pycsw Documentation

Release 3.0-dev

Tom Kralidis

2026-06-08

CONTENTS

Author
Tom Kralidis

Contact
tomkralidis at gmail.com

Release
3.0-dev

Date
2026-06-08

CHAPTER

ONE

INTRODUCTION

pycsw is an OGC API - Records and OGC CSW server implementation written in Python.

CHAPTER

TWO

FEATURES

- implements OGC API - Records - Part 1: Core
- implements OGC API - Records - Part 2: Facets
- implements OGC API - Records - Part 3: Create, Replace, Update, Delete, Harvest
- implements OGC API - Records - Part 4: Federated Search
- implements OGC API - Features - Part 3: Filtering
- implements STAC API
- implements Common Query Language (CQL2)
- certified OGC Compliant and OGC Reference Implementation for both CSW 2.0.2 and CSW 3.0.0
- harvesting support for WMS, WFS, WCS, WPS, WAF, CSW, SOS
- implements INSPIRE Discovery Services 3.0
- implements ISO Metadata Application Profile 1.0.0
- implements FGDC CSDGM Application Profile for CSW 2.0
- implements the Search/Retrieval via URL (SRU) search protocol
- implements Full Text Search capabilities
- implements OGC OpenSearch Geo and Time Extensions
- implements Open Archives Initiative Protocol for Metadata Harvesting
- implements Pub/Sub capability via OGC API Publish-Subscribe Workflow - Part 1: Core
- supports ISO, Dublin Core, DIF, FGDC, Atom, GM03 and DataCite metadata models
- CGI or WSGI deployment
- simple YAML configuration
- transactional capabilities (OGC API - Records and CSW-T)
- flexible repository configuration
- GeoNode connectivity
- HHypermap connectivity
- Open Data Catalog connectivity
- CKAN connectivity
- federated catalogue distributed searching

- realtime XML Schema validation
- extensible profile plugin architecture

2.1 Standards Support

Standard	Version(s)
OGC API - Records - Part 1: Core	1.0
OGC API - Features - Part 3: Filtering	draft
OGC API - Features - Part 4: Create, Replace, Update and Delete	draft
OGC API Publish-Subscribe Workflow - Part 1: Core	draft
OGC CSW	2.0.2/3.0.0
OGC Filter	1.1.0/2.0.0
OGC OWS Common	1.0.0/2.0.0
OGC GML	3.1.1
OGC SFSQL	1.2.1
OGC GeoRSS	1.0
Dublin Core	1.1
SOAP	1.2
ISO 19115	2003
ISO 19139	2007
ISO 19119	2005
NASA DIF	9.7
FGDC CSDGM	1998
GM03	2.1
SRU	1.1
OGC OpenSearch	1.0
OAI-PMH	2.0
DataCite	4.3

2.2 OGC API - Records support

- Part 1: Core

2.3 OGC API - Features support

- Part 3: Filtering
- Part 4: Create, Replace, Update and Delete

2.4 CQL

- Common Query Language (CQL2)

2.4.1 Supported Output Formats

- JSON (default)
- XML

2.4.2 Supported Filters

- q
- datetime
- filter / filter-lang (CQL)
- bbox
- all properties (`property=value`)

2.4.3 Paging

- limit
- offset

2.5 CSW Support

2.5.1 Supported Operations

Request	Optionality	Supported	HTTP method binding(s)
GetCapabilities	mandatory	yes	GET (KVP) / POST (XML) / SOAP
DescribeRecord	mandatory	yes	GET (KVP) / POST (XML) / SOAP
GetRecords	mandatory	yes	GET (KVP) / POST (XML) / SOAP
GetRecordById	optional	yes	GET (KVP) / POST (XML) / SOAP
GetRepositoryItem	optional	yes	GET (KVP)
GetDomain	optional	yes	GET (KVP) / POST (XML) / SOAP
Harvest	optional	yes	GET (KVP) / POST (XML) / SOAP
UnHarvest	optional	no	
Transaction	optional	yes	POST (XML) / SOAP

Note

Asynchronous processing supported for GetRecords and Harvest requests (via `csw:ResponseHandler`)

Note

Supported Harvest Resource Types are listed in *Transactions using CSW*

2.5.2 Supported Output Formats

- XML (default)
- JSON

2.5.3 Supported Output Schemas

- Dublin Core
- ISO 19139

- FGDC CSDGM
- NASA DIF
- Atom
- GM03
- DataCite

2.5.4 Supported Sorting Functionality

- ogc:SortBy
- ascending or descending
- aspatial (queryable properties)
- spatial (geometric area)

2.5.5 Supported Filters

2.5.6 Full Text Search

- csw:AnyText

2.5.7 Geometry Operands

- gml:Point
- gml:LineString
- gml:Polygon
- gml:Envelope

Note

Coordinate transformations are supported

2.5.8 Spatial Operators

- BBOX
- Beyond
- Contains
- Crosses
- Disjoint
- DWithin
- Equals
- Intersects
- Overlaps
- Touches
- Within

2.5.9 Logical Operators

- Between
- EqualTo
- LessThanEqualTo
- GreaterThan
- Like
- LessThan
- GreaterThanEqualTo
- NotEqualTo
- NullCheck

2.5.10 Functions

- length
- lower
- ltrim
- rtrim
- trim
- upper

2.6 OAI-PMH Support

2.6.1 Supported Operations

- GetRecord
- Identify
- ListIdentifiers
- ListMetadataFormats
- ListRecords
- ListSets

2.6.2 Supported Filters

- from
- until
- set

2.6.3 Paging

- resumptionToken

CHAPTER

THREE

INSTALLATION

3.1 System Requirements

pycsw is written in Python, and works with (tested) Python 3.

pycsw requires the following Python supporting libraries:

- lxml for XML support
- SQLAlchemy for database bindings
- pyproj for coordinate transformations
- PyYAML for configuration management
- Shapely for spatial query / geometry support
- OWSLib for CSW client and metadata parser
- xmltodict for working with XML similar to working with JSON
- geolinks for dealing with geospatial links

3.1.1 OGC API - Records

OGC API - Records support additionally requires the following:

- Flask for pycsw's default OGC API - Records endpoint
- pygeofilter for CQL parsing

> **Note**
>
> You can install these dependencies via pip

> **Note**
>
> For *GeoNode* or *Open Data Catalog* or *HHypermap* deployments, SQLAlchemy is not required

3.2 Installing from Source

Download the latest stable version or fetch from Git.

