invenio-explicit-acls Documentation

Release 4.5.0

UCT Prague

Apr 29, 2021

Contents

A package that adds support for elasticsearch-executed declarative ACLs. For rationale see slide 27 of https://indico.cern.ch/event/773969/contributions/3351844/attachments/1814264/2967405/Access_management-Karolina.pdf

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

CHAPTER 1

User's Guide

At first **read the principles** section to find out if this library is usable for your use cases.

Do not skip steps in the configuration section - if you skip any of them your application will be insecure.

As nobody is perfect, if you use this library **do your own security testing** on your repository.

Enjoy!

1.1 Principles

This library provides framework for writing / using declarative ACLs (Access Control Lists) for Invenio Records. The ACLs are stored in the local database and indexed into Elasticsearch for fast record retrieval.

ACLs can be assigned to a single record or a set of records identified by configurable selectors. ACLs can be added/modified/removed during runtime.

1.1.1 ACL anatomy

An ACL rule consists of two parts:

1. Description part (what can be done on which records)
2. Actor part (who can do it)

Examples:

ACL 1

Actor: *everyone*

Description: Can *Read All records* in a repository,

ACL 2

Actor: *admin*

Description: Can *Read* Records that have a property *secret=true*,

1.1.2 Description part

Description part is a database model stored in the database and consists of:

name: the name of the ACL, not used within the system, only for administrators

priority: the priority of the ACL rule, see below for details

operation: an abstract operation which the ACL allows, for example "get", "update" "delete" for standard REST operations. Can be any string for custom operations (such as "approve", "publish", ...).

schemas: a list of schemas to which the ACL is applicable. Records having other schemas (in *$schema* metadata property) are not affected by the ACL.

Priority field: when several ACLs match a given record, only those with the highest priority are applied. This enables exceptions in ACLs. For example:

ACL 1:

- Description: "*Can Read All records in a repository*",
- Actor: "*everyone*"
- Priority: 0

ACL 2:

- Mapping: "*Can Read records that have a property 'secret=true'*",
- Actor: "*admin*"
- Priority: 1

When ACLs are applied to a secret record, both ACLs match, but only the second one is used.

1.1.3 Actor part

An Actor is an abstract database model that decides who is allowed to perform an operation. Several implementations are provided, selecting users by their id, invenio role, or selecting system users (everyone, authenticated, ...).

1.1.4 Extensibility

Both the description part and actor part are extensible and can use your own implementations.

1.1.5 Performance

Get record	Fast (no extra query to elasticsearch)
List record	Fast (no extra query to elasticsearch)
Create a new record	Reasonably Fast (1 extra query to ES)
Edit record	Reasonably Fast (1 extra query to ES)
Delete record	Fast, no extra query
Create a new ACL	Depends on number of records: Must reindex all records it applies to
ACL modification	Depends on number of records: Must reindex all records it applies to
ACL deletion	Depends on number of records: Must reindex all records it applies to

- If your use case involves lot of ACLs each assigned to a small subset of records, this library might be for you.
- If your use case involves frequently changing ACLs each assigned to a small subset of records, this library might be for you.
- If your use case involves setting up "default" ACLs in advance and not modifying them afterwards, this library might be for you.
- On the other hand if you plan to have ACLs that will change frequently and each will affect a lot of records, then this library is definitely not for you.

1.2 Installation

As usual, that is:

```
pip install invenio-explicit-acls
```

To use development version (NOT recommended, use with extra care!):

```
pip install git+https://github.com/oarepo/invenio-explicit-acls.git
```

Do not forget to initialize database tables with:

```
invenio db init create
```

1.3 Configuration

invenio-explicit-acls discriminates records via their *$schema* property that needs to be present in the metadata of guarded records.

