.config), 3
PIDUnregistered, 20
PIDValueError, 20
process_status() (in module invenio_pidstore.cli), 21
provider (invenio_pidstore.fetchers.FetchedPID attribute), 19

R

recid_fetcher() (in module invenio_pidstore.fetchers), 19
recid_minter() (in module invenio_pidstore.minters), 19
RecordIdentifier (class in invenio_pidstore.models), 15
RecordIdProvider (class in invenio_pidstore.providers.recordid), 18
Redirect (class in invenio_pidstore.models), 16
redirect() (invenio_pidstore.models.PersistentIdentifier method), 14
REDIRECTED (invenio_pidstore.models.PIDStatus attribute), 15
register() (invenio_pidstore.models.PersistentIdentifier method), 15
register() (invenio_pidstore.providers.base.BaseProvider method), 18
REGISTERED (invenio_pidstore.models.PIDStatus attribute), 15
reserve() (invenio_pidstore.models.PersistentIdentifier method), 15
reserve() (invenio_pidstore.providers.base.BaseProvider method), 18
RESERVED (invenio_pidstore.models.PIDStatus attribute), 15
resolve() (invenio_pidstore.resolver.Resolver method), 17
Resolver (class in invenio_pidstore.resolver), 16
ResolverError, 20

S

status (invenio_pidstore.models.PersistentIdentifier attribute), 15
sync_status() (invenio_pidstore.models.PersistentIdentifier method), 15
sync_status() (invenio_pidstore.providers.base.BaseProvider method), 18

U

unassign() (invenio_pidstore.models.PersistentIdentifier method), 15
update() (invenio_pidstore.providers.base.BaseProvider method), 18