3.2.1 For Developers and the Truly Impatient

The 4 minute install:

```
virtualenv pycsw && cd pycsw && . bin/activate
git clone https://github.com/geopython/pycsw.git && cd pycsw
pip3 install -e . && pip3 install -r requirements-standalone.txt
cp default-sample.yml default.yml
vi default.yml
# adjust paths in
# server.home
# repository.database
# set server.url to http://localhost:8000/
# start server - CSW 2/3, OAI-PMH, OpenSearch, SRU (all endpoints at /)
python3 pycsw/wsgi.py
curl http://localhost:8000/?service=CSW&version=2.0.2&request=GetCapabilities
```

To enable OGC API - Records as well as the abovementioned search standards:

```
# configure which config file to use
export PYCSW_CONFIG=default.yml
# start server - OGC API - Records and all services (various endpoints below)
python3 pycsw/wsgi_flask.py
# OGC API - Records
curl http://localhost:8000
# OpenAPI document
curl http://localhost:8000/openapi
# OGC CSW 3
curl http://localhost:8000/csw
# OGC CSW 2
curl http://localhost:8000/csw?service=CSW&version=2.0.2&request=GetCapabilities
# OAI-PMH
curl http://localhost:8000/oaipmh
# OpenSearch
curl http://localhost:8000/opensearch
# SRU
curl http://localhost:8000/sru
```

3.2.2 The Quick and Dirty Way

```
git clone https://github.com/geopython/pycsw.git
```

Ensure that CGI is enabled for the install directory. For example, on Apache, if pycsw is installed in `/srv/www/htdocs/pycsw` (where the URL will be `http://host/pycsw/csw.py`), add the following to `httpd.conf`:

```
<Location /pycsw/>
 Options +FollowSymLinks +ExecCGI
 Allow from all
 AddHandler cgi-script .py
</Location>
```

> **Note**
>
> If pycsw is installed in `cgi-bin`, this should work as expected. In this case, the *tests* application must be moved to a different location to serve static HTML documents.

Make sure, you have all the dependencies from `requirements.txt and requirements-standalone.txt`

3.2.3 The Clean and Proper Way

```
git clone https://github.com/geopython/pycsw.git
cd pycsw
python3 setup.py build
python3 setup.py install
```

At this point, pycsw is installed as a library and requires a CGI `csw.py` or WSGI `pycsw/wsgi.py` script to be served into your web server environment (see below for WSGI configuration/deployment).

3.3 Installing from the Python Package Index (PyPI)

```
pip3 install pycsw
```

3.4 Installing from OpenSUSE Build Service

In order to install the pycsw package in openSUSE Leap (stable distribution), one can run the following commands as user `root`:

```
zypper -ar https://download.opensuse.org/repositories/Application:/Geo/openSUSE_Leap_15.
↪2/ GEO
zypper refresh
zypper install python-pycsw pycsw-cgi
```

In order to install the pycsw package in openSUSE Tumbleweed (rolling distribution), one can run the following commands as user `root`:

```
zypper -ar https://download.opensuse.org/repositories/Application:/Geo/openSUSE_
↪Tumbleweed/ GEO
zypper refresh
zypper install python-pycsw pycsw-cgi
```

An alternative method is to use the One-Click Installer.

3.5 Installing on Ubuntu/Mint

In order to install the most recent pycsw release to an Ubuntu-based distribution, one can use the UbuntuGIS Unstable repository by running the following commands:

```
sudo add-apt-repository ppa:ubuntugis/ubuntugis-unstable
sudo apt-get update
sudo apt-get install python-pycsw pycsw-cgi
```

Alternatively, one can use the UbuntuGIS Stable repository which includes older but very well tested versions:

sudo add-apt-repository ppa:ubuntugis/ppa sudo apt-get update sudo apt-get install python-pycsw pycsw-cgi

Note

Since Ubuntu 16.04 LTS Xenial release, pycsw is included by default in the official Multiverse repository.

3.6 Running on Windows

For Windows installs, change the first line of `csw.py` to:

```
#!/Users/USERNAME/AppData/Local/Programs/Python/Python36/python -u
```

Note

The use of `-u` is required to properly output gzip-compressed responses.

Note

`USERNAME` should match your username, and the Python version should match with your install (e.g. `Python36`).

Tip

MS4W (MapServer for Windows) as of its version 4.0 release includes pycsw, Apache's mod_wsgi, Python 3.7, and many other tools, all ready to use out of the box. After installing, you will find your local pycsw catalogue endpoint, and steps for further configuration, on your browser's localhost page. You can read more about pycsw inside MS4W here.

3.7 Security

By default, `default.yml` is at the root of the pycsw install. If pycsw is setup outside an HTTP server's `cgi-bin` area, this file could be read. The following options protect the configuration:

- move `default.yml` to a non HTTP accessible area, and modify `csw.py` to point to the updated location
- configure web server to deny access to the configuration. For example, in Apache, add the following to `httpd.conf`:

```
<Files ~ "\.(yml)$">
 order allow,deny
 deny from all
</Files>
```

3.8 Running on WSGI

pycsw supports the Web Server Gateway Interface (WSGI). To run pycsw in WSGI mode, use `pycsw/wsgi.py` in your WSGI server environment.

> **Note**
>
> `mod_wsgi` supports only the version of Python it was compiled with. If the target server already supports WSGI applications, pycsw will need to use the same Python version. WSGIDaemonProcess provides a `python-path` directive that may allow a virtualenv created from the Python version `mod_wsgi` uses.

