OpenTAXII

Installation, usage, and development

The OpenTAXII contributors

Contents

1 Introduction . . . 7

I Installation

2 Install OpenTAXII . . . 11

3 Installation . . . 13
3.1 Install with uv . . . 13
3.2 Install Python . . . 13
3.3 Install OpenTAXII . . . 13
3.4 Versioning . . . 14
3.5 Next steps . . . 14

4 Configuration . . . 15
4.1 Default configuration . . . 15
4.2 Properties . . . 16
4.3 Custom configuration . . . 17
4.4 Environment variables configuration . . . 17

5 Run OpenTAXII on Docker . . . 19
5.1 Configuration . . . 19
5.2 Volumes . . . 19
5.3 Extending . . . 20
5.4 TAXII 2 instance with Compose . . . 20
5.5 Full Example with Compose . . . 20
5.6 Next steps . . . 20

6 Running OpenTAXII . . . 21
6.1 Using SSL/TLS . . . 21

II Usage

7 Manage OpenTAXII . . . 25

8 Accounts . . . 27
8.1 Manual commands . . . 27
8.2 Sync data . . . 27

9 TAXII 1: Services and Collections . . . 29
9.1 Step 1 . . . 29
9.2 Step 2 . . . 29

10 TAXII 2: API Root and collections . . . 31
10.1 CLI commands . . . 31

11 Authentication and authorization . . . 33
11.1 Services Authentication (Client-side) . . . 33
11.2 Authentication Implementation . . . 33

12 Sending requests to services . . . 35
12.1 TAXII 1 . . . 35
12.2 TAXII 2 . . . 37

13 Health check . . . 39
13.1 Next steps . . . 39

III Contributing

14 Contribute and develop . . . 43

15 Contributing and developing . . . 45
15.1 Reporting issues . . . 45
15.2 Obtaining the source code . . . 45
15.3 Layout . . . 45
15.4 Database schema . . . 46
15.5 Set up a development environment . . . 48
15.6 Running the tests . . . 48
15.7 Generating the documentation . . . 48

16 Running Development OpenTAXII . . . 49

IV Reference

17 References . . . 53

18 API reference . . . 55
18.1 Configuration and server . . . 55
18.2 TAXII 1 entities and services . . . 64
18.3 TAXII 2 entities and validation . . . 78
18.4 Persistence, authentication, and signals . . . 87

19 Public code-level APIs . . . 107
19.1 Overview . . . 107

19.2 Custom API implementations . . . 107
19.3 Custom signal listeners . . . 107
19.4 Persistence API . . . 107
19.5 Authentication API . . . 107
19.6 Signals . . . 107

20 Changelog . . . 109
20.1 0.10.0 (unreleased) . . . 109
20.2 0.9.3 (2022-10-11) . . . 109
20.3 0.9.2 (2022-08-26) . . . 109
20.4 0.9.1 (2022-07-11) . . . 109
20.5 0.9.0 (2022-06-13) . . . 109
20.6 0.8.0 (2022-06-05) . . . 109
20.7 0.7.0 (2022-05-27) . . . 110
20.8 0.6.0 (2022-05-25) . . . 110
20.9 0.5.0 (2022-05-24) . . . 110
20.10 0.4.0 (2022-05-20) . . . 110
20.11 0.3.0 (2022-04-13) . . . 110
20.12 0.3.0a4 (2022-04-13) . . . 110
20.13 0.3.0a3 (2022-01-21) . . . 110
20.14 0.3.0a2 (2021-12-27) . . . 110
20.15 0.3.0a1 . . . 110
20.16 0.3.0a0 . . . 110
20.17 0.2.4 (2022-04-13) . . . 110
20.18 0.2.3 (2021-12-22) . . . 110
20.19 0.2.2 (2021-11-05) . . . 110
20.20 0.2.1 (2021-11-03) . . . 111
20.21 0.2.0 (2020-06-30) . . . 111
20.22 0.1.12 (2019-03-06) . . . 111
20.23 0.1.11 (2019-02-13) . . . 111
20.24 0.1.10 (2018-06-03) . . . 111
20.25 0.1.9 (2017-06-19) . . . 111
20.26 0.1.8 (2017-02-21) . . . 111
20.27 0.1.7 (2016-10-18) . . . 112
20.28 0.1.6 (2016-06-01) . . . 112
20.29 0.1.5 (2016-03-15) . . . 112
20.30 0.1.4 (2016-02-25) . . . 112
20.31 0.1.3 (2016-02-25) . . . 112
20.32 0.1.2 (2015-07-24) . . . 112
20.33 0.1.1 (2015-04-08) . . . 112
20.34 0.1.0 (2015-03-31) . . . 112

21 License . . . 113

1. Introduction

OpenTAXII is a Python implementation of TAXII for sharing cyber threat intelligence.
EclecticIQ originally developed OpenTAXII. The MISP Project fully maintains this fork. EclecticIQ · MISP Project

TAXII (Trusted Automated eXchange of Indicator Information) defines services and message exchanges for sharing threat intelligence between organizations. OpenTAXII implements TAXII 1.0 and 1.1 services and provides TAXII 2.1 endpoints for exchanging STIX content.

The Flask transport layer, TAXII handlers, persistence API, and authentication API are separate components. The server supports custom authentication and persistence implementations, signal listeners, and configurable logging.

See the changelog for release history and PyPI for published packages.

I Installation

2 Install OpenTAXII 11

3 Installation 13
3.1 Install with uv 13
3.2 Install Python 13
3.3 Install OpenTAXII 13
3.4 Versioning 14
3.5 Next steps 14

4 Configuration 15
4.1 Default configuration 15
4.2 Properties 16
4.3 Custom configuration 17
4.4 Environment variables configuration 17

5 Run OpenTAXII on Docker 19
5.1 Configuration 19
5.2 Volumes 19
5.3 Extending 20
5.4 TAXII 2 instance with Compose 20
5.5 Full Example with Compose 20
5.6 Next steps 20

6 Running OpenTAXII 21
6.1 Using SSL/TLS 21

2. Install OpenTAXII

- Installation
- Configuration
- Run OpenTAXII on Docker
- Running OpenTAXII

3. Installation

3.1 Install with uv

Follow the uv installation guide to install uv. From the directory where you want to keep the OpenTAXII virtual environment, run the following commands to install the PyPI release and start the development server.

```
uv venv --python 3.11
uv pip install opentaxii
uv run --no-project opentaxii-run-dev
```

uv stores the environment in `.venv` and downloads Python 3.11 if it needs to. The development server uses SQLite and listens on port 9000. In another terminal, check that the server is responding.

```
curl --fail http://localhost:9000/management/health
```

To install the version in a source checkout, run these commands from its root.

```
uv venv --python 3.11
uv pip install .
uv run --no-project opentaxii-run-dev
```

For an editable installation and the test suite, see Contributing and developing.

3.2 Install Python

OpenTAXII works with Python versions >=3.10. You can download Python here or install it with your operating system's package manager.

You can verify that Python is installed by typing `python` or `python3` in your shell. You should see something like:

```
$ python3
Python 3.10.1 (main, Dec 11 2021, 17:22:55) [GCC 11.1.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

3.3 Install OpenTAXII

To sandbox the project and protect system-wide python it is recommended to install OpenTAXII into a virtual environment (*virtualenv*).

Create a virtual environment named venv:

```
$ virtualenv venv
```

Where `venv` is a directory to place the new environment

Activate this environment:

```
$ . venv/bin/activate
(venv) $
```

Now install the latest OpenTAXII release from the Python Package Index (PyPI) using `pip`:

```
(venv) $ pip install opentaxii
```

Without the virtual environment it's just:

```
$ pip install opentaxii
```

> **Note**
>
> Since OpenTAXII has libtaxii as a dependency, the system libraries libtaxii will be installed. Check libtaxii documentation for the details.

To install OpenTAXII from source with pip, activate the virtual environment and run this command from the root of the checkout or unpacked source archive.

```
python -m pip install .
```

3.4 Versioning

OpenTAXII uses `major.minor.revision` version numbers. The revision number identifies bug-fix releases.

3.5 Next steps

Continue to Configuration page to learn how to configure OpenTAXII.

4. Configuration

OpenTAXII can be configured using YAML configuration files.

4.1 Default configuration

OpenTAXII has default configuration file built in.
By default:

- Provided Persistence API and Authentication API implementations are for SQL DB backends and use sqlite3 backend.
- The sqlite3 database files will be automatically created in `/tmp/data.db` and `/tmp/auth.db`.
- Services, collections and accounts are not configured automatically. This requires a manual operation (see below).
- No signal hooks are attached.
- There is only taxii1 support, taxii2 is disabled

Default configuration file looks like this:

```
---
domain: "localhost:9000"

support_basic_auth: yes
return_server_error_details: no

auth_api:
  class: opentaxii.auth.sqldb.SQLDatabaseAPI
  parameters:
    db_connection: sqlite:////tmp/auth.db
    create_tables: yes
    secret: SECRET-STRING-NEEDS-TO-BE-CHANGED
    token_ttl_secs: 3600

taxii1:
  save_raw_inbox_messages: yes
  xml_parser_supports_huge_tree: yes
  count_blocks_in_poll_responses: no
  unauthorized_status: UNAUTHORIZED
  hooks:
  persistence_api:
    class: opentaxii.persistence.sqldb.SQLDatabaseAPI
    parameters:
      db_connection: sqlite:////tmp/data.db
      create_tables: yes

taxii2:

logging:
  opentaxii: info
  root: info
```

> **Note**
>
> OpenTAXII uses a SQLite Database by default wich is intended only when running OpenTAXII in a development environment. Please use different SQL DB backend for running in a production environment.

An example of custom configuration that allows OpenTAXII to connect to production-ready PostgreSQL database:

```
---
auth_api:
  class: opentaxii.auth.sqldb.SQLDatabaseAPI
  parameters:
    db_connection: postgresql://username:P@ssword@db.example.com:5432/databasename
    create_tables: yes
    secret: SECRET-STRING-NEEDS-TO-BE-CHANGED

taxii1:
  persistence_api:
    class: opentaxii.persistence.sqldb.SQLDatabaseAPI
    parameters:
      db_connection: postgresql://username:P@ssword@db.example.com:5432/databasename
      create_tables: yes
```

4.2 Properties

- `domain` — domain that will be used in service URLs in TAXII responses.
- `support_basic_auth` — enable/disable Basic Authentication support. If disabled, only JWT authentication is allowed.
- `return_server_error_details` — allow OpenTAXII to return error details in error-status TAXII response.
- `auth_api` — configuration properties for Authentication API implementation.
  - `class` — the full import name of the class to use
  - `parameters` — the parameters used to instantiate the class
    - `db_connection` — the database connetion string
    - `create_tables` — boolean, if true, create tables on startup
    - `secret` — the secret with which the generated tokens are encoded
    - `token_ttl_secs` — time that generated tokens are valid
- `taxii1` — taxii1-specific settings
  - `save_raw_inbox_message` — enable/disable storing of raw TAXII Inbox messages via Persistence API's `create_inbox_message` method. This is useful for bookkeeping but significantly increases storage requirements.
  - `xml_parser_supports_huge_tree` — enable/disable security restrictions in lxml library to allow support for very deep trees and very long text content. If this is disabled, OpenTAXII will not be able to parse TAXII messages with content blocks larger than roughly 10MB.
  - `count_blocks_in_poll_responses` — enable/disable total count in TAXII Poll responses. It is disabled by default since `count` operation might be very slow in some SQL DBs.
  - `unauthorized_status` — TAXII status type for authorization error. "UNAUTHORIZED" by default. see libtaxii.constants.ST_TYPES_11 for the list of available values.
  - `hooks` - custom python module with signal subscriptions to import. See documentation on custom signals and an example.
  - `persistence_api` — configuration properties for Persistence API implementation.
    - `class` — the full import name of the class to use
    - `parameters` — the parameters used to instantiate the class
      - `db_connection` — the database connetion string
      - `create_tables` — boolean, if true, create tables on startup
- `taxii2` — taxii2-specific settings
  - `persistence_api` — configuration properties for Persistence API implementation.
    - `class` — the full import name of the class to use
    - `parameters` — the parameters used to instantiate the class
      - `db_connection` — the database connetion string
      - `create_tables` — boolean, if true, create tables on startup
  - `max_content_length` — the maximum size of the request body in bytes that the server can support. Required field

  - `allow_custom_properties` — boolean, if true, allow custom stix2 properties when posting objects (default: true)
  - `public_discovery` - boolean, if true, do not require authentication for discovery of api roots (default: false)
  - `title` - title of the server, returned as part of the discovery of api roots. Required field
  - `contact` - contact for the server, returned as part of the discovery of api roots
  - `description` - description of the server, returned as part of the discovery of api roots
- `logging` — logging configuration.

4.3 Custom configuration

To pass custom configuration to OpenTAXII server, specify an absolute path to your configuration file in environment variable `OPENTAXII_CONFIG`.:

```
$ export OPENTAXII_CONFIG=/path/to/configuration/file.yml
```

This configuration file may fully or partially override default settings.

Example custom configuration:

```
---
domain: taxii.mydomain.com
support_basic_auth: no

auth_api:
  class: opentaxii.auth.sqldb.SQLDatabaseAPI
  parameters:
    db_connection: postgresql://scott:tiger@localhost:5432/mydatabase
    create_tables: yes
    secret: aueHenjitweridUcviapEbsJocdiDrelHonsyorl

taxii1:
  xml_parser_supports_huge_tree: no
  hooks: mypackage.opentaxii.custom_hooks
  persistence_api:
    class: mypackage.opentaxii.CustomPersistenceAPI
    parameters:
      rest_api: http://rest.mydomain.com/api
```

The built-in implementation of the Persistence and Authentication APIs support SQLite, PostgreSQL, MySQL, and other SQL databases. Check SQLAlchemy website to get the full list.

OpenTAXII CLI tools are implemented to call corresponding API methods and support any API implementation.

4.4 Environment variables configuration

You can (re)define any configuration option with environment variables. Start variable name with `OPENTAXII_`. Use `__` to separate parts of the config path. Use uppercase. Specify values in YAML syntax.