The following configuration steps should be carried out for each enabled record type:

1. Make sure that the *$schema* property is always set and can not be changed or removed to circumvent ACLs. To guarantee that on internal API level, use your own implementation of *Record* inherited from *SchemaKeepingRecordMixin* or *SchemaEnforcingRecord* (a helper class inheriting from *Record* and *SchemaKeepingRecordMixin*). The *ALLOWED_SCHEMAS* is a list of schemas that are allowed in user data, *PREFERRED_SCHEMA* will be used when user does not specify a schema. Whenever you call (internally) a *ThesisRecord.create(...)* the *$schema* will get added automatically.

```
# myapp/constants.py
    ACL_ALLOWED_SCHEMAS = ('http://localhost/schemas/theses/thesis-v1.0.0.json',)
    ACL_PREFERRED_SCHEMA = 'http://localhost/schemas/theses/thesis-v1.0.0.json'

# myapp/api.py
class ThesisRecord(SchemaEnforcingRecord):
    ALLOWED_SCHEMAS = ACL_ALLOWED_SCHEMAS
    PREFERRED_SCHEMA = ACL_PREFERRED_SCHEMA

# myapp/config.py
RECORDS_REST_ENDPOINTS = {
    'thesis': dict(
        # ...
        record_class=ThesisRecord,
        # ...
    )
}
```

2. Make Invenio use your Record class for all API calls (just a precaution, real validation on REST calls is in the next step):

```
THESES_PID = 'pid(recid,record_class="myapp.api:ThesisRecord")'

# myapp/config.py
RECORDS_REST_ENDPOINTS = {
    'thesis': dict(
        # ...
        item_route='/theses/<{0}:pid_value>'.format(THESES_PID),
        # ...
    )
}
```

3. For metadata validation during REST, extend your marshmallow schema to inherit from *SchemaEnforcingMixin* and do not forget to set *ALLOWED_SCHEMAS* and *PREFERRED_SCHEMA*:

```
# myapp/marshmallow/json.py
class ThesisMetadataSchemaV1(SchemaEnforcingMixin,
                             StrictKeysMixin):
    """Schema for the thesis metadata."""

    ALLOWED_SCHEMAS = ACL_ALLOWED_SCHEMAS
    PREFERRED_SCHEMA = ACL_PREFERRED_SCHEMA

    title = SanitizedUnicode(required=True, validate=validate.Length(min=3))
    # ... metadata fields

# myapp/loaders/init.py
json_v1 = marshmallow_loader(ThesisMetadataSchemaV1)

# myapp/config.py
RECORDS_REST_ENDPOINTS = {
    'thesis': dict(
        # ...
        record_loaders={
            'application/json': 'myapp.loaders:json_v1',
        },
        # ...
```

(continues on next page)

(continued from previous page)

```
    )
}
```

4. Use ACLRecordsSearch as your REST search class:

```
# myapp/config.py
RECORDS_REST_ENDPOINTS = {
    'thesis': dict(
        # ...
        search_class=ACLRecordsSearch,
        # ...
    )
}
```

5. Use permissions from *invenio_explicit_acls.permissions* as your permission factory impl:

```
# myapp/config.py
RECORDS_REST_ENDPOINTS = {
    'thesis': dict(
        # ...
        read_permission_factory_imp=acl_read_permission_factory,
        update_permission_factory_imp=acl_update_permission_factory,
        delete_permission_factory_imp=acl_delete_permission_factory,
        # ...
    )
}
```

6. Do not forget to supply your own *create_permission_factory_impl* - it is not handled by this library!
7. If not using marshmallow, adapt your loader to check and fill the *$schema* property. Never trust user (or your code) and always check!
8. For each of the schemas defined in step 1, create additional indices in ES:

```
# run in bash
invenio explicit-acls prepare <schema-url>
```

schema-url is a relative (short) schema name, for example records/record-v1.0.0.json

9. Restart the server and you are ready to go.

1.4 Usage

1.4.1 Description part

The description part is called *ACL* within the library.

The following implementations are built-in:

IdACL: the ACL applies to records identified by their internal Invenio UUIDs

DefaultACL: the ACL applies to all records in a given schema(s)

ElasticsearchACL: the ACL applies to all records in the given schema(s) that match the given ES query

PropertyValueACL: simpler implementation of ElasticsearchACL. The ACL applies to all records in the given schema(s) whose named property/set of properties has a given value

1.4.2 Actors

Actor defines who has access to a set of resources identified by mapping above. The following implementations are built-in:

UserActor: a set of users (direct enumeration) that have access

RoleActor: a set of user roles that have access

SystemRoleActor: an actor that matches anonymous users, authenticated users or everyone

Actors can also take data from the indexed document. For example, if the document contains a property "creator_id", one can use *RecordUserActor(..., path='/creator_id')* to write an ACL matching the creator (whoever it is).