Below is an example of configuring with Apache:

```
WSGIDaemonProcess host1 home=/var/www/pycsw processes=2
WSGIProcessGroup host1
WSGIScriptAlias /pycsw-wsgi /var/www/pycsw/wsgi.py
<Directory /var/www/pycsw>
  Order deny,allow
  Allow from all
</Directory>
```

or use the WSGI reference implementation:

```
python3 ./pycsw/wsgi.py
Serving on port 8000...
```

which will publish pycsw to `http://localhost:8000/`

CHAPTER

FOUR

DOCKER

4.1 Installation

pycsw provides an official Docker image which is made available on both the geopython Docker Hub and our GitHub Container Registry.

Either `IMAGE` can be called with the `docker` command, `geopython/pycsw` from DockerHub or `ghcr.io/geophython/pycsw` from the GitHub Container Registry. Examples below use `geopython/pygeoapi`.

Assuming you already have docker installed, you can get a pycsw instance up and running run with the default built-in configuration:

```
docker run -p 8000:8000 geopython/pycsw

# or

docker run -p 8000:8000 ghcr.io/geopython/pycsw
```

...then browse to http://localhost:8000

Docker will retrieve the pycsw image (if needed) and then start a new container listening on port 8000.

The default configuration will run pycsw with an sqlite repository backend loaded with some test data from the CITE test suite. You can use this to take pycsw for a test drive.

4.2 Inspect logs

The default configuration for the docker image outputs logs to stdout. This is common practice with docker containers and enables the inspection of logs with the `docker logs` command:

```
# run a pycsw container in the background
docker run \
    --name pycsw-test \
    --publish 8000:8000 \
    --detach \
    geopython/pycsw

# inspect logs
docker logs pycsw-test
```

> **Note**
>
> In order to have pycsw logs being sent to standard output you must set `server.logfile=` in the pycsw configuration file.

4.3 Using pycsw-admin.py

`pycsw-admin.py` can be executed on a running container by using `docker exec`:

```
docker exec -ti <running-container-id> pycsw-admin.py --help
```

4.4 Running custom pycsw containers

4.4.1 pycsw configuration

It is possible to supply a custom configuration file for pycsw as a bind mount or as a docker secret (in the case of docker swarm). The configuration file is searched at the value of the `PYCSW_CONFIG` environmental variable, which defaults to `/etc/pycsw/pycsw.yml`.

Supplying the configuration file via bind mount:

```
docker run \
    --name pycsw \
    --detach \
    --volume <path-to-local-pycsw.yml>:/etc/pycsw/pycsw.yml \
    --publish 8000:8000 \
    geopython/pycsw
```

Supplying the configuration file via docker secrets:

```
# first create a docker secret with the pycsw config file
docker secret create pycsw-config <path-to-local-pycsw.yml>
docker service create \
    --name pycsw \
    --secret src=pycsw-config,target=/etc/pycsw/pycsw.yml \
    --publish 8000:8000
    geopython/pycsw
```

4.4.2 sqlite repositories

The default database repository is the CITE database that is used for running pycsw's test suites. Docker volumes may be used to specify a custom sqlite database path. It should be mounted under `/var/lib/pycsw`:

```
# first create a docker volume for persisting the database when
# destroying containers
docker volume create pycsw-db-data
docker run \
    --volume db-data:/var/lib/pycsw \
    --detach \
    --publish 8000:8000
    geopython/pycsw
```

4.4.3 PostgreSQL repositories

Specifying a PostgreSQL repository is just a matter of configuring a custom pycsw.yml file with the correct specification.

Check pycsw's GitHub repository for an example of a docker compose/stack file that spins up a postgis database together with a pycsw instance.

4.5 Setting up a development environment with docker

Working on pycsw's code using docker enables an isolated environment that helps ensuring reproducibility while at the same time keeping your base system free from pycsw related dependencies. This can be achieved by:

- Cloning pycsw's repository locally;
- Starting up a docker container with appropriately set up bind mounts. In addition, the pycsw docker image supports a `reload` flag that turns on automatic reloading of the gunicorn web server whenever the code changes;
- Installing the development dependencies by using `docker exec` with the root user;

The following instructions set up a fully working development environment:

```
# clone pycsw's repo
git clone https://github.com/geopython/pycsw.git

# start a container for development
cd pycsw
docker run \
    --name pycsw-dev \
    --detach \
    --volume ${PWD}/pycsw:/usr/lib/python3.7/site-packages/pycsw \
    --volume ${PWD}/docs:/home/pycsw/docs \
    --volume ${PWD}/LICENSE.txt:/home/pycsw/LICENSE.txt \
    --volume ${PWD}/COMMITTERS.txt:/home/pycsw/COMMITTERS.txt \
    --volume ${PWD}/CONTRIBUTING.rst:/home/pycsw/CONTRIBUTING.rst \
    --volume ${PWD}/pycsw/plugins:/home/pycsw/pycsw/plugins \
    --publish 8000:8000 \
    geopython/pycsw --reload

# install additional dependencies used in tests and docs
docker exec \
    -ti \
    --user root \
    pycsw-dev pip3 install -r pycsw/requirements-dev.txt

# run tests (for example unit tests)
docker exec -ti pycsw-dev pytest -m unit pycsw

# build docs
docker exec -ti pycsw-dev sh -c "cd pycsw/docs && make html"
```

Note

The pycsw image uses a specific Python version and does not install pycsw in editable mode. As such it is not possible to use `tox`.

Since the docs directory is bind mounted from your host machine into the container, after building the docs you may inspect their content visually, for example by running:

```
firefox docs/_build/html/index.html
```

CHAPTER

FIVE

DOCKER COMPOSE

For Docker Compose deployment, run the following in `docker/compose`:

```
PYCSW_DOCKER_IMAGE=latest docker compose up
```

Note

The `PYCSW_DOCKER_IMAGE` setting is required to set the Docker image version/tag.