```
export OPENTAXII_DOMAIN='taxii.mydomain.com'
export OPENTAXII_SUPPORT_BASIC_AUTH='no'
export OPENTAXII__PERSISTENCE_API__CLASS='mypackage.opentaxii.CustomPersistenceAPI'
export
OPENTAXII__AUTH_API__PARAMETERS__SECRET='aueHenjitweridUcviapEbsJocdiDrelHonsyorl'
```

Environment variables applied after all other configs and can be used to redefine any option.

4.4.1 Next steps

Continue to the Running OpenTAXII page to see how to run OpenTAXII.

5. Run OpenTAXII on Docker

OpenTAXII can also be run using docker. This guide assumes that you have access to a local or remote docker server, and won't go into the setup of docker.

From the root of this repository, build and run a development image.

```
docker build -t opentaxii:local .
docker run --pull=never -d -p 9000:9000 opentaxii:local
```

To have the instance preloaded with example data, see volumes

> **i** Note
>
> OpenTAXII is now accessible through port 9000, with data stored locally in a SQLite database.

5.1 Configuration

Configuration is done through environment variables.

Common configuration parameters are:

`OPENTAXII_DOMAIN` : (optional)
This specifies under which domain the OpenTAXII server is available, default: `localhost:9000`

Setting up authentication is done with the following two variables:

`OPENTAXII_SECRET` : (optional)
This is the secret with which the generated token is encoded.

If you want to use a PostgreSQL database, instead of the included SQLite database, you can use the following environment variables for configuration:

`DATABASE_HOST` : (required)
This is the database host to connect to

`DATABASE_PORT` : (optional)
Default is `5432`

`DATABASE_NAME` : (optional)
The database to use, by default uses `postgres`

If you would like to use a different Database for authentication, you can also set the following variables (any variable not set, will use it's regular DB counterpart):

`AUTH_DATABASE_HOST` : (required)
This is the database host to connect to

`AUTH_DATABASE_PORT` : (optional)
Default is `5432`

`AUTH_DATABASE_USER` : (optional)
If not set, the default `postgres` is used.

`AUTH_DATABASE_PASS` : (optional)
If not set, the database can be accessed by all containers on the same host!

`AUTH_DATABASE_NAME` : (optional)
The database to use, by default uses `postgres`

5.2 Volumes

This docker container exposes two volumes, which can be attached to a running instance:

`/data`
This volume will contain the SQLite databases used by the default instance.

`/input/data-configuration.yml`
If you want to pre-load the running instance with services/collections/accounts, An example is provided with data-configuration.yml.

`/input/opentaxii.yml`
To overwrite the default configuration used by the docker instance.

After building `opentaxii:local`, run this command from the repository root to load the example data.

```
docker run --pull=never -d -p 9000:9000 -v "$PWD/examples:/input:ro" opentaxii:local
```

> **Note**
>
> Make sure your naming is correct. It will only execute actions when files `data-configuration.yml` or `opentaxii.yml` are present.

5.3 Extending

To install a custom package and supply an OpenTAXII configuration file, use the locally built `opentaxii:local` image as the base for a Dockerfile.

```
FROM opentaxii:local

RUN pip install custom-package

COPY opentaxii.yml /input/opentaxii.yml
```

Build the custom image from the directory containing this Dockerfile and your `opentaxii.yml`.

```
docker build -t opentaxii:custom .
```

5.4 TAXII 2 instance with Compose

The TAXII 2 Compose example in `examples/docker-compose-taxii2.yml` builds OpenTAXII from this checkout. Run these commands from the repository root.

```
# Start
docker compose -f examples/docker-compose-taxii2.yml up --build

# To add dummy data, run this while the compose project is running
docker compose -f examples/docker-compose-taxii2.yml exec -T opentaxii bash < examples/
taxii2/data-setup.sh
```

5.5 Full Example with Compose

The full Compose example in `examples/docker-compose.yml` builds OpenTAXII from this checkout and starts two OpenTAXII instances backed by PostgreSQL. It preloads the services, collections, and accounts from `examples/data-configuration.yml`.

Run this command from the repository root.

```
docker compose -f examples/docker-compose.yml up --build
```

5.6 Next steps

Continue to Configuration page to learn how to configure OpenTAXII.

6. Running OpenTAXII

To run OpenTAXII in production it is recommended to use standalone WSGI container that works with Flask. In this example we use Gunicorn WSGI HTTP Server. For complete logging configuration we recommend Gunicorn 19.8 and above.

> **Note**
>
> Run `pip install gunicorn` to install gunicorn. Yes, it's that simple.

To run OpenTAXII with Gunicorn execute:

```
(venv) $ gunicorn opentaxii.http:app \
  --bind localhost:9000 --config python:opentaxii.http
```

Common practice is to wrap gunicorn execution into supervisord, to be able to monitor, start, and stop it easily.

Example supervisord configuration file:

```
[program:opentaxii]

command =
    /opt/eclecticiq/opentaxii-venv/bin/gunicorn opentaxii.http:app
        --workers 2
        --log-level info
        --log-file -
        --timeout 300
        --bind localhost:9000
        --config python:opentaxii.http
environment =
    OPENTAXII_CONFIG="/opt/eclecticiq/custom-opentaxii-configuration.yml"

stdout_logfile = /var/log/opentaxii.log
redirect_stderr = true
autostart = true
autorestart = true
```

6.1 Using SSL/TLS

If you want to run OpenTAXII with SSL, you need to use a web server like Nginx, that provides SSL/TLS layer. You can find details on how to run Nginx with SSL here.

Make sure you configure your TAXII services in OpenTAXII with proper protocol bindings:

- use `urn:taxii.mitre.org:protocol:https:1.0` if you're planning on serving data via HTTPS.
- use `urn:taxii.mitre.org:protocol:http:1.0` if the server is going to support unsecure HTTP as well.

You can use multiple protocol bindings per service. That would tell OpenTAXII you want to advertise services over both HTTPs and HTTP. TAXII services create external URLs according to their protocol bindings, using `http://` or `https://` schemas.

Continue to Manage OpenTAXII pages to learn how to manage the OpenTAXII instance.

II Usage

7 Manage OpenTAXII 25

8 Accounts 27
8.1 Manual commands 27
8.2 Sync data 27

9 TAXII 1: Services and Collections . . 29
9.1 Step 1 29
9.2 Step 2 29

10 TAXII 2: API Root and collections . . 31
10.1 CLI commands 31

11 Authentication and authorization . . 33
11.1 Services Authentication (Client-side) 33
11.2 Authentication Implementation 33

12 Sending requests to services 35
12.1 TAXII 1 35
12.2 TAXII 2 37

13 Health check 39
13.1 Next steps 39

7. Manage OpenTAXII

- Accounts
- TAXII 1: Services and Collections
- TAXII 2: API Root and collections
- Authentication and authorization
- Sending requests to services
- Health check

8. Accounts

Accounts can be created with CLI commands or with custom code talking to a specific Persistent API implementation/backend.

8.1 Manual commands

```
opentaxii-create-account --username <username> --password <pass> --admin

# Update password
opentaxii-update-account --username <username> --field password --value <pass>

# Set admin
opentaxii-update-account --username <username> --field admin --value yes
# Reset admin
opentaxii-update-account --username <username> --field admin --value no
```

8.2 Sync data

```
accounts:
- username: admin
    password: admin
    is_admin: yes
```

9. TAXII 1: Services and Collections

Services and Collections can be created with CLI commands `opentaxii-sync-data` or with custom code talking to a specific Persistence API implementation/backend.

> **Note**
>
> The services and collections created with CLI command `opentaxii-sync-data` are only available with TAXII 1.

9.1 Step 1

Create a YAML file with your collections, services, and accounts. The example below comes from the MISP-maintained fork. Example configuration · EclecticIQ upstream · MISP Project

Services:

- 2 Inbox Services (with ids `inbox_a` and `inbox_b`),
- 1 Discovery Service (with id `discovery_a`),
- 1 Collection Management Service (with id `collection_management_a`),
- 1 Poll Service (with id `poll_a`).

> **Note**
>
> Services have relative path in the address field, which means OpenTAXII will prepend it with domain configured in server configuration file (`localhost:9000` in Default configuration).

Collections:

- `collection-a` that has type `DATA_SET`, accepts all content types and is attached to services `inbox_a`, `collection_management_a`, and `poll_a`.
- `collection-b` that accepts only content types specified in field `content_bindings` and is attached to services `inbox_a`, `inbox_b`, `collection_management_a` and `poll_a`.
- `collection-c` that accepts not only STIX v1.1.1 content type but also custom content type `urn:custom.bindings.com:json:0.0.1`. It is attached to services `inbox_a`, `collection_management_a` and `poll_a`.
- `col-not-available` that is marked as not available, even though it is attached to `inbox_b` and `collection_management_a`.

Accounts:

- account with username `test` and password `test`, with ability to modify collection `collection-a`, read `collection-b` and `coll-stix-and-custom`, and unknown permission `some` for non-existing collection `collection-xyz`. Incorrect settings will be ignored during sync.
- account with username `admin` and password `admin` that has admin permissions because `is_admin` is set to `yes`.

> **Note**
>
> Without an account you can't access services with `authentication_required` enabled.

9.2 Step 2

Use `opentaxii-sync-data` command to synchronize data configuration in provided file and in DB.
Usage help:

```
(venv) $ opentaxii-sync-data --help
usage: opentaxii-sync-data [-h] [-f] config

Create services/collections/accounts

positional arguments:
  config              YAML file with data configuration

optional arguments:
  -h, --help          show this help message and exit
  -f, --force-delete  force deletion of collections and their content blocks
                      if collection is not defined in configuration file
                      (default: False)
```

To sync data run:

```
(venv) $ opentaxii-sync-data examples/data-configuration.yml
```

> **Note**
>
> To drop the databases, just delete sqlite3 files `/tmp/data.db`, `/tmp/auth.db` and restart OpenTAXII server.

Now OpenTAXII has services, collections and accounts configured and can function as a TAXII server. Check Use OpenTAXII to see how to use it.

10. TAXII 2: API Root and collections

API Root and collections can be created with CLI commands or with custom code talking to a specific Persistent API implementation/backend.

10.1 CLI commands

```
opentaxii-add-api-root --title default --description "The default API root" --default

opentaxii-add-collection --rootid <root-uuid> --title "A collection" --public
```

11. Authentication and authorization

11.1 Services Authentication (Client-side)

Authentication can be configured per service instance with `authentication_required`. This is a boolean service class field which defines if the client needs to be authenticated before accessing the service.

> **Note**
>
> OpenTAXII supports session-less token-based authentication as primary method of authentication with fallback to HTTP Basic Authentication.

If authentication is required, OpenTAXII expects clients to obtain a token first and send it as `Authorization` HTTP header value (formatted as `Bearer TOKENVALUE`) with every TAXII request.

To obtain a token, client sends a `POST` request with username and password to authentication service running on `/management/auth` endpoint.

Request data can be a form-encoded string or a JSON dictionary with `username` and `password` fields:

```
(venv) $ curl -d 'username=test&password=test' http://localhost:9000/management/auth
```

or:

```
(venv) $ curl -H "Content-Type: application/json" -d
'{"username":"test","password":"test"}' http://localhost:9000/management/auth
```

A server will reply with a JSON dictionary that contains a token. For example:

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhY2NvdW50X2lkIjoxLCJleHAiOjE0Mjc2MzgyMjN9.oKd43j4KR1O
}
```

To query a service with authentication enabled, pass the token in a value of `Authorization` HTTP header. Example request using Cabby library CLI command:

```
# using raw HTTP header
(venv) $ taxii-poll \
            --path http://localhost:9000/services/poll-a \
            --collection collection-a \
        --header Authorization:'Bearer eyJleHAiOjE0MjY3OTMwOTYsImFsZyI6IkhTMjU2IiwiaWF0IjoxNDI2Nzg1ODk2fQ

# using JWT support in Cabby
(venv) $ taxii-poll \
            --host localhost \
            --port 9000 \
            --path /services/poll-a \
            --collection collection-a \
            --username test \
            --password test \
            --jwt-auth /management/auth
```

An alternative to that is to use Basic Authentication: client sends username/password formatted as Basic Authentication header with every TAXII request, OpenTAXII decodes it and passes username/password pair to Auth API for authentication.

11.2 Authentication Implementation

The build-in Authentication API implementation uses JWT (JSON Web Token) to create tokens.

Token TTL is set to 60 minutes by default but can be configured with `token_ttl_secs` in `Auth API` settings in custom configuration.

Authentication API needs to be able to identify a client based on the token (method opentaxii.auth.api.OpenTAXIIAuthAPI.get_account of the API). The built-in implementation does that by encoding account ID inside the token.

11.2.1 Next steps

Continue to Public code-level APIs page for the details about OpenTAXII APIs.

12. Sending requests to services

12.1 TAXII 1

The easiest way to send requests to TAXII services is to use Cabby library CLI tools:

```
(venv) $ pip install cabby
```

If you are running OpenTAXII in default configuration and you created services using provided examples, you should be able to communicate with the services.