The following record actors are built-in:

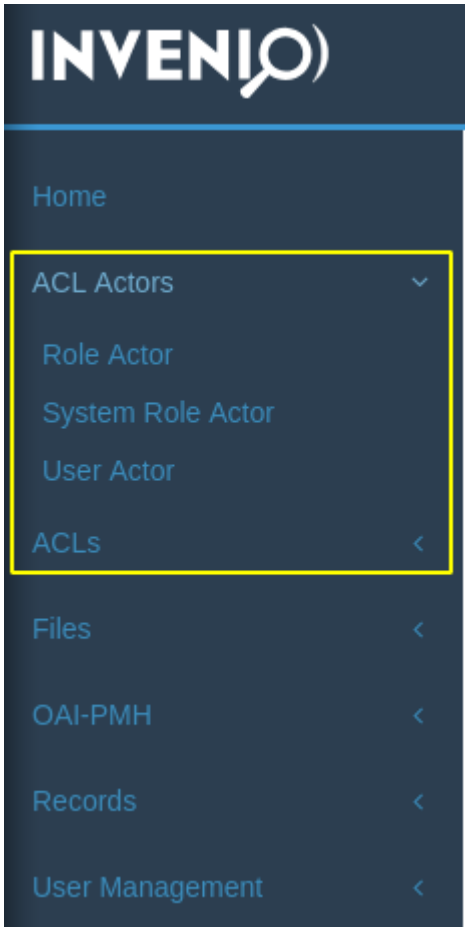
RecordUserActor: a set of users enumerated in a property in the indexed record

RecordRoleActor: a set of user roles enumerated in a property in the indexed record

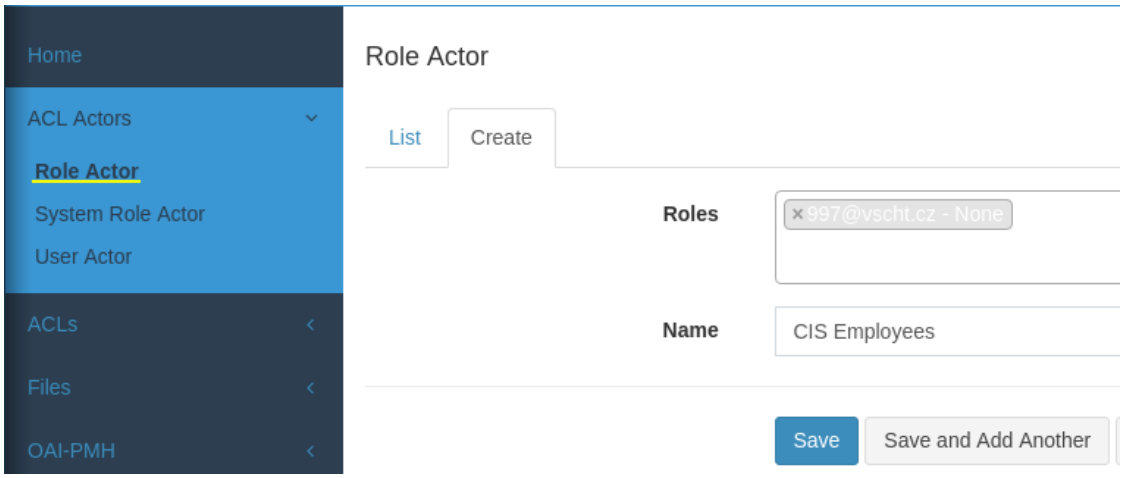
Again, these are extensible, so for example if the record metadata contains property *faculty*, one can write a custom *PRBACRoleActor(role='administrator', parameter='faculty', path='/faculty')* to assign rights to the correct faculty administrator defined in a local PRBAC (parametrized role-based access control) system - would match all users with prbac role "administrator[faculty=<value of faculty property in the document>]"