5.1 Scaling

To scale via Docker Compose, run the following in `docker/compose`:

```
PYCSW_DOCKER_IMAGE=latest docker compose -f docker-compose.scale.yml up
```

Note

In `docker/compose/docker-compose.scale.yml`, adjust the `services.pycsw.deploy` and services.pycsw.ports values to scale accordingly. The port range specified must match the number of replicas defined.

CHAPTER

SIX

KUBERNETES

For Kubernetes orchestration, run the following in `docker/kubernetes`:

```
make up
make open
```

CHAPTER

SEVEN

HELM

For Kubernetes deployment via Helm, run the following in `docker/helm`:

```
helm install pycsw .
minikube service pycsw --url
```

CHAPTER

EIGHT

CONFIGURATION

pycsw's runtime configuration is defined by `default.yml`. pycsw ships with a sample configuration (`default-sample.yml`). Copy the file to `default.yml` and edit the following:

server

- **home**: the full filesystem path to pycsw
- **url**: the URL of the resulting service
- **mimetype**: the MIME type when returning HTTP responses
- **language**: the ISO 639-1 language and ISO 3166-1 alpha2 country code of the service (e.g. `en-CA`, `fr-CA`, `en-US`)
- **encoding**: the content type encoding (e.g. `ISO-8859-1`, see https://docs.python.org/2/library/codecs.html#standard-encodings). Default value is 'UTF-8'
- **maxrecords**: the maximum number of records to return by default. This value is enforced if a CSW's client's `maxRecords` parameter is greater than `server.maxrecords` to limit capacity. See *MaxRecords Handling* for more information
- **level**: the logging level (see https://docs.python.org/library/logging.html#logging-levels)
- **logfile**: the full file path to the logfile
- **ogc_schemas_base**: base URL of OGC XML schemas tree file structure (default is http://schemas.opengis.net)
- **distributedsearch**: distributed catalogue endpoints to be used for distributed searching, if requested by the client (see *Distributed Searching*)
- **pretty_print**: whether to pretty print the output (`true` or `false`). Default is `false`
- **gzip_compresslevel**: gzip compression level, lowest is `1`, highest is `9`. Default is off. **NOTE**: if gzip compression is already enabled via your web server, do not enable this directive (or else the server will try to compress the response twice, resulting in degraded performance)
- **domainquerytype**: for GetDomain operations, how to output domain values. Accepted values are `list` and `range` (min/max). Default is `list`
- **domaincounts**: for GetDomain operations, whether to provide frequency counts for values. Accepted values are `true` and `False`. Default is `false`
- **smtp_host**: SMTP host for processing `csw:ResponseHandler` parameter via outgoing email requests (default is `localhost`)
- **smtp_user**: SMTP user name related to the account (default is '')
- **smtp_pass**: SMTP password related to the account (default is '')

- **smtp_ssl**: Option to choose between SMTP and SMTP_SSL. To enable it, set the value to `true` (default is `false`)
- **spatial_ranking**: parameter that enables (`true` or `false`) ranking of spatial query results as per K.J. Lanfear 2006 - A Spatial Overlay Ranking Method for a Geospatial Search of Text Objects.
- **workers**: set the number of workers used by the wsgi server when lunching pycsw using the provided docker/entrypoint.py. If not set, it will use 2 workers as Default.

profiles

- list of profiles to load at runtime (default is none). See *Profile Plugins*

manager

- **transactions**: whether to enable transactions (`true` or `false`). Default is `false` (see *Transactions using CSW*)
- **allowed_ips**: list of IP addresses (e.g. 192.168.0.103), wildcards (e.g. 192.168.0.*) or CIDR notations (e.g. 192.168.100.0/24) allowed to perform transactions (see *Transactions using CSW*)
- **csw_harvest_pagesize**: when harvesting other CSW servers, the number of records per request to page by (default is 10)

pubsub

- **broker**: Publish-Subscribe definition

pubsub.broker

- **show_link**: whether to display as a link in the landing page (`true` or `false`)
- **type**: type of broker
- **url**: endpoint of broker

> **Note**
>
> See *Publish-Subscribe integration (Pub/Sub)* for configuring your instance with Pub/Sub capability.

metadata

metadata.identification

- **title**: the title of the service
- **description**: some descriptive text about the service
- **keywords**: list of keywords about the service
- **keywords_type**: keyword type as per the ISO 19115 MD_KeywordTypeCode codelist). Accepted values are `discipline`, `temporal`, `place`, `theme`, `stratum`
- **fees**: fees associated with the service
- **accessconstraints**: access constraints associated with the service

metadata.provider

- **name**: the name of the service provider
- **url**: the URL of the service provider

metadata.contact

- **name**: the name of the provider contact

- **position**: the position title of the provider contact
- **address**: the address of the provider contact
- **city**: the city of the provider contact
- **stateorprovince**: the province or territory of the provider contact
- **postalcode**: the postal code of the provider contact (enclose in quotes)
- **country**: the country of the provider contact
- **phone**: the phone number of the provider contact (enclose in quotes)
- **fax**: the facsimile number of the provider contact (enclose in quotes)
- **email**: the email address of the provider contact
- **url**: the URL to more information about the provider contact
- **hours**: the hours of service to contact the provider
- **instructions**: the how to contact the provider contact
- **role**: the role of the provider contact as per the ISO 19115 CI_RoleCode codelist). Accepted values are `author`, `processor`, `publisher`, `custodian`, `pointOfContact`, `distributor`, `user`, `resourceProvider`, `originator`, `owner`, `principalInvestigator`

repository

- **database**: the full file path to the metadata database, in database URL format (see https://docs.sqlalchemy.org/en/latest/core/engines.html#database-urls)
- **table**: the table name for metadata records (default is `records`). If you are using PostgreSQL with a DB schema other than `public`, qualify the table like `myschema.table`
- **mappings**: custom repository mappings (see *Mapping to an Existing Repository*)
- **source**: the source of this repository only if not local (e.g. *GeoNode Configuration*, *Open Data Catalog Configuration*). Supported values are `geonode`, `odc`
- **filter**: server side database filter to apply as mask to all CSW requests (see *Repository Filters*)
- **max_retries**: max number of retry attempts when connecting to records-repository database
- **stable_sort**: enables stable sorting by appending an identifier sort as the last sortable. Default is `false`
- **facets**: list of facetable properties for search results

> **Note**
>
> See *Administration* for connecting your metadata repository and supported information models.