Assuming OpenTAXII is running on `http://localhost:9000`, to get the list of advertised services, run:

```
(venv) $ taxii-discovery --path http://localhost:9000/services/discovery-a
```

You should see the following output:

```
2018-06-03 21:26:32,188 INFO: Sending Discovery_Request to http://localhost:9000/
services/discovery-a
2018-06-03 21:26:32,207 INFO: 7 services discovered
=== Service Instance ===
  Service Type: INBOX
  Service Version: urn:taxii.mitre.org:services:1.1
  Protocol Binding: urn:taxii.mitre.org:protocol:http:1.0
  Service Address: http://localhost:9000/services/inbox-a
  Message Binding: urn:taxii.mitre.org:message:xml:1.0
  Message Binding: urn:taxii.mitre.org:message:xml:1.1
  Inbox Service AC: []
  Available: True
  Message: Custom Inbox Service Description A

=== Service Instance ===
  Service Type: INBOX
  Service Version: urn:taxii.mitre.org:services:1.1
  Protocol Binding: urn:taxii.mitre.org:protocol:http:1.0
  Service Address: http://localhost:9000/services/inbox-b
  Message Binding: urn:taxii.mitre.org:message:xml:1.0
  Message Binding: urn:taxii.mitre.org:message:xml:1.1
              Inbox       Service       AC:       ['urn:stix.mitre.org:xml:1.1.1',
'urn:custom.example.com:json:0.0.1']
  Available: True
  Message: Custom Inbox Service Description B

=== Service Instance ===
  Service Type: DISCOVERY
  Service Version: urn:taxii.mitre.org:services:1.1
  Protocol Binding: urn:taxii.mitre.org:protocol:http:1.0
  Service Address: http://localhost:9000/services/discovery-a
  Message Binding: urn:taxii.mitre.org:message:xml:1.0
  Message Binding: urn:taxii.mitre.org:message:xml:1.1
  Available: True
  Message: Custom Discovery Service description

..... snip .....
```

One of the configured services is a Collection Management Service. To get the collections list, run:

```
(venv) $ taxii-collections --path http://localhost:9000/services/collection-
management-a
```

You should see the following output:

```
2018-06-03 21:30:03,315 INFO: Sending Collection_Information_Request to http://
localhost:9000/services/collection-management-a
=== Data Collection Information ===
  Collection Name: collection-a
  Collection Type: DATA_SET
  Available: True
  Collection Description: None
  Supported Content: All
  === Polling Service Instance ===
    Poll Protocol: urn:taxii.mitre.org:protocol:http:1.0
    Poll Address: http://localhost:9000/services/poll-a
    Message Binding: urn:taxii.mitre.org:message:xml:1.0
    Message Binding: urn:taxii.mitre.org:message:xml:1.1
  === Subscription Service ===
    Protocol Binding: urn:taxii.mitre.org:protocol:http:1.0
    Address: http://localhost:9000/services/collection-management-a
    Message Binding: urn:taxii.mitre.org:message:xml:1.0
    Message Binding: urn:taxii.mitre.org:message:xml:1.1
  === Subscription Service ===
    Protocol Binding: urn:taxii.mitre.org:protocol:https:1.0
    Address: https://localhost:9000/services/collection-management-a
    Message Binding: urn:taxii.mitre.org:message:xml:1.0
    Message Binding: urn:taxii.mitre.org:message:xml:1.1
  === Receiving Inbox Service ===
    Protocol Binding: urn:taxii.mitre.org:protocol:http:1.0
    Address: http://localhost:9000/services/inbox-a
    Message Binding: urn:taxii.mitre.org:message:xml:1.0
    Message Binding: urn:taxii.mitre.org:message:xml:1.1
    Supported Contents: All
==================================


=== Data Collection Information ===
  Collection Name: collection-b
  Collection Type: DATA_FEED
  Available: True
  Collection Description: None
  Supported Content:   urn:stix.mitre.org:xml:1.1.1
  === Polling Service Instance ===
    Poll Protocol: urn:taxii.mitre.org:protocol:http:1.0
    Poll Address: http://localhost:9000/services/poll-a
    Message Binding: urn:taxii.mitre.org:message:xml:1.0
    Message Binding: urn:taxii.mitre.org:message:xml:1.1
  === Subscription Service ===
    Protocol Binding: urn:taxii.mitre.org:protocol:http:1.0
    Address: http://localhost:9000/services/collection-management-a
    Message Binding: urn:taxii.mitre.org:message:xml:1.0
    Message Binding: urn:taxii.mitre.org:message:xml:1.1
  === Subscription Service ===
    Protocol Binding: urn:taxii.mitre.org:protocol:https:1.0
    Address: https://localhost:9000/services/collection-management-a
    Message Binding: urn:taxii.mitre.org:message:xml:1.0
    Message Binding: urn:taxii.mitre.org:message:xml:1.1
  === Receiving Inbox Service ===
    Protocol Binding: urn:taxii.mitre.org:protocol:http:1.0
    Address: http://localhost:9000/services/inbox-a
```

```
    Message Binding: urn:taxii.mitre.org:message:xml:1.0
    Message Binding: urn:taxii.mitre.org:message:xml:1.1
    Supported Contents: All
  === Receiving Inbox Service ===
    Protocol Binding: urn:taxii.mitre.org:protocol:http:1.0
    Address: http://localhost:9000/services/inbox-b
    Message Binding: urn:taxii.mitre.org:message:xml:1.0
    Message Binding: urn:taxii.mitre.org:message:xml:1.1
    Supported Content: urn:stix.mitre.org:xml:1.1.1
    Supported Content: urn:custom.example.com:json:0.0.1
==================================

.... snip .....
```

See Cabby documentation for more examples.

12.2 TAXII 2

12.2.1 HTTP

You can make HTTP requests directly to the server. The discovery endpoint is at `/taxii2/`. The trailing `/` is important.

Then, the other endpoints are the ones provided by the TAXII 2.1 specs.

12.2.2 Python

```python
import taxii2client

# With public discovery
server = taxii2client.Server(url="http://localhost:9000/taxii2/")

print(server.title)
for root in server.api_roots:
    print(root.url)

# With authentication
api_root = taxii2client.ApiRoot(
  root.url, user="<username>", password="<password>"
)
```

13. Health check

OpenTAXII has an endpoint that can be used to check health of the service:

```
$ curl http://localhost:9000/management/health
{
  "alive": true
}
```

13.1 Next steps

Continue to Authentication page to learn how OpenTAXII authentication process works.

III Contributing

14 Contribute and develop 43

15 Contributing and developing 45
15.1 Reporting issues 45
15.2 Obtaining the source code 45
15.3 Layout 45
15.4 Database schema 46
15.5 Set up a development environment 48
15.6 Running the tests 48
15.7 Generating the documentation 48

16 Running Development OpenTAXII . 49

14. Contribute and develop

- Contributing and developing
- Running Development OpenTAXII

15. Contributing and developing

15.1 Reporting issues

OpenTAXII uses Github's issue tracker. See the OpenTAXII project page on Github.

15.2 Obtaining the source code

The OpenTAXII source code can be found on Github. See the OpenTAXII project page.

15.3 Layout

OpenTAXII repository has the following layout:

- `docker/` - Docker configuration files (OpenTAXII Docker documentation);
- `docs/` - used to build the documentation;
- `examples/` - configuration and code examples;
- `opentaxii/` - OpenTAXII source;
- `tests/` - opentaxii tests.

15.4 Database schema

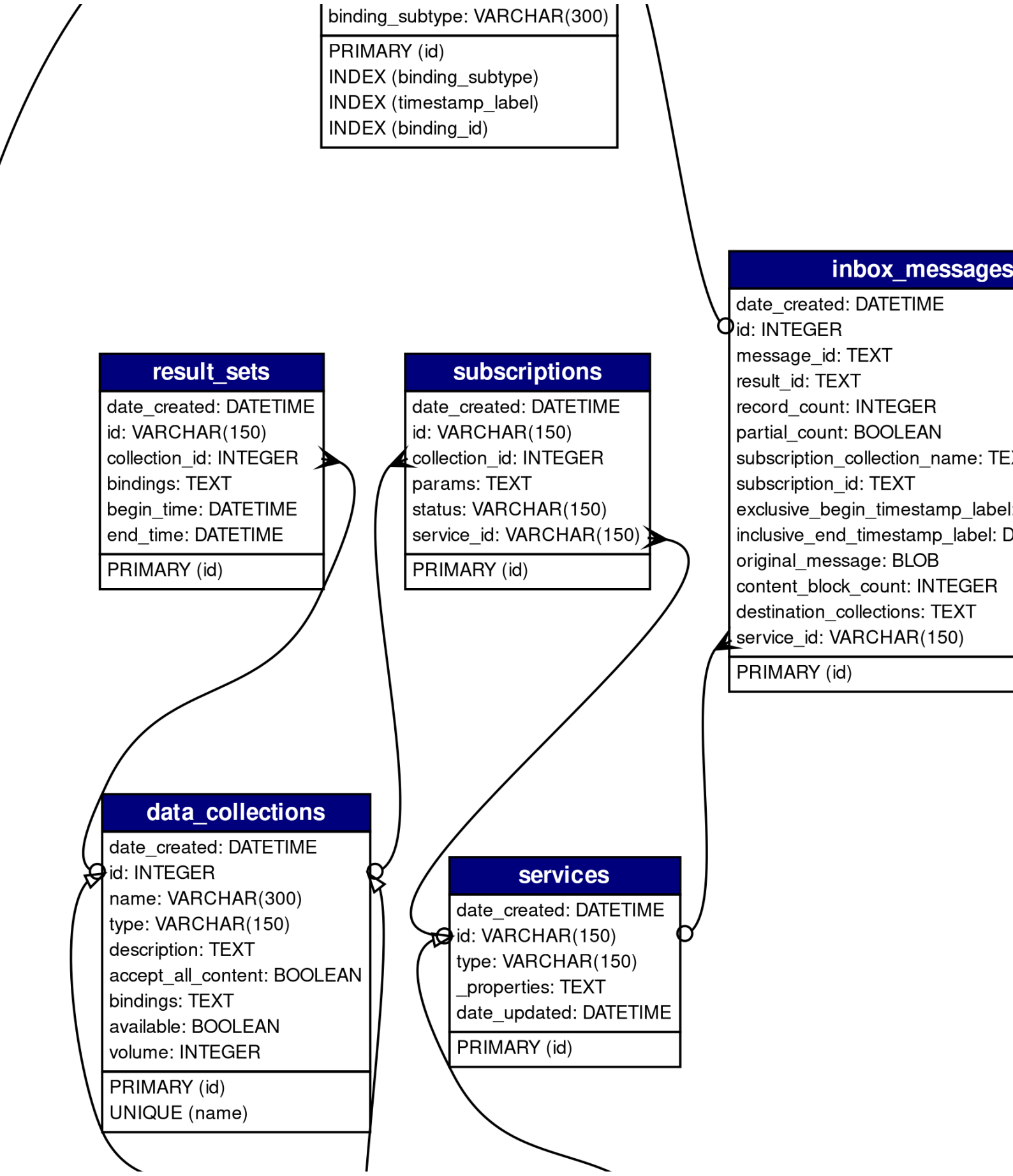


Figure 15.1: image

15.5 Set up a development environment

Install uv as described in the installation guide. From the repository root, create a virtual environment and install an editable copy of OpenTAXII with its development dependencies.

```
uv venv --python 3.11
uv pip install -r requirements-dev.txt -e .
```

OpenTAXII's package metadata lives in `setup.py`. Use `uv run --no-project` to run commands in `.venv` with the installed package.

15.6 Running the tests

Run the test suite from the repository root. It uses temporary SQLite databases by default.

```
uv run --no-project python -m pytest
```

The CI matrix in `tox.ini` also tests against MySQL, MariaDB, and PostgreSQL.

15.7 Generating the documentation

The documentation is written in Quarto Markdown and built with Quarto. From the repository root, install the documentation dependencies and render the book.

```
uv pip install -r docs/requirements.txt
uv run --no-project python docs/render-references.py
quarto render docs
```

Quarto writes the HTML, PDF, and EPUB to `docs/_book/`. Run `quarto preview docs` for a local preview.

GitHub Actions builds the book for pull requests and publishes it to GitHub Pages from `master`.

15.7.1 Next steps

Continue to License.

16. Running Development OpenTAXII

Here we describe how to run OpenTAXII in a *development*. Development mode activates Flask debug mode, simplifies log messages (for humans) and runs the server in one thread. While in production mode there are no debug messages, proper json log messages and multithreaded if configured.

From the repository root, run the development server in the environment created by the development setup.

```
uv run --no-project opentaxii-run-dev
```

OpenTAXII listens on `localhost:9000`.