1.4.3 Admin interface

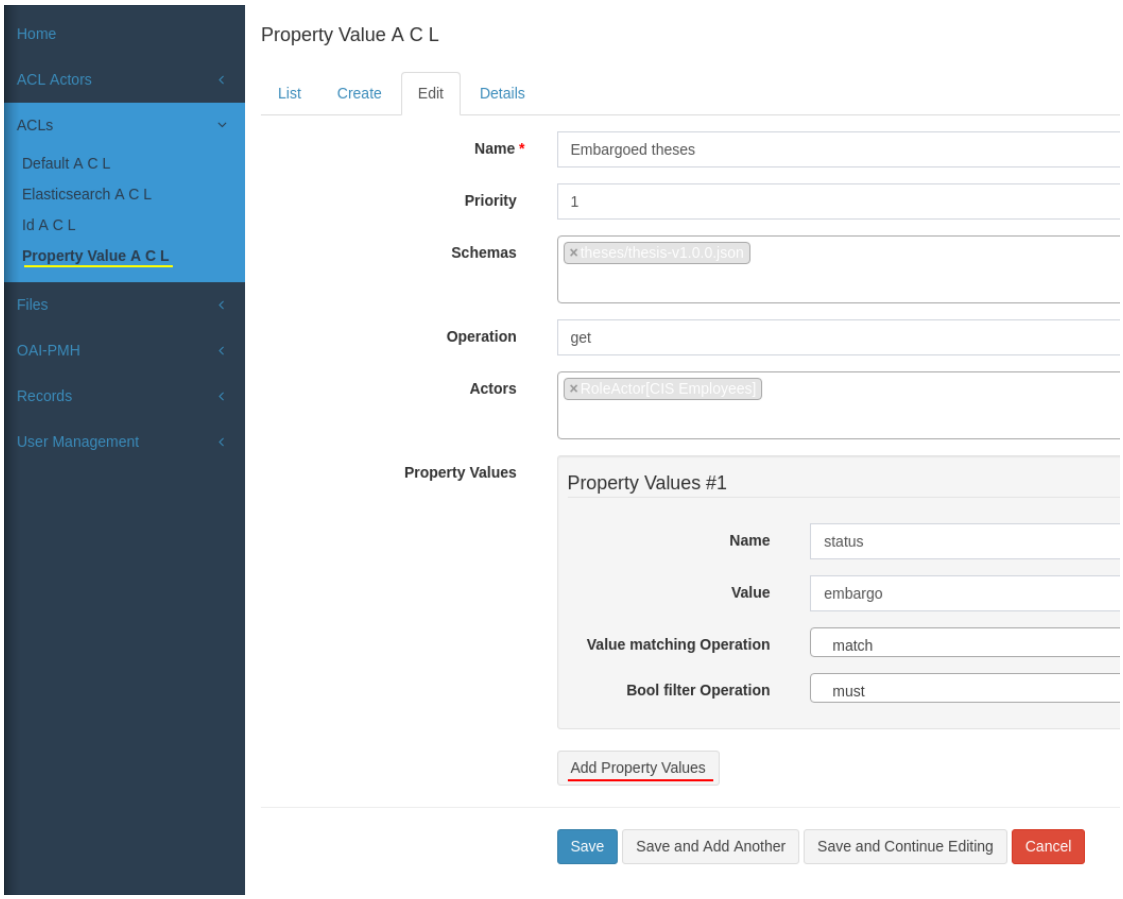
The ACLs and actors can be set in the admin interface (albeit not comfortably - we expect that the ACLs are created by your custom code/ui to restrict users to create only ACLs of certain type).



At first create an ACL Actor, in this example a Role ACL (make sure you have the role defined in User Management / Role tab).



In the second step map the actor to operation and records, for example property-value based:



1.4.4 Within Python code

Creating/Updating/Deleting ACLs

The ACL and Actors are normal sqlalchemy models, use them as usual. For example:

```
acl = DefaultACL(name='record default', schemas=[RECORD_SCHEMA],
                 operation='get',
                 originator=current_user)
user = User.query.filter(email='...').one()
actor = UserActor(name='VIP users', originator=test_users.u1,
                  users=[user])
db.session.add(acl)
db.session.add(actor)
```

To (re)apply the ACL to existing records do not forget to call:

```
from invenio_explicit_acls.proxies import current_explicit_acls

acl = ....
current_explicit_acls.reindex_acl(acl, delayed=False)
```

for cases when ACL is created / modified and:

```
from invenio_explicit_acls.proxies import current_explicit_acls
```

(continues on next page)

(continued from previous page)

```
acl = .... # (a removed acl)
current_explicit_acls.reindex_acl_removed(acl, delayed=False)
```

when ACL has been removed.

Searching with current_user

To search records within a request for the *current_user* just replace RecordsSearch class with ACLRecordsSearch. For example:

```
index, doc_type = schema_to_index(RECORD_SCHEMA)

data = ACLRecordsSearch(index=index, doc_type=doc_type).execute().hits
```

For more info see https://invenio-search.readthedocs.io.