8.1 MaxRecords Handling

The The following describes how `maxRecords` is handled by the configuration when handling OGC API - Records items or CSW `GetRecords` requests:

server.maxrecords	OGC API - Records limit/CSW GetRecords.maxRecords	Result
none set	none passed	10 (CSW default)
20	14	20
20	none passed	20
none set	100	100
20	200	20

8.2 Using environment variables in configuration files

pycsw configuration supports using system environment variables, which can be helpful for deploying into 12 factor environments for example.

Below is an example of how to integrate system environment variables in pycsw:

```
repository:
    database: ${PYCSW_REPOSITORY_DATABASE_URI}
    table: ${MY_TABLE}
```

8.3 Alternate Configurations

By default, pycsw loads `default.yml` at runtime. To load an alternate configuration, modify `csw.py` to point to the desired configuration. Alternatively, pycsw supports explicitly specifiying a configuration by appending `config=/path/to/default.yml` to the base URL of the service (e.g. `http://localhost/pycsw/csw.py?config=tests/suites/default/default.yml&service=CSW&version=2.0.2&request=GetCapabilities`). When the `config` parameter is passed by a CSW client, pycsw will override the default configuration location and subsequent settings with those of the specified configuration.

This also provides the functionality to deploy numerous CSW servers with a single pycsw installation.

8.3.1 Hiding the Location

Some deployments with alternate configurations prefer not to advertise the base URL with the `config=` approach. In this case, there are many options to advertise the base URL.

Environment Variables

pycsw supports the following environment variables:

- `PYCSW_CONFIG`: specifies the filepath to a pycsw configuraiton

8.3.2 Configuration file location

One option is using Apache's `Alias` and `SetEnvIf` directives. For example, given the base URL `http://localhost/pycsw/csw.py?config=foo.yml`, set the following in your Apache configuration:

```
Alias /pycsw/csw-foo.py /var/www/pycsw/csw.py
SetEnvIf Request_URI "/pycsw/csw-foo.py" PYCSW_CONFIG=/var/www/pycsw/csw-foo.yml.
```

> **Note**
>
> Apache must be restarted after changes to configuration

pycsw will use the configuration as set in the `PYCSW_CONFIG` environment variable in the same manner as if it was specified in the base URL. Note that the configuration value `server.url` value must match the `Request_URI` value so as to advertise correctly in pycsw's Capabilities XML.

Wrapper Script

Another option is to write a simple wrapper (e.g. `csw-foo.sh`), which provides the same functionality and can be deployed without restarting Apache:

```
#!/bin/sh

export PYCSW_CONFIG=/var/www/pycsw/csw-foo.yml

/var/www/pycsw/csw.py
```

CHAPTER

NINE

ADMINISTRATION

pycsw administration is handled by the `pycsw-admin.py` utility. `pycsw-admin.py` is installed as part of the pycsw install process and should be available in your PATH.

Note

Run `pycsw-admin.py --help` to see all administration operations and parameters

9.1 Metadata Repository Setup

pycsw supports the following databases:

- SQLite3
- PostgreSQL (without PostGIS)
- PostgreSQL with PostGIS enabled
- MySQL

Note

The easiest and fastest way to deploy pycsw is to use SQLite3 as the backend. To use an SQLite in-memory database, in the pycsw configuration, set *repository.database* to `sqlite://`.

Note

PostgreSQL support includes support for PostGIS functions if enabled

Note

If PostGIS is activated before setting up the pycsw/PostgreSQL database, then native PostGIS geometries will be enabled.

To expose your geospatial metadata via pycsw, perform the following actions:

- setup the database
- import metadata

- publish the repository

9.2 Supported Information Models

By default, pycsw's API supports the core OGC API - Records and CSW Record information models. From the database perspective, the pycsw metadata model is loosely based on ISO 19115 and is able to transform to other formats as part of transformation during OGC API - Records/CSW requests.

> **Note**
>
> See *Profile Plugins* for information on enabling profiles

> **Note**
>
> See *Metadata Model Reference* for detailed information on pycsw's internal metadata model

9.3 Setting up the Database

```
pycsw-admin.py setup-repository --config default.yml
```

This will create the necessary tables and values for the repository.

The database created is an OGC SFSQL compliant database, and can be used with any implementing software. For example, to use with GDAL:

```
ogrinfo /path/to/records.db
INFO: Open of 'records.db'
using driver 'SQLite' successful.
1: records (Polygon)
ogrinfo -al /path/to/records.db
# lots of output
```

> **Note**
>
> If PostGIS is detected, the `pycsw-admin.py` script does not create the SFSQL tables as they are already in the database.

9.4 Loading Records

```
pycsw-admin.py load-records --config default.yml --path /path/to/records
```

This will import all `*.xml` records from `/path/to/records` into the database specified in `default.yml` (`repository.database`). Passing `-r` to the script will process `/path/to/records` recursively. Passing `-y` to the script will force overwrite existing metadata with the same identifier. Note that `-p` accepts either a directory path or single file.

> **Note**
>
> Records can also be imported using CSW-T (see *Transactions using CSW*).

9.5 Exporting the Repository

```
pycsw-admin.py export-records --config default.yml --path /path/to/output_dir
```

This will write each record in the database specified in `default.yml` (`repository.database`) to an XML document on disk, in directory `/path/to/output_dir`.