IV Reference

17 References 53

18 API reference 55
18.1 Configuration and server 55
18.2 TAXII 1 entities and services 64
18.3 TAXII 2 entities and validation 78
18.4 Persistence, authentication, and signals 87

19 Public code-level APIs 107
19.1 Overview 107
19.2 Custom API implementations 107
19.3 Custom signal listeners 107
19.4 Persistence API 107
19.5 Authentication API 107
19.6 Signals 107

20 Changelog 109
20.1 0.10.0 (unreleased) 109
20.2 0.9.3 (2022-10-11) 109
20.3 0.9.2 (2022-08-26) 109
20.4 0.9.1 (2022-07-11) 109
20.5 0.9.0 (2022-06-13) 109
20.6 0.8.0 (2022-06-05) 109
20.7 0.7.0 (2022-05-27) 110
20.8 0.6.0 (2022-05-25) 110
20.9 0.5.0 (2022-05-24) 110
20.10 0.4.0 (2022-05-20) 110
20.11 0.3.0 (2022-04-13) 110
20.12 0.3.0a4 (2022-04-13) 110
20.13 0.3.0a3 (2022-01-21) 110
20.14 0.3.0a2 (2021-12-27) 110
20.15 0.3.0a1 110
20.16 0.3.0a0 110
20.17 0.2.4 (2022-04-13) 110
20.18 0.2.3 (2021-12-22) 110
20.19 0.2.2 (2021-11-05) 110
20.20 0.2.1 (2021-11-03) 111
20.21 0.2.0 (2020-06-30) 111
20.22 0.1.12 (2019-03-06) 111
20.23 0.1.11 (2019-02-13) 111
20.24 0.1.10 (2018-06-03) 111
20.25 0.1.9 (2017-06-19) 111
20.26 0.1.8 (2017-02-21) 111
20.27 0.1.7 (2016-10-18) 112
20.28 0.1.6 (2016-06-01) 112
20.29 0.1.5 (2016-03-15) 112
20.30 0.1.4 (2016-02-25) 112
20.31 0.1.3 (2016-02-25) 112
20.32 0.1.2 (2015-07-24) 112
20.33 0.1.1 (2015-04-08) 112
20.34 0.1.0 (2015-03-31) 112

21 License 113

17. References

- API reference
- Public code-level APIs
- Changelog

18. API reference

This reference documents the Python classes, methods, functions, and attributes used by OpenTAXII. For protocol terminology, see the TAXII project and the TAXII 2.1 specification.

18.1 Configuration and server

Load configuration, create the Flask application, and access the server interfaces.

18.1.1 config

`config`

18.1.1.1 Attributes

Name	Description
`CONFIG_ENV_VAR`	
`DEFAULT_CONFIG`	
`DEFAULT_CONFIG_NAME`	
`ENV_VAR_PREFIX`	
`current_dir`	

18.1.1.2 Classes

Name	Description
ServerConfig	Class responsible for loading configuration files.

ServerConfig

```
config.ServerConfig(optional_env_var=CONFIG_ENV_VAR, extra_configs=None)
```

Class responsible for loading configuration files.

This class will load default configuration file (shipped with OpenTAXII) and apply user specified configuration file on top of default one.

Users can specify custom configuration file (YAML formatted) using enviromental variable. The variable should contain a full path to a custom configuration file.

Parameters

Name	Type	Description	Default
optional_env_var	str	name of the enviromental variable	`CONFIG_ENV_VAR`
extra_configs	list	list of additional config filenames	`None`

Attributes

Name	Description
`ALL_VALID_OPTIONS`	

Name	Description
VALID_BASE_OPTIONS	
VALID_TAXII1_OPTIONS	
VALID_TAXII2_OPTIONS	
VALID_TAXII_OPTIONS	

18.1.1.3 Functions

Name	Description
merge	

merge

```
config.merge(d1, d2)
```

18.1.2 server.TAXIIServer

```
server.TAXIIServer(config)
```

TAXII Server class.

This is the main entrypoint for http requests. It keeps the Auth API manager instance and dispatches requests and error handling to :class:`TAXII1Server` and :class:`TAXII2Server`

18.1.2.1 Attributes

Name	Description
AUTH_MANAGER_CLASS	
app	
auth	
config	
current_server	
real_servers	
servers	

18.1.2.2 Methods

Name	Description
get_endpoint	Get first endpoint matching relative_path.
handle_http_exception	Dispatch http exception handling to appropriate taxii* server.
handle_internal_error	Dispatch internal error handling to appropriate taxii* server.
handle_request	Dispatch request to appropriate taxii* server.
handle_status_exception	Dispatch status exception handling to appropriate taxii* server.

Name	Description
handle_validation_exception	Dispatch validation exception handling to appropriate taxii* server.
init_app	Connect taxii1, taxii2 and auth to flask.
is_basic_auth_supported	Check if basic auth is a supported feature.
raise_unauthorized	Dispatch unauthorized handling to appropriate taxii* server.

get_endpoint

```
server.TAXIIServer.get_endpoint(relative_path)
```

Get first endpoint matching relative_path.

handle_http_exception

```
server.TAXIIServer.handle_http_exception(error)
```

Dispatch http exception handling to appropriate taxii* server.

handle_internal_error

```
server.TAXIIServer.handle_internal_error(error)
```

Dispatch internal error handling to appropriate taxii* server.

handle_request

```
server.TAXIIServer.handle_request(relative_path)
```

Dispatch request to appropriate taxii* server.

handle_status_exception

```
server.TAXIIServer.handle_status_exception(error)
```

Dispatch status exception handling to appropriate taxii* server.

handle_validation_exception

```
server.TAXIIServer.handle_validation_exception(error)
```

Dispatch validation exception handling to appropriate taxii* server.

init_app

```
server.TAXIIServer.init_app(app)
```

Connect taxii1, taxii2 and auth to flask.

is_basic_auth_supported

```
server.TAXIIServer.is_basic_auth_supported()
```

Check if basic auth is a supported feature.

raise_unauthorized

```
server.TAXIIServer.raise_unauthorized()
```

Dispatch unauthorized handling to appropriate taxii* server.

18.1.3 server.TAXII1Server

```
server.TAXII1Server(config)
```

TAXII1 Server class.

This class keeps Presistence API manager instance for TAXII1 and creates TAXII1 Service instances on request.

18.1.3.1 Parameters

Name	Type	Description	Default
config	dict	OpenTAXII1 server configuration	*required*

18.1.3.2 Attributes

Name	Description
`TYPE_TO_SERVICE`	
`config`	
`persistence`	

18.1.3.3 Methods

Name	Description
check_allowed_methods	
get_domain	Get domain either from request handler or config.
get_endpoint	Get first endpoint matching relative_path.
get_service	Get service by ID.
get_services	Get services registered with this TAXII server instance.
get_services_for_collection	Get list of services with type `service_type`, attached
handle_internal_error	Handle internal error and return appropriate response.
handle_request	Handle request and return appropriate response.
handle_status_exception	Handle status exception and return appropriate response.
raise_unauthorized	Handle unauthorized access.

check_allowed_methods

```
server.TAXII1Server.check_allowed_methods()
```

get_domain

```
server.TAXII1Server.get_domain(service_id)
```

Get domain either from request handler or config.

get_endpoint

```
server.TAXII1Server.get_endpoint(relative_path)
```

Get first endpoint matching relative_path.

get_service

```
server.TAXII1Server.get_service(id)
```

Get service by ID.

Parameters

Name	Type	Description	Default
id	str	service ID	*required*

Returns

Name	Type	Description
	:py:class:`opentaxii.taxii.services.abstract.TAXIIService`	service with specified ID or None

get_services

```
server.TAXII1Server.get_services(service_ids=None)
```

Get services registered with this TAXII server instance.

Parameters

Name	Type	Description	Default
service_ids	list	list of service IDs (as strings)	`None`

Returns

Name	Type	Description
	list of :py:class:`opentaxii.taxii.services.abstract.TAXIIService`	list of services

get_services_for_collection

```
server.TAXII1Server.get_services_for_collection(collection, service_type)
```

Get list of services with type `service_type`, attached to collection `collection`.

Parameters

Name	Type	Description	Default
collection	`opentaxii.taxii.entities.CollectionEntity`	collection in question	*required*
service_type	str	service type, supported values are listed as keys in :py:attr:`TYPE_TO_SERVICE`	*required*

Returns

Name	Type	Description
	list of :py:class:`opentaxii.taxii.services.abstract.TAXIIService` # noqa	list of services

handle_internal_error

```
server.TAXII1Server.handle_internal_error(error)
```

Handle internal error and return appropriate response.

handle_request

```
server.TAXII1Server.handle_request(endpoint)
```

Handle request and return appropriate response.
Process :class:`TAXIIService` with either :meth:`_process_with_service` or :meth:`_process_options_request`.

handle_status_exception

```
server.TAXII1Server.handle_status_exception(error)
```

Handle status exception and return appropriate response.

raise_unauthorized

```
server.TAXII1Server.raise_unauthorized()
```

Handle unauthorized access.

18.1.4 server.TAXII2Server

```
server.TAXII2Server(config)
```

TAXII2 server class.
Stub, implementation pending.

18.1.4.1 Attributes

Name	Description
`config`	
`persistence`	

18.1.4.2 Methods

Name	Description
api_root_handler	
check_allowed_methods	
check_authentication	Check if account is authenticated, unless endpoint handles that itself.
check_content_length	
check_headers	
collection_handler	

Name	Description
collections_handler	
discovery_handler	
get_endpoint	
handle_http_exception	Return JSON instead of HTML for HTTP errors.
handle_request	
handle_validation_exception	Handle validation exception and return appropriate response.
job_handler	
manifest_handler	
object_delete_handler	
object_get_handler	
object_handler	
objects_get_handler	
objects_handler	
objects_post_handler	
raise_unauthorized	Handle unauthorized access.
versions_handler	

api_root_handler

```
server.TAXII2Server.api_root_handler(api_root_id)
```

check_allowed_methods

```
server.TAXII2Server.check_allowed_methods(endpoint)
```

check_authentication

```
server.TAXII2Server.check_authentication(endpoint)
```

Check if account is authenticated, unless endpoint handles that itself.

check_content_length

```
server.TAXII2Server.check_content_length()
```

check_headers

```
server.TAXII2Server.check_headers(endpoint)
```

collection_handler

```
server.TAXII2Server.collection_handler(api_root_id, collection_id_or_alias)
```

collections_handler

```
server.TAXII2Server.collections_handler(api_root_id)
```

discovery_handler

```
server.TAXII2Server.discovery_handler()
```

get_endpoint

```
server.TAXII2Server.get_endpoint(relative_path)
```

handle_http_exception

```
server.TAXII2Server.handle_http_exception(error)
```

Return JSON instead of HTML for HTTP errors.

handle_request

```
server.TAXII2Server.handle_request(endpoint)
```

handle_validation_exception

```
server.TAXII2Server.handle_validation_exception(error)
```

Handle validation exception and return appropriate response.

job_handler

```
server.TAXII2Server.job_handler(api_root_id, job_id)
```

manifest_handler

```
server.TAXII2Server.manifest_handler(api_root_id, collection_id_or_alias)
```

object_delete_handler

```
server.TAXII2Server.object_delete_handler(
    api_root_id,
    collection_id_or_alias,
    object_id,
)
```

object_get_handler

```
server.TAXII2Server.object_get_handler(
    api_root_id,
    collection_id_or_alias,
    object_id,
)
```

object_handler

```
server.TAXII2Server.object_handler(
    api_root_id,
    collection_id_or_alias,
```

```
    object_id,
)
```

objects_get_handler

```
server.TAXII2Server.objects_get_handler(api_root_id, collection_id_or_alias)
```

objects_handler

```
server.TAXII2Server.objects_handler(api_root_id, collection_id_or_alias)
```

objects_post_handler

```
server.TAXII2Server.objects_post_handler(api_root_id, collection_id_or_alias)
```

raise_unauthorized

```
server.TAXII2Server.raise_unauthorized()
```

Handle unauthorized access.

versions_handler

```
server.TAXII2Server.versions_handler(
    api_root_id,
    collection_id_or_alias,
    object_id,
)
```

18.1.5 entities

`entities`

18.1.5.1 Classes

Name	Description
Account	Represents Account entity.

Account

```
entities.Account(id, username, permissions, is_admin=False, **details)
```

Represents Account entity.
This class holds user-specific information and is used for authorization.
The permissions are defined as follow:

1. key: A collection identifier. For TAXII1 collection, it is a string name while it is a stringified UUID for TAXII2.
2. value: The permission level. For TAXII1 collection, it is a string with either “read” or “modify” value. However for TAXII2, it is a list with one or both values: “read” and “write”. The reason is that TAXII2 allows to write without read access and vice versa.

Attributes

Name	Description
`details`	

Name	Description
`id`	
`is_admin`	
`permissions`	
`username`	

18.1.6 middleware.create_app

```
middleware.create_app(server)
```

Create Flask application and attach TAXII server instance `server` to it.

18.1.6.1 Parameters

Name	Type	Description	Default
server	`opentaxii.server.TAXIIServer`	TAXII server instance	*required*

18.1.6.2 Returns

Name	Type	Description
		Flask app

18.2 TAXII 1 entities and services

Work with TAXII 1 entities and service implementations.

18.2.1 taxii.entities

`taxii.entities`

18.2.1.1 Classes

Name	Description
CollectionEntity	TAXII Collection entity.
ContentBindingEntity	TAXII Content Binding entity.
ContentBlockEntity	TAXII Content Block entity.
InboxMessageEntity	TAXII Inbox Message Entity
PollRequestParametersEntity	TAXII Poll Request Parameters entity.
ResultSetEntity	TAXII Result Set entity.
ServiceEntity	TAXII Service entity.
SubscriptionEntity	TAXII Subscription entity.
SubscriptionParameters	TAXII Subscription Parameters entity.

CollectionEntity

```
taxii.entities.CollectionEntity(
    name,
    id=None,
    description=None,
    type=TYPE_FEED,
    volume=None,
    accept_all_content=False,
    supported_content=None,
    available=True,
)
```

TAXII Collection entity.