Searching on behalf of another user

Sometimes we need to search on behalf of another user or the current_user is not set (when working outside the request context, such as in celery task). The ACLs need to get:

- the user
- set of system roles, such as *any_user*, *authenticated_user* from *invenio_access.permissions*

```
from invenio_access.permissions import authenticated_user

tested_user = ...

data = ACLRecordsSearch(
    index=index, doc_type=doc_type,
    user=tested_user,
    context = {
        system_roles=[authenticated_user]
    }
).execute().hits
```

Always provide *system_roles*. If not provided, *SystemRoleActor* will take them from *g.identity* which is probably not what you want in this context !

1.5 Implementation

This section summarizes how the ACLs are implemented.

1.5.1 prepare index script

When the following script is run:

```
# run in bash
invenio explicit-acls prepare <schema-url>
```

the following steps will take place:

Adding Property to Existing Index Mapping

The mapping for elasticsearch index corresponding to the schema-url (retrieved from *invenio_search.utils.schema_to_index* call) is enhanced with an extra property *_invenio_explicit_acls* (called ACL field in this text). The definition of the mapping for the ACL field looks like this:

```
{
    _invenio_explicit_acls: {
        type: "nested",
        properties: {
            id: {
                type: "keyword"
            },
            operation: {
                type: "keyword"
            },
            role: {
                type: "integer"
            },
            system_role: {
                type: "keyword"
            },
            timestamp: {
                type: "date"
            },
            user: {
                type: "integer"
            }
        }
    }
}
```

During indexing the library at first gets a set of ACLs that match the record being indexed. Each matching ACL is serialized into the ACL field as follows:

id	The database id of the ACL rule
operation	The operation (get, update, delete or custom
timestamp	The current server timestamp

The rest of the sub-properties (role, system_role, user) are coming from Actor instances and define which role, system_role or user have access to the resource's operation. If a custom Actor is implemented, other sub-properties will be present in the ACL field as well.

An example of ACL field as found in a document in the index:

```
{
    _invenio_explicit_acls: [
        {
            "id": 1,
            "operation": "update",
            "timestamp": "2019-01-01T00:00:00Z",
            "user": [1, 2]
        },
        {
            "id": 2,
            "operation": "get",
            "timestamp": "2019-01-01T00:00:00Z",
```

(continues on next page)

(continued from previous page)

```
            "system_role": "any_user"
        }

}
```

Only users with ids 1 or 2 (= id of the User model) will be able to update this document and everyone will be able to read it.

During get/search (performed by *invenio_explicit_acls.acl_records_search.ACLRecordsSearch*) the search query is wrapped with the following query (simplified, the real bool query contains operation checks and role as well) so that the search returns only the resources to which the user has access (under the operation being checked - get, update, delete or any custom operation):

```
{
    "must": [
        original_query,
        "bool": {
            "should": [
                {
                     "nested" : {
                        "path" : "_invenio_explicit_acls",
                        "query" : {
                            "term": { "_invenio_explicit_acls.user": current_user_id }
                        }
                     }
                },
                {
                     "nested" : {
                        "path" : "_invenio_explicit_acls",
                        "query" : {
                            "term": { "_invenio_explicit_acls.system_role": "any_user
↪" }
                        }
                     }
                },
            ],
            "min_should_match": 1
        }
    ]
}
```

Creating a new index for percolate queries

The second half of the story is that given a record we need to get the set of ACLs that describe the record. It is easy for the simple ones:

1. DefaultACL - maps to every record with a given *$schema* property
2. IdACL - maps to just one record whose Invenio uuid is stored in the ACL

The more usable ACLs need more handling:

3. PropertyValueACL defines a set of properties, their values and matching operation in elasticsearch (term, match) and a combining operation (must, should, must not). If the condition holds against a given record, the ACL matches
4. ElasticsearchACL allows to specify a generic ES query that is run against the record.

To efficiently match record against these types of ACL we define a new index, called for example *invenio_explicit_acls-acl-v1.0.0-theses-thesis-v1.0.0* with the following mapping:

```
{
    $schema: {
        type: "keyword",
        index: false
    },
    __acl_record_selector: {
        type: "percolator"
    },
    __acl_record_type: {
        type: "keyword"
    },
    // the rest of the mapping from theses-thesis-v1.0.0
}
```

Whenever an ES-backed ACL is defined to operate on thesis record type, its query is stored to the *__acl_record_selector* property.