9.6 Optimizing the Database

```
pycsw-admin.py optimize-db --config default.yml
pycsw-admin.py rebuild-db-indexes --config default.yml
```

> **Note**
>
> This feature is relevant only for PostgreSQL and MySQL

9.7 Deleting Records from the Repository

```
pycsw-admin.py delete-records --config default.yml
```

This will empty the repository of all records.

9.8 Database Specific Notes

9.8.1 PostgreSQL

- To enable PostgreSQL support, the database user must be able to create functions within the database.
- PostgreSQL Full Text Search is supported for `csw:AnyText` based queries. pycsw creates a tsvector column based on the text from anytext column. Then pycsw creates a GIN index against the anytext_tsvector column. This is created automatically in `pycsw.core.repository.setup`. Any query against the OGC API - Records `q` parameter or CSW *csw:AnyText* or *apiso:AnyText* will process using PostgreSQL FTS handling

9.8.2 PostGIS

- pycsw makes use of PostGIS spatial functions and native geometry data type.
- It is advised to install the PostGIS extension before setting up the pycsw database
- If PostGIS is detected, the `pycsw-admin.py` script will create both a native geometry column and a WKT column, as well as a trigger to keep both synchronized
- In case PostGIS gets disabled, pycsw will continue to work with the WKT column
- In case of migration from plain PostgreSQL database to PostGIS, the spatial functions of PostGIS will be used automatically

- When migrating from plain PostgreSQL database to PostGIS, in order to enable native geometry support, a "GEOMETRY" column named "wkb_geometry" needs to be created manually (along with the update trigger in `pycsw.core.repository.setup`). Also the native geometries must be filled manually from the WKT field. Next versions of pycsw will automate this process

9.9 Mapping to an Existing Repository

pycsw supports publishing metadata from an existing repository. To enable this functionality, the default database mappings must be modified to represent the existing database columns mapping to the abstract core model (the default mappings are in `pycsw/core/config.py:StaticContext.md_core_model`).

To override the default settings:

- define a custom database mapping based on `etc/mappings.py`
- in `default.yml`, set `repository.mappings` to the location of the mappings.py file:

```
repository:
    ...
    mappings: path/to/mappings.py
```

Note you can also reference mappings as a Python object as a dotted path:

```
repository:
    ...
    mappings: path.to.pycsw_mappings
```

See the *GeoNode Configuration*, *HHypermap-Registry Configuration*, and *Open Data Catalog Configuration* for further examples.

9.9.1 Existing Repository Requirements

pycsw requires certain repository attributes and semantics to exist in any repository to operate as follows:

- `pycsw:Identifier`: unique identifier
- `pycsw:Typename`: typename for the metadata; typically the value of the root element tag (e.g. `csw:Record`, `gmd:MD_Metadata`)
- `pycsw:Schema`: schema for the metadata; typically the target namespace (e.g. `http://www.opengis.net/cat/csw/2.0.2`, `http://www.isotc211.org/2005/gmd`)
- `pycsw:InsertDate`: date of insertion
- `pycsw:XML`: full XML representation (deprecated; will be removed in a future release)
- `pycsw:Metadata`: full metadata representation
- `pycsw:MetadataType`: media type of metadata representation
- `pycsw:AnyText`: bag of XML element text values, used for full text search. Realized with the following design pattern:
  - capture all XML element and attribute values
  - store in repository
- `pycsw:BoundingBox`: string of WKT or EWKT geometry

The following repository semantics exist if the attributes are specified:

- `pycsw:Keywords`: comma delimited list of keywords

- `pycsw:Themes`: Text field of JSON list of objects with properties `concepts`, `scheme`

```
[
  {
    "concepts": [
      {
        "id": "atmosphericComposition"
      },
      {
        "id": "pollution"
      },
      {
        "id": "observationPlatform"
      },
      {
        "id": "rocketSounding"
      }
    ],
    "scheme": "https://wis.wmo.int/2012/codelists/WMOCodeLists.xml#WMO_CategoryCode"
  }
]
```

- `pycsw:Contacts`: Text field of JSON list of objects with properties as per the OGC API - Records party definition

```
[
  {
    "name": "contact",
    "individual": "Lastname, Firstname",
    "positionName": "Position Title",
    "contactInfo": {
      "phone": {
        "office": "+xx-xxx-xxx-xxxx"
      },
      "email": {
        "office": "you@example.org"
      },
      "address": {
        "office": {
          "deliveryPoint": "Mailing Address",
          "city": "City",
          "administrativeArea": "Administrative Area",
          "postalCode": "Zip or Postal Code",
          "country": "COuntry"
        },
        "onlineResource": {
          "href": "Contact URL"
        }
      },
      "hoursOfService": "Hours of Service",
      "contactInstructions": "During hours of service.  Off on weekends",
      "url": {
        "rel": "canonical",
        "type": "text/html",
```

(continues on next page)

(continued from previous page)

```
        "href": "https://example.org"
      }
    },
    "roles": [
      {
        "name": "pointOfContact"
      }
    ]
  }
]
```

- `pycsw:Links`: Text field of JSON list of objects with properties `name`, `description`, `protocol`, `url`

```
[
  {
    "name": "foo",
    "description": "bar",
    "protocol": "OGC:WMS",
    "url": "https://example.org/wms"
  }
]
```

Note

The `pycsw:Links` field should be a text type, not a JSON object type

- `pycsw:Bands`: Text field of JSON list of dicts with properties: `name`, `units`, `min`, `max`

```
[
  {
    "name": "B1",
    "units": "nm",
    "min": 0.1,
    "max": 0.333
  }
]
```

Note

The `pycsw:Bands` field should be a text type, not a JSON object type

Values of mappings can be derived from the following mechanisms:

- text fields
- Python datetime.datetime or datetime.date objects
- Python functions

Further information is provided in `pycsw/config.py:MD_CORE_MODEL`.