Parameters

Name	Type	Description	Default
id	str	collection id	None
name	str	collection name	*required*
description	str	description for the collection	None
type	:attr:TYPE_FEED | :attr:TYPE_SET	collection type	TYPE_FEED
accept_all_content	bool	if collection accepts all content types	False
supported_content	list	list of the supported content bindings as a list of :class:ContentBindingEntity instances	None
available	bool	if collection is available	True

Attributes

Name	Description
TYPE_FEED	
TYPE_SET	
accept_all_content	
available	
description	
id	
name	
supported_content	
type	
volume	

Methods

Name	Description
can_modify	
can_read	

Name	Description
get_matching_bindings	
is_content_supported	

can_modify

```
taxii.entities.CollectionEntity.can_modify(account)
```

can_read

```
taxii.entities.CollectionEntity.can_read(account)
```

get_matching_bindings

```
taxii.entities.CollectionEntity.get_matching_bindings(requested_bindings)
```

is_content_supported

```
taxii.entities.CollectionEntity.is_content_supported(content_binding)
```

ContentBindingEntity

```
taxii.entities.ContentBindingEntity(binding, subtypes=None)
```

TAXII Content Binding entity.

Parameters

Name	Type	Description	Default
binding	str	content binding ID	*required*
subtypes	list	list of subtype ids	`None`

Attributes

Name	Description
`binding`	
`subtypes`	

ContentBlockEntity

```
taxii.entities.ContentBlockEntity(
    content,
    timestamp_label,
    content_binding=None,
    id=None,
    message=None,
    inbox_message_id=None,
)
```

TAXII Content Block entity.

Parameters

Name	Type	Description	Default
id	str	content block ID	`None`
content	str	payload	*required*
timestamp_label	datetime	content block timestamp	*required*
content_binding	`ContentBindingEntity`	content binding	`None`
message	str	message attached to the content block	`None`
inbox_message_id	str	internal ID of the inbox message entity	`None`

Attributes

Name	Description
`content`	
`content_binding`	
`id`	
`inbox_message_id`	
`message`	
`timestamp_label`	

InboxMessageEntity

```
taxii.entities.InboxMessageEntity(
    message_id,
    original_message,
    content_block_count,
    service_id,
    id=None,
    result_id=None,
    destination_collections=None,
    record_count=None,
    partial_count=False,
    subscription_collection_name=None,
    subscription_id=None,
    exclusive_begin_timestamp_label=None,
    inclusive_end_timestamp_label=None,
)
```

TAXII Inbox Message Entity

Parameters

Name	Type	Description	Default
message_id	str	TAXII message ID	*required*
original_message	str	XML serialized original TAXII message	*required*
content_block_count	int	how many content blocks this message contains	*required*
service_id	str	ID of the Inbox Service that received the message	*required*

Name	Type	Description	Default
id	str	internal ID of the inbox message entity	`None`
result_id	str	ID of the Result Set part of which this message delivers	`None`
destination_collections	list	a list of destination collections, as a list of strings	`None`
record_count	int	how many items left in the Result Set this message is part of	`None`
partial_count	bool	if the record count is partial	`False`
subscription_id	str	ID of a subscription	`None`
subscription_collection_name	str	collection name of the subscription	`None`
exclusive_begin_timestamp_label	label-time	subscription's exclusive begin timestamp label	`None`
inclusive_begin_timestamp_label	label-time	subscription's inclusive begin timestamp label	*required*

Attributes

Name	Description
`content_block_count`	
`destination_collections`	
`exclusive_begin_timestamp_label`	
`id`	
`inclusive_end_timestamp_label`	
`message_id`	
`original_message`	
`partial_count`	
`record_count`	
`result_id`	
`service_id`	
`subscription_collection_name`	
`subscription_id`	

PollRequestParametersEntity

```
taxii.entities.PollRequestParametersEntity(
    response_type=SubscriptionParameters.FULL,
    content_bindings=None,
)
```

TAXII Poll Request Parameters entity.
Note: allow_asynch and delivery_parameters fields are not supported

Parameters

Name	Type	Description	Default
response_type	str	response type, supported values are :attr:`FULL` and :attr:`COUNT_ONLY`	`SubscriptionParameters.FULL`
content_bindings	list	list of :class:`ContentBindingEntity` instances	`None`

ResultSetEntity

```
taxii.entities.ResultSetEntity(
    id,
    collection_id,
    content_bindings=None,
    timeframe=None,
)
```

TAXII Result Set entity.

Parameters

Name	Type	Description	Default
id	str	ID of a Result Set	*required*
collection_id	str	ID of a collection	*required*
content_bindings	list	list of :class:`ContentBindingEntity` instances	`None`
timeframe	tuple	a timeframe of the Result Set in a form of `(begin, end)`	`None`

Attributes

Name	Description
`collection_id`	
`content_bindings`	
`id`	
`timeframe`	

ServiceEntity

```
taxii.entities.ServiceEntity(type, properties, id=None)
```

TAXII Service entity.

Parameters

Name	Type	Description	Default
type	str	service type, supported values are listed as keys in :py:attr:`opentaxii.server.TAXIIServer.TYPE_TO_SERVICE`	*required*
properties	dict	a dictionary with service-specific properties	*required*
id	str	service ID	`None`

Attributes

Name	Description
`id`	
`properties`	
`type`	

SubscriptionEntity

```
taxii.entities.SubscriptionEntity(
    service_id,
    collection_id,
    subscription_id=None,
    status=ACTIVE,
    poll_request_params=None,
)
```

TAXII Subscription entity.

Parameters

Name	Type	Description	Default
service_id	str	ID of a service	*required*
collection_id	str	ID of a collection	*required*
subscription_id	str	ID of a subscription	`None`
status	str	subscription status, supported values are: :attr:`ACTIVE`, :attr:`PAUSED`, :attr:`UNSUBSCRIBED`	`ACTIVE`
poll_request_params	`PollRequestParameters`	Poll Request Parameters entity	`None`

Attributes

Name	Description
`ACTIVE`	
`PAUSED`	
`UNSUBSCRIBED`	
`collection_id`	
`params`	
`service_id`	
`status`	
`subscription_id`	

SubscriptionParameters

```
taxii.entities.SubscriptionParameters(response_type=FULL, content_bindings=None)
```

TAXII Subscription Parameters entity.
Note: query formats specification is not supported

Parameters

Name	Type	Description	Default
response_type	str	response type, supported values are :attr:`FULL` and :attr:`COUNT_ONLY`	`FULL`
content_bindings	list	list of :class:`ContentBindingEntity` instances	`None`

Attributes

Name	Description
`COUNT_ONLY`	
`FULL`	
`content_bindings`	
`response_type`	

18.2.1.2 Functions

Name	Description
deserialize_content_bindings	

deserialize_content_bindings

```
taxii.entities.deserialize_content_bindings(supported_content)
```

18.2.2 taxii.services.abstract

`taxii.services.abstract`

18.2.2.1 Classes

Name	Description
TAXIIService	Generic TAXII Service class.

TAXIIService

```
taxii.services.abstract.TAXIIService(
    id,
    server,
    address,
    description=None,
    path=None,
    protocol_bindings=None,
    available=True,
    authentication_required=False,
)
```

Generic TAXII Service class.

This class implements common methods for all TAXII services.

Parameters

Name	Type	Description	Default
id	str	service ID	*required*

Name	Type	Description	Default
address	str	service address as absolute URL	*required*
description	str	service description	`None`
protocol_bindings	list	list of supported protocol bindings as a list of strings	`None`
available	bool	if the service is available	`True`
authentication_required	bool	if authentication required	`False`
path	str	relative path if service is configured in the server	`None`

Attributes

Name	Description
`address`	
`authentication_required`	
`available`	
`description`	
`id`	
`log`	
`path`	
`server`	
`supported_message_bindings`	
`supported_protocol_bindings`	

Methods

Name	Description
generate_id	
get_absolute_address	
get_message_handler	
process	
to_service_instances	

generate_id

```
taxii.services.abstract.TAXIIService.generate_id()
```

get_absolute_address

```
taxii.services.abstract.TAXIIService.get_absolute_address(binding)
```

get_message_handler

```
taxii.services.abstract.TAXIIService.get_message_handler(message)
```

process

```
taxii.services.abstract.TAXIIService.process(headers, message)
```

to_service_instances

```
taxii.services.abstract.TAXIIService.to_service_instances(version)
```

18.2.3 taxii.services.discovery

`taxii.services.discovery`

18.2.3.1 Classes

Name	Description
DiscoveryService	

DiscoveryService

```
taxii.services.discovery.DiscoveryService(services=None, **kwargs)
```

Attributes

Name	Description
`advertised_services`	
`handlers`	
`service_type`	

Methods

Name	Description
set_advertised_services	

set_advertised_services

```
taxii.services.discovery.DiscoveryService.set_advertised_services(services)
```

18.2.4 taxii.services.inbox

`taxii.services.inbox`

18.2.4.1 Classes

Name	Description
InboxService	

InboxService

```
taxii.services.inbox.InboxService(
    accept_all_content=False,
    destination_collection_required=False,
    supported_content=None,
```

```
    **kwargs,
)
```

Attributes

Name	Description
accept_all_content	
destination_collection_required	
handlers	
service_type	
supported_content	

Methods

Name	Description
get_destination_collections	
get_supported_content	
is_content_supported	
to_service_instances	
validate_destination_collection_names	

get_destination_collections

```
taxii.services.inbox.InboxService.get_destination_collections()
```

get_supported_content

```
taxii.services.inbox.InboxService.get_supported_content(version)
```

is_content_supported

```
taxii.services.inbox.InboxService.is_content_supported(
    content_binding,
    version=None,
)
```

to_service_instances

```
taxii.services.inbox.InboxService.to_service_instances(version)
```

validate_destination_collection_names

```
taxii.services.inbox.InboxService.validate_destination_collection_names(
    name_list,
    in_response_to,
)
```

18.2.5 taxii.services.poll

`taxii.services.poll`

18.2.5.1 Attributes

Name	Description
DEFAULT_MAX_RESULT_COUNT	
DEFAULT_MAX_RESULT_SIZE	
log	

18.2.5.2 Classes

Name	Description
PollService	

PollService

```
taxii.services.poll.PollService(
    subscription_required=False,
    max_result_size=-1,
    max_result_count=-1,
    **kwargs,
)
```

Attributes

Name	Description
can_push	
handlers	
max_result_count	
max_result_size	
service_type	
subscription_required	
wait_time	

Methods

Name	Description
create_result_set	
get_collection	
get_content_blocks	
get_content_blocks_count	
get_offset_limit	
get_result_set	
get_subscription	

create_result_set

```
taxii.services.poll.PollService.create_result_set(
    collection,
```

```
    content_bindings=None,
    timeframe=None,
)
```

get_collection

```
taxii.services.poll.PollService.get_collection(name)
```

get_content_blocks

```
taxii.services.poll.PollService.get_content_blocks(
    collection,
    timeframe=None,
    content_bindings=None,
    part_number=1,
)
```

get_content_blocks_count

```
taxii.services.poll.PollService.get_content_blocks_count(
    collection,
    timeframe=None,
    content_bindings=None,
)
```

get_offset_limit

```
taxii.services.poll.PollService.get_offset_limit(part_number)
```

get_result_set

```
taxii.services.poll.PollService.get_result_set(result_set_id)
```

get_subscription

```
taxii.services.poll.PollService.get_subscription(subscription_id)
```

18.2.6 taxii.services.collection_management

`taxii.services.collection_management`

18.2.6.1 Classes

Name	Description
CollectionManagementService	

CollectionManagementService

```
taxii.services.collection_management.CollectionManagementService(
    subscription_supported=True,
    subscription_message=None,
    **kwargs,
)
```

Attributes

Name	Description
advertised_collections	
handlers	
service_type	
subscription_handlers	
subscription_message	
subscription_supported	

Methods

Name	Description
create_subscription	
get_collection	
get_polling_services	
get_push_methods	
get_receiving_inbox_services	
get_subscription	
get_subscription_services	
get_subscriptions	
update_subscription	

create_subscription

```
taxii.services.collection_management.CollectionManagementService.create_subscription(
    subscription,
)
```

get_collection

```
taxii.services.collection_management.CollectionManagementService.get_collection(
    name,
)
```

get_polling_services

```
taxii.services.collection_management.CollectionManagementService.get_polling_services(
    collection,
)
```

get_push_methods

```
taxii.services.collection_management.CollectionManagementService.get_push_methods(
    collection,
)
```

get_receiving_inbox_services

```
taxii.services.collection_management.CollectionManagementService.get_receiving_inbox_services(
    collection,
)
```

get_subscription

```
taxii.services.collection_management.CollectionManagementService.get_subscription(
    subscription_id,
)
```

get_subscription_services

```
taxii.services.collection_management.CollectionManagementService.get_subscription_services(
    collection,
)
```

get_subscriptions

```
taxii.services.collection_management.CollectionManagementService.get_subscriptions(
)
```

update_subscription

```
taxii.services.collection_management.CollectionManagementService.update_subscription(
    subscription,
)
```

18.3 TAXII 2 entities and validation

Work with TAXII 2 resources and request validation.

18.3.1 taxii2.entities

`taxii2.entities`
Taxii2 entities.

18.3.1.1 Classes

Name	Description
ApiRoot	TAXII2 API Root entity.
Collection	TAXII2 Collection entity.
Job	TAXII2 Job entity, called a “status resource” in taxii2 docs.
JobDetail	TAXII2 JobDetail entity, part of “status resource” in taxii2 docs.
JobDetails	
ManifestRecord	TAXII2 ManifestRecord entity.
STIXObject	TAXII2 STIXObject entity.
VersionRecord	TAXII2 VersionRecord entity.