Later on, when we search which ACLs describe a record we perform a percolate query against this index that efficiently evaluates all the *__acl_record_selector* and returns those ACLs that match the record.

1.5.2 The role of $schema

The percolate index above needs to be defined for every data model as it has to contain the properties from the data model. To know which percolate index to use we need to know the type of the record being indexed and the only reliable record property we can use is its schema stored in the *$schema* property.

To make sure that the ACLs are not circumvented we need to make sure that once the schema is set it can not be removed or modified. This logic is contained in *invenio_explicit_acls.record.SchemaKeepingRecordMixin* and *invenio_explicit_acls.record.SchemaEnforcingRecord* that make sure that:

1. If *$schema* is not set, it will be set with a default value, defined by *self.PREFERRED_SCHEMA* class property
2. If *$schema* is set, it will not be removed (with *clear()*, *update()*, *record['$schema']='...'* or *del record['$schema']*)
3. If *$schema* is changed, it might get changed only to a set of predefined values, defined by *self.ALLOWED_SCHEMAS* property

Any other modifications to *$schema* will raise an *AttributeError*.

To be double protected on the REST level, extend your metadata marshmallow with *invenio_explicit_acls.marshmallow.SchemaEnforcingMixin* that performs the same checks as above before the record is converted to its internal form.

1.5.3 ACL update

Whenever ACL is updated, we need to modify the ACL field of records in the target index. Unfortunately this can not be done effectively with ES update-by-query call as Invenio depends on external versioning in Elasticsearch that would get broken by update-by-query. That's why the following code gets called (defined in *invenio_explicit_acls.tasks*):

```
@shared_task(ignore_result=True)
def acl_changed_reindex(acl_id):
    """
    ACL has been changed so reindex all the documents in the given index.
```

(continues on next page)

(continued from previous page)

```
    :param acl_id:   id of ACL instance
    """
    logger.info('Reindexing started for ACL=%s', acl_id)
```

At first we remember the current time. It will be used later to handle records that are no longer covered by the ACL. We also flush the index to get the following queries up to date:

```
timestamp = datetime.datetime.now().astimezone().isoformat()

acl = ACL.query.filter_by(id=acl_id).one_or_none()

if not acl:
    # deleted in the meanwhile, so just return
    return          # pragma no cover

# make sure all indices are flushed so that no resource is obsolete in index
for schema in acl.schemas:
    current_search_client.indices.flush(index=schema_to_index(schema)[0])

indexer = RecordIndexer()
updated_count = 0
removed_count = 0
```

For each of the matching records we reindex them. This will propagate the changes made to the ACL and will also update the *timestamp* property on ACL field.

```
for id in acl.get_matching_resources():
    try:
        rec = Record.get_record(id)
    except:     # pragma no cover
        # record removed in the meanwhile by another thread/process,
        # indexer should have been called to remove it from ES
        # won't test this so pragma no cover
        continue
    try:
        indexer.index(rec)
        updated_count += 1
    except Exception as e:  # pragma no cover
        logger.exception('Error indexing ACL for resource %s: %s', id, e)
```

Finally we select all the records that were covered by the ACL and whose *timestamp* property is older than the time of the beginning of the indexing. This will reindex records that will no longer be covered by the ACL:

```
# reindex the resources those were indexed by this acl but no longer should be
for id in acl.used_in_records(older_than_timestamp=timestamp):
    try:
        rec = Record.get_record(id)
    except NoResultFound:                           # pragma no cover
        continue
    except:                                         # pragma no cover
        logger.exception('Unexpected exception in record reindexing')
        continue

    try:
        removed_count += 1
```

(continues on next page)

(continued from previous page)

```
        indexer.index(rec)
    except:     # pragma no cover
        logger.exception('Error indexing ACL for obsolete resource %s', id)
```

1.6 Extending ACLs

Both the description part (`invenio_explicit_acls.models.ACL`) and actor part (`invenio_explicit_acls.models.Actor`) are extensible and custom implementation can be provided via inheriting from these classes.

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 Models

2.1.2 Signals

2.1.3 Tasks

2.1.4 Permissions

2.1.5 Serializers