> **Note**
>
> See *Metadata Model Reference* for detailed information on pycsw's internal metadata model

9.9.2 Using a SQL View as the repository table

If your pre-existing database stores information in a normalized fashion, *i.e.* distributed on multiple tables rather than on a single table (which is what pycsw expects by default), you have the option to create a DB view and use that as pycsw's repository.

As a practical example, lets say you have a CKAN project which you would like to also provide pycsw integration with. CKAN stores dataset-related information over multiple tables:

- `package` - has base metadata fields for each dataset;
- `package_extra` - additional custom metadata fields, depending on the user's metadata schema;
- `package_tag` - dataset_related keywords;
- `tag` - dataset_related keywords;
- `group` - details about a dataset's owner organization;
- etc.

One way to adapt such a DB structure to be able to integrate with pycsw is to create a PostgreSQL Materialized View. For example:

```
CREATE MATERIALIZED VIEW IF NOT EXISTS my_pycsw_view AS
    WITH cte_extras AS (
        SELECT
               p.id,
               p.title,
               g.title AS org_name,
               json_object_agg(pe.key, pe.value) AS extras,
               array_agg(DISTINCT t.name) AS tags
               -- remaining columns omitted for brevity
        FROM package AS p
            JOIN package_extra AS pe ON p.id = pe.package_id
            JOIN "group" AS g ON p.owner_org = g.id
            JOIN package_tag AS pt ON p.id = pt.package_id
            JOIN tag AS t on pt.tag_id = t.id
        WHERE p.state = 'active'
         AND p.private = false
        GROUP BY p.id, g.title
    )
    SELECT
           c.id AS identifier,
           c.title AS title,
           c.org_name AS organization,
           ST_GeomFromGeoJSON(c.extras->>'spatial')::geometry(Polygon, 4326) AS geom,
           c.extras->>'reference_date' AS date,
           concat_ws(', ', VARIADIC c.tags) AS keywords
           -- remaining columns omitted for brevity
    FROM cte_extras AS c
WITH DATA;
```

Creating this SQL view in the database means that all we now have the CKAN dataset information all on a single flat table, ready for pycsw to integrate with.

A crucial setup that is required in order for SQL Views to be usable by pycsw is to include the additional `column_constraints` property in your custom mappings. This property is used to specify which column(s) should function as the primary key of the SQL View:

```
# contents of my_custom_pycsw_mappings.py
from sqlalchemy.schema import PrimaryKeyConstraint

MD_CORE_MODEL = {
    "column_constraints": (PrimaryKeyConstraint("identifier"),),
    "typename": "pycsw:CoreMetadata",
    "outputschema": "http://pycsw.org/metadata",
    "mappings": {
        "pycsw:Identifier": "identifier",
        # remaining mappings omitted for brevity
```

The above code snippet demonstrates how you could instruct sqlalchemy, which is what pycsw uses to interface with the DB, that the `identifier` column of the SQL view should be assumed to be the primary key of the table.

Finally, we can configure pycsw with the path to the custom mappings and the name of the SQL view:

```
# file: pycsw.yml

repository:
    database: postgresql://${DB_USERNAME}:${DB_PASSWORD}@${DB_HOST}/${DB_NAME}
    mappings: /path/to/my_custom_pycsw_mappings.py
    table: my_pycsw_view
```

CHAPTER

TEN

METADATA MODEL REFERENCE

pycsw's metadata repository model is designed to support queryable elements as well full-text search. The model is rooted in ISO 19115 and is updated as required to be able to support additional metadata models in a generic fashion.

10.1 Overview

10.2 Model Crosswalk

Table 1: pycsw model

Database column	pycsw mapping name	Queryable name	ISO 19115 (XPath)	CSW Record/Dub Core (XPath)	OGC API - Records (JSON-Path)	STAC (JSON-Path)	Description
`identifier`	`pycsw:Iden`	`apiso:Iden`	`gmd:fileId gco:Charac`	`dc:identif`	`id`	`id`	Record identifier (Primary key)
`typename`	`pycsw:Type`						CSW typename (e.g. `csw:Record`, `gmd:MD_Metadata`), see also ref:*existing-repository-requirements*

continues on next page

Table 1 – continued from previous page

Database column	pycsw mapping name	Queryable name	ISO 19115 (XPath)	CSW Record/Dub Core (XPath)	OGC API - Records (JSON-Path)	STAC (JSON-Path)	Description
schema	pycsw:Sche						Schema namespace, i.e. http://www.opengis.net/cat/csw/2.0.2, http://www.isotc211.org/2005/gmd, http://www.opengis.net/spec/ogcapi-records-1/1.0/req/record-core
mdsource	pycsw:MdSo						Origin of resource, either `local` (default), or URL to location of record/resource/service
insert_dat	`pycsw:Ins						Date of insertion of the metadata record, see also ref:*existing-repository-requirements*

continues on next page

Table 1 – continued from previous page

Database column	pycsw mapping name	Queryable name	ISO 19115 (XPath)	CSW Record/Dub Core (XPath)	OGC API - Records (JSON-Path)	STAC (JSON-Path)	Description
`xml`	`pycsw:XML`						Raw XML metadata as it was inserted into the repository. Note that this field is deprecated for the `metadata` field, and will be removed in a future release, see also ref:*existing-repository-requirements*
`metadata`	`pycsw:Meta`						Raw metadata payload (replaces `xml` field, supports any metadata type [JSON, XML, etc.]), see also ref:*existing-repository-requirements*
`metadata_t`	pycsw:Metad						Media type of the metadata payload (`application/xml` [default], `application/json` for OGC API - Records and STAC)