ApiRoot

```
taxii2.entities.ApiRoot(id, default, title, description, is_public)
```

TAXII2 API Root entity.

Parameters

Name	Type	Description	Default
id	uuid.UUID	id of this API root	*required*
default	bool	indicator of default api root, should only be True once	*required*
title	str	human readable plain text name used to identify this API Root	*required*
description	Optional[str]	human readable plain text description for this API Root	*required*
is_public	bool	whether this is a publicly readable API root	*required*

Attributes

Name	Description
`default`	
`description`	
`id`	
`is_public`	
`title`	

Collection

```
taxii2.entities.Collection(
    id,
    api_root_id,
    title,
    description,
    alias,
    is_public,
    is_public_write,
)
```

TAXII2 Collection entity.

Parameters

Name	Type	Description	Default
id	uuid.UUID	id of this collection	*required*
api_root_id	uuid.UUID	id of the :class:`ApiRoot` this collection belongs to	*required*
title	str	human readable plain text name used to identify this collection	*required*
description	str	human readable plain text description for this collection	*required*

Name	Type	Description	Default
alias	Optional[str]	human readable collection name that can be used on systems to alias a collection id	*required*
is_public	bool	whether this is a publicly readable collection	*required*
is_public_write	bool	whether this is a publicly writable collection	*required*

Attributes

Name	Description
`alias`	
`api_root_id`	
`description`	
`id`	
`is_public`	
`is_public_write`	
`title`	

Methods

Name	Description
can_read	Determine if `account` is allowed to read from this collection.
can_write	Determine if `account` is allowed to write to this collection.

can_read

```
taxii2.entities.Collection.can_read(account)
```

Determine if `account` is allowed to read from this collection.

can_write

```
taxii2.entities.Collection.can_write(account)
```

Determine if `account` is allowed to write to this collection.

Job

```
taxii2.entities.Job(
    id,
    api_root_id,
    status,
    request_timestamp,
    completed_timestamp=None,
    total_count=0,
    success_count=0,
    failure_count=0,
    pending_count=0,
```

```
    details=None,
)
```

TAXII2 Job entity, called a “status resource” in taxii2 docs.

Parameters

Name	Type	Description	Default
id	uuid.UUID	id of this job	*required*
api_root_id	uuid.UUID	id of the :class:`ApiRoot` this collection belongs to	*required*
status	str	status of this job	*required*
request_timestamp	datetime	the datetime of the request that this status resource is monitoring	*required*
completed_timestamp	Optional[datetime]	the datetime of the completion of this job (used for cleanup)	`None`
total_count	int	the total number of stix objects in this job	`0`
success_count	int	the number of successful stix objects in this job	`0`
failure_count	int	the number of failed stix objects in this job	`0`
pending_count	int	the number of pending stix objects in this job	`0`
details	Optional[JobDetails]	the details per status of this job	`None`

Attributes

Name	Description
`api_root_id`	
`completed_timestamp`	
`details`	
`failure_count`	
`id`	
`pending_count`	
`request_timestamp`	
`status`	
`success_count`	
`total_count`	

Methods

Name	Description
as_taxii2_dict	Turn this object into a taxii2 dict.

as_taxii2_dict

```
taxii2.entities.Job.as_taxii2_dict()
```

Turn this object into a taxii2 dict.

JobDetail

```
taxii2.entities.JobDetail(id, job_id, stix_id, version, message, status)
```

TAXII2 JobDetail entity, part of "status resource" in taxii2 docs.

Parameters

Name	Type	Description	Default
id	uuid.UUID	id of this job detail	*required*
job_id	uuid.UUID	id of the job this detail belongs to	*required*
stix_id	str	id of the :class:`STIXObject` this detail tracks	*required*
version	datetime	the version of this object	*required*
message	str	message indicating more information about the object being created, its pending state, or why the object failed to be created.	*required*
status	str	status of this job	*required*

Attributes

Name	Description
`id`	
`job_id`	
`message`	
`status`	
`stix_id`	
`version`	

Methods

Name	Description
as_taxii2_dict	Turn this object into a taxii2 dict.

as_taxii2_dict

```
taxii2.entities.JobDetail.as_taxii2_dict()
```

Turn this object into a taxii2 dict.

JobDetails

```
taxii2.entities.JobDetails()
```

Attributes

Name	Description
`failure`	
`pending`	
`success`	

ManifestRecord

```
taxii2.entities.ManifestRecord(id, date_added, version, spec_version)
```

TAXII2 ManifestRecord entity.
This is a cut-down version of :class:`STIXObject`, for efficiency.

Parameters

Name	Type	Description	Default
id	str	id of this stix object	*required*
date_added	datetime	the date and time this object was added	*required*
version	datetime	the version of this object	*required*
spec_version	str	stix version this object matches	*required*

Attributes

Name	Description
`date_added`	
`id`	
`spec_version`	
`version`	

STIXObject

```
taxii2.entities.STIXObject(
    id,
    collection_id,
    type,
    spec_version,
    date_added,
    version,
    serialized_data,
)
```

TAXII2 STIXObject entity.

Parameters

Name	Type	Description	Default
id	str	id of this stix object	*required*
collection_id	uuid.UUID	id of the :class:`Collection` this stix object belongs to	*required*
type	str	type of this stix object	*required*
spec_version	str	stix version this object matches	*required*

Name	Type	Description	Default
date_added	datetime	the date and time this object was added	*required*
version	datetime	the version of this object	*required*
serialized_data	dict	the payload of this object	*required*

Attributes

Name	Description
`collection_id`	
`date_added`	
`id`	
`serialized_data`	
`spec_version`	
`type`	
`version`	

VersionRecord

```
taxii2.entities.VersionRecord(date_added, version)
```

TAXII2 VersionRecord entity.
This is a cut-down version of :class:`STIXObject`, for efficiency.

Parameters

Name	Type	Description	Default
date_added	datetime	the date and time this object was added	*required*
version	datetime	the version of this object	*required*

Attributes

Name	Description
`date_added`	
`version`	

18.3.2 taxii2.validation

`taxii2.validation`
Taxii2 validation functions.

18.3.2.1 Classes

Name	Description
DeleteFilterParamsSchema	Schema for the object delete endpoint filters.
ListFilterParamsSchema	Schema for the object list endpoint filters.
ObjectFilterParamsSchema	Schema for the object endpoint filters.
PersistenceApiMxin	Store persistence api on schema instance, to reference in `Taxii2Next`

Name	Description
Taxii2DateTime	Taxii2 formatting compliant datetime field.
Taxii2Filter	General taxii2 filter implementation.
Taxii2Next	Implemenatation of the taxii2 `next` query param.
Taxii2VersionFilter	Taxii2 compliant version filter.
VersionFilterParamsSchema	Schema for the versions endpoint filters.

DeleteFilterParamsSchema

```
taxii2.validation.DeleteFilterParamsSchema()
```

Schema for the object delete endpoint filters.

Attributes

Name	Description
`match_spec_version`	
`match_version`	

ListFilterParamsSchema

```
taxii2.validation.ListFilterParamsSchema(persistence_api, *args, **kwargs)
```

Schema for the object list endpoint filters.

Attributes

Name	Description
`added_after`	
`limit`	
`match_id`	
`match_spec_version`	
`match_type`	
`match_version`	
`next_kwargs`	

ObjectFilterParamsSchema

```
taxii2.validation.ObjectFilterParamsSchema(persistence_api, *args, **kwargs)
```

Schema for the object endpoint filters.

Attributes

Name	Description
`added_after`	
`limit`	
`match_spec_version`	
`match_version`	

Name	Description
next_kwargs	

PersistenceApiMxin

```
taxii2.validation.PersistenceApiMxin(persistence_api, *args, **kwargs)
```

Store persistence api on schema instance, to reference in `Taxii2Next`

Attributes

Name	Description
persistence_api	

Taxii2DateTime

```
taxii2.validation.Taxii2DateTime()
```

Taxii2 formatting compliant datetime field.

Attributes

Name	Description
DEFAULT_FORMAT	

Taxii2Filter

```
taxii2.validation.Taxii2Filter()
```

General taxii2 filter implementation.

Taxii2Next

```
taxii2.validation.Taxii2Next()
```

Implemenatation of the taxii2 `next` query param.

Taxii2VersionFilter

```
taxii2.validation.Taxii2VersionFilter()
```

Taxii2 compliant version filter.

VersionFilterParamsSchema

```
taxii2.validation.VersionFilterParamsSchema(persistence_api, *args, **kwargs)
```

Schema for the versions endpoint filters.

Attributes

Name	Description
added_after	
limit	
match_spec_version	
next_kwargs	

18.3.2.2 Functions

Name	Description
validate_delete_filter_params	Validate and load filter params for the delete endpoint.
validate_envelope	Validate if `json_data` is a valid taxii2 envelope.
validate_list_filter_params	Validate and load filter params for the list endpoint.
validate_object_filter_params	Validate and load filter params for the object endpoint.
validate_versions_filter_params	Validate and load filter params for the versions endpoint.

validate_delete_filter_params

```
taxii2.validation.validate_delete_filter_params(filter_params)
```

Validate and load filter params for the delete endpoint.

validate_envelope

```
taxii2.validation.validate_envelope(json_data, allow_custom=False)
```

Validate if `json_data` is a valid taxii2 envelope.

Parameters

Name	Type	Description	Default
json_data	Union[str, bytes]	the data to check	*required*
allow_custom	bool	if true, allow non-standard stix types	`False`

validate_list_filter_params

```
taxii2.validation.validate_list_filter_params(filter_params, persistence_api)
```

Validate and load filter params for the list endpoint.

validate_object_filter_params

```
taxii2.validation.validate_object_filter_params(filter_params, persistence_api)
```

Validate and load filter params for the object endpoint.

validate_versions_filter_params

```
taxii2.validation.validate_versions_filter_params(
    filter_params,
    persistence_api,
)
```

Validate and load filter params for the versions endpoint.

18.4 Persistence, authentication, and signals

Extend storage and authentication or subscribe to server signals.

18.4.1 persistence.api

`persistence.api`

18.4.1.1 Classes

Name	Description
OpenTAXII2PersistenceAPI	Abstract class that represents OpenTAXII Persistence API.
OpenTAXIIPersistenceAPI	Abstract class that represents OpenTAXII Persistence API.

OpenTAXII2PersistenceAPI

```
persistence.api.OpenTAXII2PersistenceAPI()
```

Abstract class that represents OpenTAXII Persistence API.
Stub, pending implementation.

Methods

Name	Description
add_objects	
delete_object	
get_api_root	
get_api_roots	Parse provided `next_param` into kwargs to be used to filter stix objects.
get_collection	
get_collections	
get_job_and_details	
get_manifest	
get_next_param	Get value for `next` based on :class:`Dict` instance.
get_object	Get single object from database.
get_objects	
get_versions	Get all versions of single object from database.
init_app	
parse_next_param	Parse provided `next_param` into kwargs to be used to filter stix objects.

add_objects

```
persistence.api.OpenTAXII2PersistenceAPI.add_objects(
    api_root_id,
    collection_id,
    objects,
)
```

delete_object

```
persistence.api.OpenTAXII2PersistenceAPI.delete_object(
    collection_id,
    object_id,
    match_version=None,
    match_spec_version=None,
)
```

get_api_root

```
persistence.api.OpenTAXII2PersistenceAPI.get_api_root(api_root_id)
```

get_api_roots

```
persistence.api.OpenTAXII2PersistenceAPI.get_api_roots()
```

Parse provided `next_param` into kwargs to be used to filter stix objects.

get_collection

```
persistence.api.OpenTAXII2PersistenceAPI.get_collection(
    api_root_id,
    collection_id_or_alias,
)
```

get_collections

```
persistence.api.OpenTAXII2PersistenceAPI.get_collections(api_root_id)
```

get_job_and_details

```
persistence.api.OpenTAXII2PersistenceAPI.get_job_and_details(
    api_root_id,
    job_id,
)
```

get_manifest

```
persistence.api.OpenTAXII2PersistenceAPI.get_manifest(
    collection_id,
    limit=None,
    added_after=None,
    next_kwargs=None,
    match_id=None,
    match_type=None,
    match_version=None,
    match_spec_version=None,
)
```

get_next_param

```
persistence.api.OpenTAXII2PersistenceAPI.get_next_param(kwargs)
```

Get value for `next` based on :class:`Dict` instance.

Parameters

Name	Type	Description	Default
		class:`Dict` kwargs: The dict to base the `next` param on	*required*

Returns

Name	Type	Description
	str	The value to use as `next` param

get_object

```
persistence.api.OpenTAXII2PersistenceAPI.get_object(
    collection_id,
    object_id,
    limit=None,
    added_after=None,
    next_kwargs=None,
    match_version=None,
    match_spec_version=None,
)
```

Get single object from database.
Should return `None` when object matching object_id doesn't exist.

get_objects

```
persistence.api.OpenTAXII2PersistenceAPI.get_objects(
    collection_id,
    limit=None,
    added_after=None,
    next_kwargs=None,
    match_id=None,
    match_type=None,
    match_version=None,
    match_spec_version=None,
)
```

get_versions

```
persistence.api.OpenTAXII2PersistenceAPI.get_versions(
    collection_id,
    object_id,
    limit=None,
    added_after=None,
    next_kwargs=None,
    match_spec_version=None,
)
```

Get all versions of single object from database.
Should return `None` when object matching object_id doesn't exist.

init_app

```
persistence.api.OpenTAXII2PersistenceAPI.init_app(app)
```

parse_next_param

```
persistence.api.OpenTAXII2PersistenceAPI.parse_next_param(next_param)
```

Parse provided `next_param` into kwargs to be used to filter stix objects.

OpenTAXIIPersistenceAPI

```
persistence.api.OpenTAXIIPersistenceAPI()
```

Abstract class that represents OpenTAXII Persistence API.
This class defines required methods that need to exist in a specific Persistence API implementation.

Methods

Name	Description
attach_collection_to_services	Attach collection to the services.
create_collection	Create a collection.
create_content_block	Create a content block.
create_inbox_message	Create an inbox message.
create_result_set	Create a result set.
create_service	Create a service.
create_subscription	Create a subscription.
delete_collection	Delete collection.
delete_content_blocks	Delete content blocks in a specified collection with
delete_service	Delete service. To implement in subclass
get_collection	Get a collection by name and service ID.
get_collections	Get the collections. If `service_id` is provided, return collection
get_content_blocks	Get the content blocks associated with a collection.
get_content_blocks_count	Get a count of the content blocks associated with a collection.
get_domain	Get configured domain needed to create absolute URLs.
get_result_set	Get a result set entity by ID.
get_services	Get configured services.
get_subscription	Get a subscription entity by ID.
get_subscriptions	Get the subscriptions attached to/ created via a service.
init_app	
set_collection_services	Set collection's services. To implement in subclass
update_collection	Update collection.
update_service	Update service. To implement in subclass

Name	Description
update_subscription	Update a subscription status.

attach_collection_to_services

```
persistence.api.OpenTAXIIPersistenceAPI.attach_collection_to_services(
    collection_id,
    service_ids,
)
```

Attach collection to the services.

NOTE: Additional data management method that is not used in TAXII server logic but only in helper scripts.

Parameters

Name	Type	Description	Default
collection_id	str	collection entity in question	*required*
service_ids	list	collection entity in question	*required*

Returns

Name	Type	Description
	:py:class:`opentaxii.taxii.entities.CollectionEntity`	Updated collection entity, with ID field not None

create_collection

```
persistence.api.OpenTAXIIPersistenceAPI.create_collection(collection_entity)
```

Create a collection.

NOTE: Additional data management method that is not used in TAXII server logic but only in helper scripts.

Parameters

Name	Type	Description	Default
collection_entity	`opentaxii.taxii.entities.CollectionEntity`	collection entity in question	*required*

Returns

Name	Type	Description
	:py:class:`opentaxii.taxii.entities.CollectionEntity`	Updated collection entity, with ID field not None

create_content_block

```
persistence.api.OpenTAXIIPersistenceAPI.create_content_block(
    content_block_entity,
    collection_ids=None,
    service_id=None,
)
```

Create a content block.

Parameters

Name	Type	Description	Default
content_block_entity		content block in question	*required*
collection_ids	list	a list of collection IDs as strings	`None`
service_id	str	ID of an inbox service via which content block was created	`None`

Returns

Name	Type	Description
	:py:class:`opentaxii.taxii.entities.ContentBlockEntity`	updated content block entity

create_inbox_message

```
persistence.api.OpenTAXIIPersistenceAPI.create_inbox_message(
    inbox_message_entity,
)
```

Create an inbox message.

Parameters

Name	Type	Description	Default
inbox_message_entity		inbox message entity in question	*required*

Returns

Name	Type	Description
	:py:class:`opentaxii.taxii.entities.InboxMessageEntity`	updated inbox message entity

create_result_set

```
persistence.api.OpenTAXIIPersistenceAPI.create_result_set(result_set_entity)
```

Create a result set.

Parameters

Name	Type	Description	Default
result_set_entity	`opentaxii.taxii.entities.ResultSetEntity`	result set entity in question	*required*

Returns

Name	Type	Description
	:py:class:`opentaxii.taxii.entities.ResultSetEntity`	updated result set entity

create_service

```
persistence.api.OpenTAXIIPersistenceAPI.create_service(service_entity)
```

Create a service.

NOTE: Additional data management method that is not used in TAXII server logic but only in helper scripts.

Parameters

Name	Type	Description	Default
service_entity	opentaxii.taxii.entities.ServiceEntity	service entity in question	*required*

Returns

Name	Type	Description
	:py:class:opentaxii.taxii.entities.ServiceEntity	updated service entity, with ID field not None

create_subscription

```
persistence.api.OpenTAXIIPersistenceAPI.create_subscription(subscription_entity)
```

Create a subscription.

Parameters

Name	Type	Description	Default
subscription_entity		subscription entity in question.	*required*

Returns

Name	Type	Description
	:py:class:opentaxii.taxii.entities.SubscriptionEntity	updated subscription entity

delete_collection

```
persistence.api.OpenTAXIIPersistenceAPI.delete_collection(collection_name)
```

Delete collection.

Parameters

Name	Type	Description	Default
collection_id	int	id of a collection object	*required*

delete_content_blocks

```
persistence.api.OpenTAXIIPersistenceAPI.delete_content_blocks(
    collection_name,
    start_time,
    end_time=None,
    with_messages=False,
)
```

Delete content blocks in a specified collection with timestamp label in a specified time frame.

Parameters

Name	Type	Description	Default
collection_name	str	collection name	*required*
start_time	datetime	exclusive beginning of a timeframe	*required*
end_time	datetime	inclusive end of a timeframe	None

Name	Type	Description	Default
with_messages	bool	delete related inbox messages	`False`

delete_service

```
persistence.api.OpenTAXIIPersistenceAPI.delete_service(service_id)
```

Delete service. To implement in subclass

get_collection

```
persistence.api.OpenTAXIIPersistenceAPI.get_collection(
    collection_name,
    service_id=None,
)
```

Get a collection by name and service ID.

Collection name is unique globally, so can be used as a key. Method retrieves collection entity using collection name `collection_name` and service ID `service_id` as a composite key.

Parameters

Name	Type	Description	Default
collection_name	str	a collection name	*required*
service_id	str	ID of a service	`None`

Returns

Name	Type	Description
	:py:class:`opentaxii.taxii.entities.CollectionEntity`	collection entity

get_collections

```
persistence.api.OpenTAXIIPersistenceAPI.get_collections(service_id=None)
```

Get the collections. If `service_id` is provided, return collection attached to a service.

Parameters

Name	Type	Description	Default
service_id	str	ID of a service in question	`None`

Returns

Name	Type	Description
	list of :py:class:`opentaxii.taxii.entities.CollectionEntity`	list of collection entities.

get_content_blocks

```
persistence.api.OpenTAXIIPersistenceAPI.get_content_blocks(
    collection_id,
    start_time=None,
    end_time=None,
    bindings=None,
    offset=0,
```

```
    limit=10,
)
```

Get the content blocks associated with a collection.

Parameters

Name	Type	Description	Default
collection_id	str	ID fo a collection in question	*required*
start_time	date-time	start of a time frame	`None`
end_time	date-time	end of a time frame	`None`
bindings	list	list of :py:class:`opentaxii.taxii.entities.ContentBindingEntity`	`None`
offset	int	result set offset	`0`
limit	int	result set max size	`10`

Returns

Name	Type	Description
	list of :py:class:`opentaxii.taxii.entities.ContentBlockEntity`	content blocks list

get_content_blocks_count

```
persistence.api.OpenTAXIIPersistenceAPI.get_content_blocks_count(
    collection_id,
    start_time=None,
    end_time=None,
    bindings=None,
)
```

Get a count of the content blocks associated with a collection.

Parameters

Name	Type	Description	Default
collection_id	str	ID fo a collection in question	*required*
start_time	date-time	start of a time frame	`None`
end_time	date-time	end of a time frame	`None`
bindings	list	list of :py:class:`opentaxii.taxii.entities.ContentBindingEntity`	`None`

Returns

Name	Type	Description
	int	content block count

get_domain

```
persistence.api.OpenTAXIIPersistenceAPI.get_domain(service_id)
```

Get configured domain needed to create absolute URLs.

Returns `None` by default.

Parameters

Name	Type	Description	Default
service_id	str	ID of a service	*required*

get_result_set

```
persistence.api.OpenTAXIIPersistenceAPI.get_result_set(result_set_id)
```

Get a result set entity by ID.

Parameters

Name	Type	Description	Default
result_set_id	str	ID of a result set.	*required*

Returns

Name	Type	Description
	:py:class:`opentaxii.taxii.entities.ResultSetEntity`	result set entity

get_services

```
persistence.api.OpenTAXIIPersistenceAPI.get_services(collection_id=None)
```

Get configured services.

Parameters

Name	Type	Description	Default
collection_id	str	get only services assigned to collection with provided ID	`None`

Returns

Name	Type	Description
	list of :py:class:`opentaxii.taxii.entities.ServiceEntity`	list of service entities.

get_subscription

```
persistence.api.OpenTAXIIPersistenceAPI.get_subscription(subscription_id)
```

Get a subscription entity by ID.

Parameters

Name	Type	Description	Default
subscription_id	str	ID of a subscription	*required*

Returns

Name	Type	Description
	:py:class:`opentaxii.taxii.entities.SubscriptionEntity`	subscription entity

get_subscriptions

```
persistence.api.OpenTAXIIPersistenceAPI.get_subscriptions(service_id)
```

Get the subscriptions attached to/created via a service.

Parameters

Name	Type	Description	Default
service_id	str	ID of a service	*required*

Returns

Name	Type	Description
	list of :py:class:`opentaxii.taxii.entities.SubscriptionEntity`	list of subscription entities

init_app

```
persistence.api.OpenTAXIIPersistenceAPI.init_app(app)
```

set_collection_services

```
persistence.api.OpenTAXIIPersistenceAPI.set_collection_services(
    collection_id,
    service_ids,
)
```

Set collection's services. To implement in subclass

update_collection

```
persistence.api.OpenTAXIIPersistenceAPI.update_collection(collection_entity)
```

Update collection.

Parameters

Name	Type	Description	Default
collection_entity	`opentaxii.taxii.entities.CollectionEntity`	collection entity object	*required*

Returns

Name	Type	Description
	:py:class:`opentaxii.taxii.entities.CollectionEntity`	updated collection entity

update_service

```
persistence.api.OpenTAXIIPersistenceAPI.update_service(obj)
```

Update service. To implement in subclass

update_subscription

```
persistence.api.OpenTAXIIPersistenceAPI.update_subscription(subscription_entity)
```

Update a subscription status.

Parameters

Name	Type	Description	Default
subscription_entity		subscription entity in question	*required*

Returns

Name	Type	Description
	:py:class:`opentaxii.taxii.entities.SubscriptionEntity`	updated subscription entity

18.4.2 persistence.sqldb.api

`persistence.sqldb.api`

18.4.2.1 Classes

Name	Description
SQLDatabaseAPI	SQL database implementation of OpenTAXII Persistence API.

SQLDatabaseAPI

```
persistence.sqldb.api.SQLDatabaseAPI(
    db_connection,
    create_tables=False,
    **engine_parameters,
)
```

SQL database implementation of OpenTAXII Persistence API.
Implementation will work with any DB supported by SQLAlchemy package.
Note: this implementation ignores `context.account` and does not have any access rules.

Parameters

Name	Type	Description	Default
db_connection	str	a string that indicates database dialect and connection arguments that will be passed directly to :func:`~sqlalchemy.engine.create_engine` method.	*required*
create_tables	bool	If True, tables will be created in the DB.	*required*
engine_parameters		If defined, these arguments would be passed to sqlalchemy.create_engine	*required*

Attributes

Name	Description
BASEMODEL	

Methods

Name	Description
create_collection	
create_content_block	
create_inbox_message	
create_result_set	

Name	Description
create_service	
create_subscription	
delete_collection	
delete_content_blocks	
delete_service	
get_collection	
get_collections	
get_content_blocks	
get_content_blocks_count	
get_result_set	
get_service	
get_services	
get_subscription	
get_subscriptions	
set_collection_services	
update_collection	
update_service	
update_subscription	

create_collection

```
persistence.sqldb.api.SQLDatabaseAPI.create_collection(entity)
```

create_content_block

```
persistence.sqldb.api.SQLDatabaseAPI.create_content_block(
    entity,
    collection_ids=None,
    service_id=None,
)
```

create_inbox_message

```
persistence.sqldb.api.SQLDatabaseAPI.create_inbox_message(entity)
```

create_result_set

```
persistence.sqldb.api.SQLDatabaseAPI.create_result_set(entity)
```

create_service

```
persistence.sqldb.api.SQLDatabaseAPI.create_service(entity)
```

create_subscription

```
persistence.sqldb.api.SQLDatabaseAPI.create_subscription(entity)
```

delete_collection

```
persistence.sqldb.api.SQLDatabaseAPI.delete_collection(collection_name)
```

delete_content_blocks

```
persistence.sqldb.api.SQLDatabaseAPI.delete_content_blocks(
    collection_name,
    start_time,
    end_time=None,
    with_messages=False,
)
```

delete_service

```
persistence.sqldb.api.SQLDatabaseAPI.delete_service(service_id)
```

get_collection

```
persistence.sqldb.api.SQLDatabaseAPI.get_collection(name, service_id=None)
```

get_collections

```
persistence.sqldb.api.SQLDatabaseAPI.get_collections(service_id=None)
```

get_content_blocks

```
persistence.sqldb.api.SQLDatabaseAPI.get_content_blocks(
    collection_id=None,
    start_time=None,
    end_time=None,
    bindings=None,
    offset=0,
    limit=None,
)
```

get_content_blocks_count

```
persistence.sqldb.api.SQLDatabaseAPI.get_content_blocks_count(
    collection_id=None,
    start_time=None,
    end_time=None,
    bindings=None,
)
```

get_result_set

```
persistence.sqldb.api.SQLDatabaseAPI.get_result_set(result_set_id)
```

get_service

```
persistence.sqldb.api.SQLDatabaseAPI.get_service(service_id)
```

get_services

```
persistence.sqldb.api.SQLDatabaseAPI.get_services(collection_id=None)
```

get_subscription

```
persistence.sqldb.api.SQLDatabaseAPI.get_subscription(subscription_id)
```

get_subscriptions

```
persistence.sqldb.api.SQLDatabaseAPI.get_subscriptions(service_id)
```

set_collection_services

```
persistence.sqldb.api.SQLDatabaseAPI.set_collection_services(
    collection_id,
    service_ids,
)
```

update_collection

```
persistence.sqldb.api.SQLDatabaseAPI.update_collection(entity)
```

update_service

```
persistence.sqldb.api.SQLDatabaseAPI.update_service(obj)
```

update_subscription

```
persistence.sqldb.api.SQLDatabaseAPI.update_subscription(entity)
```

18.4.3 auth.api

`auth.api`

18.4.3.1 Classes

Name	Description
OpenTAXIIAuthAPI	Abstract class that represents OpenTAXII Authentication API.

OpenTAXIIAuthAPI

```
auth.api.OpenTAXIIAuthAPI()
```

Abstract class that represents OpenTAXII Authentication API.

This class defines required methods that need to exist in a specific Authentication API implementation.

Methods

Name	Description
authenticate	Authenticate a user.
create_account	Create an account.
delete_account	

Name	Description
get_account	Get account for auth token.
get_accounts	
init_app	
update_account	Update an account.

authenticate

```
auth.api.OpenTAXIIAuthAPI.authenticate(username, password)
```

Authenticate a user.

Parameters

Name	Type	Description	Default
username	str	username	*required*
password	str	password	*required*

Returns

Name	Type	Description
	str | None	auth token

create_account

```
auth.api.OpenTAXIIAuthAPI.create_account(username, password, is_admin=False)
```

Create an account.

delete_account

```
auth.api.OpenTAXIIAuthAPI.delete_account(username)
```

get_account

```
auth.api.OpenTAXIIAuthAPI.get_account(token)
```

Get account for auth token.

Parameters

Name	Type	Description	Default
token	str	auth token	*required*

get_accounts

```
auth.api.OpenTAXIIAuthAPI.get_accounts()
```

init_app

```
auth.api.OpenTAXIIAuthAPI.init_app(app)
```

update_account

```
auth.api.OpenTAXIIAuthAPI.update_account(obj, password=None)
```

Update an account.

Parameters

Name	Type	Description	Default
obj	AccountEntity	an ipdated user entity (old one matched by username)	*required*
password	str | None	a new password	`None`

18.4.4 auth.sqldb.api

`auth.sqldb.api`

18.4.4.1 Classes

Name	Description
SQLDatabaseAPI	

SQLDatabaseAPI

```
auth.sqldb.api.SQLDatabaseAPI(
    db_connection,
    create_tables=False,
    secret=None,
    token_ttl_secs=None,
    **engine_parameters,
)
```

Attributes

Name	Description
`BASEMODEL`	
`secret`	
`token_ttl_secs`	

Methods

Name	Description
authenticate	
create_account	
delete_account	
get_account	
get_accounts	
update_account	

authenticate

```
auth.sqldb.api.SQLDatabaseAPI.authenticate(username, password)
```

create_account

```
auth.sqldb.api.SQLDatabaseAPI.create_account(username, password, is_admin=False)
```

delete_account

```
auth.sqldb.api.SQLDatabaseAPI.delete_account(username)
```

get_account

```
auth.sqldb.api.SQLDatabaseAPI.get_account(token)
```

get_accounts

```
auth.sqldb.api.SQLDatabaseAPI.get_accounts()
```

update_account

```
auth.sqldb.api.SQLDatabaseAPI.update_account(obj, password=None)
```

18.4.5 signals

`signals`

18.4.5.1 Attributes

Name	Description
`CONTENT_BLOCK_CREATED`	
`INBOX_MESSAGE_CREATED`	
`SUBSCRIPTION_CREATED`	

19. Public code-level APIs

This page describes OpenTAXII's public code-level APIs.

19.1 Overview

OpenTAXII ships with code-level APIs that can be extended by users. Built-in implementations use SQL database as a backend (everything that SQLAlchemy supports).

Additionaly, OpenTAXII supports anychronous notifications and users can attach custom listeners to the specific events.

19.2 Custom API implementations

It is possible to attach custom API implementations to OpenTAXII.

Custom API class should inherit base class (opentaxii.persistence.api.OpenTAXIIPersistenceAPI for Persistence API and opentaxii.auth.api.OpenTAXIIAuthAPI for Authentication API) and implement all defined methods.

Class constructor can accept any parameters. These parameters (as well as API class full name) have to be set in OpenTAXII configuration file. See example configuration for exact syntax. OpenTAXII will load the class from the `PYTHONPATH` and create API instance during server's start up procedure.

OpenTAXII APIs are documented below.

19.3 Custom signal listeners

Users can attach custom listeners for the events OpenTAXII emits. See Signals to find a list of supported signals.

To attach custom signals, specify full module name as a value for `hooks` field in OpenTAXII configuration file. Note that the module needs to be in OpenTAXII's `PYTHONPATH`.

Example of the implementation is provided in OpenTAXII GitHub repo - examples/hooks.py.

19.4 Persistence API

Persistence API takes care of all CRUD operations with entities and wraps backend storage layer.

See Configuration page for the details about how to attach custom implementation of Persistence API.

opentaxii.persistence.api

The persistence API is backed by the sqldb api.

opentaxii.persistence.sqldb.api

19.5 Authentication API

Authentication API represents an authority for authentication-related queries.

See Configuration page for the details about how to attach custom implementation of Authentication API.

opentaxii.auth.api

The authentication API is backed by the sqldb api.

opentaxii.auth.sqldb.api

19.6 Signals

Signals provide the ability for the user's code to receive asynchronous notification for predefined signals.

See Custom signal listeners chapter for the details about how to attach listeners for the signals.

`opentaxii.signals`

- `CONTENT_BLOCK_CREATED` — Emitted when a new content block is created.
- `INBOX_MESSAGE_CREATED` — Emitted when a new inbox message is created.
- `SUBSCRIPTION_CREATED` — Emitted when a new subscription is created.

20. Changelog

20.1 0.10.0 (unreleased)

Features:

- Add –public and –id params to the CLI opentaxii-add-api-root #283
- Enable TAXII 2.1 in the default configuration #293
- Improve get objects performance #297

Bug fixes:

- Fix UUID json serialization #289
- Returned `spec_version` was equal to the `type` #237 (@meetghodasara-crest) and #284
- Consider `--admin` flag when creating an account from the CLI with `opentaxii-create-account` #237
- TAXII 2.1 version supports now objects without a `modified` property. `created` is used as a fallback and if missing `01/01/1970` date is used instead. #240 (@MagsenAbbeThales)
- Deleted objects were not persisted in the database #275 (@meetghodasara-crest)
- `can_read` or `can_write` could be returned with the `null` value on a TAXII2 collection #292

Dependencies:

- Add Python 3.11 and 3.12 support
- Drop EOL Python 3.6, 3.7, 3.8 and 3.9 support
- Drop sqlalchemy 1.3 support
- Require gunicorn `>=22` for security reasons #269
- Remove mypy_extensions dependency

Development:

- Add Docker build & push action in the CI #296
- Add and enforce mypy type checker #289
- Add and enforce black and isort formatter #290

20.2 0.9.3 (2022-10-11)

- Add public write support.

20.3 0.9.2 (2022-08-26)

- Improve readability and navigation of docs (#238 thanks @zed-eiq for the improvement).

20.4 0.9.1 (2022-07-11)

- Implement raise_unauthorized for taxii2, this was missing and lead to 500 errors.

20.5 0.9.0 (2022-06-13)

- Allow custom properties. This can be disabled by config option `allow_custom_properties`

20.6 0.8.0 (2022-06-05)

- Nest details inside taxii2 job and allow counts without details

20.7 0.7.0 (2022-05-27)

- Nest taxii2 endpoints under /taxii2/

20.8 0.6.0 (2022-05-25)

- Add public_discovery option to taxii2 config
- Add support for publicly readable taxii 2 api roots

20.9 0.5.0 (2022-05-24)

- Add support for publicly readable taxii 2 collections

20.10 0.4.0 (2022-05-20)

- Move next_param handling into OpenTAXII2PersistenceAPI

20.11 0.3.0 (2022-04-13)

- Implement taxii2.1 support

20.12 0.3.0a4 (2022-04-13)

- Merge changes from 0.2.4 maintenance release

20.13 0.3.0a3 (2022-01-21)

- Fix bug that prevented booting with only taxii1 config (#217 thanks @azurekid for the report)

20.14 0.3.0a2 (2021-12-27)

- Merge changes from 0.2.3 maintenance release

20.15 0.3.0a1

- Add python 3.10 support

20.16 0.3.0a0

- Enablement for future taxii2 implementation
- Fix documentation build issues

20.17 0.2.4 (2022-04-13)

- Make sure werkzeug <2.1 and >=2.1 work correctly with auth system

20.18 0.2.3 (2021-12-22)

- Fix bug in multithreaded use of sqlite (#210 thanks @rohits144 for the report)

20.19 0.2.2 (2021-11-05)

- Fix readthedocs build

20.20 0.2.1 (2021-11-03)

- Add tests for python 3.6, 3.7, 3.8, 3.9, pypy
- Add tests for sqlite, mysql, mariadb, postgresql
- Fix bug that broke `delete_content_blocks` when using mysql on sqlalchemy 1.3
- Docs: Add db schema diagram
- Docs: Clarify how to get default data in a default (development) docker instance
- Fix implicit routing in TAXII 1.1 Inboxes
- Update jwt usage to pyjwt >= 2.0 (thanks @SanyaKapoor)

20.21 0.2.0 (2020-06-30)

- Enforce UTC usage in datetime fields in SQL DB Persistence API.
- Fix for #114: reintroduce `opentaxii-create-account` CLI command.
- Fix for #153: check if user can modify a collection before advertising it over inbox service.
- Multiple coding style fixes.
- Various documentation updates.

20.22 0.1.12 (2019-03-06)

- Remove unnecessary print statements.

20.23 0.1.11 (2019-02-13)

- Make JSON logging consistent when the application is run via Gunicorn.
- Set `acceptable_destination` key in status details instead of extended headers
- Allow passing `engine_parameters` to `SQLDatabaseAPI` for those who want to customize SQLAlchemy engine parameters.
- Require recent version of `lxml` for security reasons.
- Various test and Docker infrastructure improvements.

20.24 0.1.10 (2018-06-03)

- Replace separate service/collection/account creation process with single `opentaxii-sync-data` CLI command.
- Persistence and Auth APIs extended with missing CRUD methods, that are used by `opentaxii-sync-data`.
- Read/modify collection level ACL added.
- DB models for default implementation of Persistence API and Auth API were changed. No automatic migration code is provided (sorry!), so upgrading might require manual DB migration.
- Drop python2.7 from testing scope.
- Various bug fixes and improvements.

20.25 0.1.9 (2017-06-19)

- libtaxii dependency upgraded to 1.1.111.
- Various bug fixes and improvements (thanks to @bjigmp, @chorsley, @rjprins).

20.26 0.1.8 (2017-02-21)

- Ability to enable/disable "huge trees" support in XML parser. Configuration property `xml_parser_supports_huge_tree` set to `yes` or `true` will disable security restrictions and force XML parser to support very deep trees and very long text content.
- Adjust SQL Persistence API implemetation so it works smoothly with MySQL backend.
- Use Python 3.5 instead of Python 3.4 for tests.

20.27 0.1.7 (2016-10-18)

- Minor fixes.
- Dependencies were changed from hard-pinned to more flexible.
- Example of production DB configuration added to docs.

20.28 0.1.6 (2016-06-01)

- Python 3.4 compatibility of the codebase. Tox configuration extended with python 3.4 environment run.
- Flake8 full style compatibility. Flake8 check added to Tox configuration.
- SQLAlchemy session scope issue fixed (related to #38).
- opentaxii-delete-blocks CLI command added (related to #45).
- delete_content_blocks method added to Persistence API.
- Collection's name is required to be unique in default SQL DB Auth API implementation.

20.29 0.1.5 (2016-03-15)

- Fix for the issue with persistence backend returning `None` instead of `InboxMessage` object

20.30 0.1.4 (2016-02-25)

- Hard-coded dependencies in `setup.py` removed.

20.31 0.1.3 (2016-02-25)

- Versions of dependencies are pinned.
- Code adjusted for a new version of anyconfig API.
- Test for configuration loading added.

20.32 0.1.2 (2015-07-24)

- Docker configuration added.
- Health check endpoint added.
- Basic authentication support added.
- Temporary workaround for Issue #191.
- Method `get_domain` in Persistence API returns domain value configured for `service_id`. If nothing returned, value set in configuration file will be used.
- Performance optimisations.
- Bug fixes and style improvements.

20.33 0.1.1 (2015-04-08)

- Alias for Root Logger added to logging configuration.
- Context object in a request scope that holds account and token added.
- Support for OPTIONS HTTP request to enable auto version negotiation added.
- Documentation improved.

20.34 0.1.0 (2015-03-31)

- Initial release

21. License

Copyright (c) [2016], EclecticIQ All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.