continues on next page

Table 1 – continued from previous page

Database column	pycsw mapping name	Queryable name	ISO 19115 (XPath)	CSW Record/Dub Core (XPath)	OGC API - Records (JSON-Path)	STAC (JSON-Path)	Description
anytext	pycsw:AnyT	apiso:AnyT	//text() (csw:AnyTe: [queryable only, not a return-able])	//text() (csw:AnyTe: [queryable only, not a return-able])	q= (for API)	q= (for API)	Bag of metadata element and attributes content ONLY, no XML tags or JSON element names, see also *Existing Repository Requirements*
language	pycsw:Lang	apiso:Lang	gmd:langua gmd:Langua gmd:langua gco:Charac	dc:languag	properties language	properties language	
title	pycsw:Titl	apiso:Titl	gmd:identi gmd:MD_Dat gmd:citati gmd:CI_Cit gmd:title/ gco:Charac	dc:title	properties title	properties title	
abstract	pycsw:Abst	apiso:Abst	gmd:identi gmd:MD_Dat gmd:abstra gco:Charac	dct:abstra	properties descriptio	properties descriptio	
edition	pycsw:Edit	apiso:Edit	gmd:identi gmd:MD_Dat gmd:citati gmd:CI_Cit gmd:editio gco:Charac				
keywords	pycsw:Keyw	apiso:Subj	gmd:identi gmd:MD_Ide gmd:descri gmd:MD_Key gmd:keywor gco:Charac gmd:identi gmd:MD_Dat gmd:topicC gmd:MD_Top	dc:subject	properties keywords		CSV of keywords, see also *Existing Repository Requirements*

continues on next page

Table 1 – continued from previous page

Database column	pycsw mapping name	Queryable name	ISO 19115 (XPath)	CSW Record/DubCore (XPath)	OGC API - Records (JSON-Path)	STAC (JSON-Path)	Description
keywordsty	pycsw:Keyw	apiso:Keyw	gmd:identi gmd:MD_Dat gmd:descri gmd:MD_Key gmd:type/ gmd:MD_Key				
themes	pycsw:Them				properties themes		JSON list of concepts/schemes, see also *Existing Repository Requirements*
format	pycsw:Form	apiso:Form	gmd:distri gmd:MD_Dis gmd:distri gmd:MD_For gmd:name/ gco:Charac	dc:format	properties formats		
source	pycsw:Sour			dc:source	properties externalId		
date	pycsw:Date			dc:date	time	properties datetime	
date_modif	pycsw:Modi	apiso:Modi	gmd:dateSt gco:Date	dct:modifi	properties updated	properties updated	
type	pycsw:Type	apiso:Type	gmd:hierar gmd:MD_Sco	dc:type	properties type		
wkt_geomet	pycsw:Boun	apiso:Boun	apiso:Boun	ows:Boundi	geometry	geometry	WKT/EWKT of geometry

continues on next page

Table 1 – continued from previous page

Database column	pycsw mapping name	Queryable name	ISO 19115 (XPath)	CSW Record/Dub Core (XPath)	OGC API - Records (JSON-Path)	STAC (JSON-Path)	Description
crs	pycsw:CRS	apiso:CRS	gmd:refere gmd:MD_Ref gmd:refere gmd:RS_Ide gmd:codeSp gco:Charac gmd:refere gmd:MD_Ref gmd:refere gmd:RS_Ide gmd:versio gco:Charac gmd:refere gmd:MD_Ref gmd:refere gmd:RS_Ide gmd:code/ gco:Charac	dct:spatia			
title_alte	pycsw:Alte	apiso:Alte	gmd:identi gmd:MD_Dat gmd:citati gmd:CI_Cit gmd:altern gco:Charac	dct:altern			
date_revis	pycsw:Revi	apiso:Revi	gmd:identi gmd:MD_Dat gmd:citati gmd:CI_Cit gmd:date/ gmd:CI_Dat gmd:CI_Dat @codeListV gmd:date/ gco:Date		properties updated	created or properties updated	
date_creat	pycsw:Crea	apiso:Crea	gmd:identi gmd:MD_Dat gmd:citati gmd:CI_Cit gmd:date/ gmd:CI_Dat gmd:CI_Dat @codeListV gmd:date/ gco:Date		properties created	created or properties created	

continues on next page

Table 1 – continued from previous page

Database column	pycsw mapping name	Queryable name	ISO 19115 (XPath)	CSW Record/Dub Core (XPath)	OGC API - Records (JSON-Path)	STAC (JSON-Path)	Description
date_publi	pycsw:Publ	apiso:Publ	gmd:identi gmd:MD_Dat gmd:citati gmd:CI_Cit gmd:date/ gmd:CI_Dat gmd:CI_Dat @codeListV gmd:date/ gco:Date				
organizati	pycsw:Orga	apiso:Orga	gmd:identi gmd:MD_Dat gmd:pointO gmd:CI_Res gmd:organi gco:Charac				
securityco	pycsw:Secu	apiso:HasS	gmd:identi gmd:MD_Dat gmd:resour gmd:MD_Sec				
parentiden	pycsw:Pare	apiso:Pare	gmd:parent gco:Charac		collection	collection	
topicatego	pycsw:Topi	apiso:Topi	gmd:identi gmd:MD_Dat gmd:topicC gmd:MD_Top				
resourcela	pycsw:Reso	apiso:Reso	md:identif gmd:MD_Dat gmd:citati gmd:CI_Cit gmd:identi gmd:code/ gmd:MD_Lan				
geodescode	pycsw:Geog	apiso:Geog	gmd:identi gmd:MD_Dat gmd:extent gmd:EX_Ext gmd:geogra gmd:EX_Geo gmd:geogra gmd:MD_Ide gmd:code/ gco:Charac				

continues on next page

Table 1 – continued from previous page

Database column	pycsw mapping name	Queryable name	ISO 19115 (XPath)	CSW Record/Dub Core (XPath)	OGC API - Records (JSON-Path)	STAC (JSON-Path)	Description
denominato	pycsw:Deno	apiso:Deno	gmd:identi gmd:MD_Dat gmd:spatia gmd:MD_Res gmd:equiva gmd:MD_Rep gmd:denomi gco:Intege				
distanceva	pycsw:Dist	apiso:Dist	gmd:identi gmd:MD_Dat gmd:spatia gmd:MD_Res gmd:distan gco:Distan			gsd or properties gsd	
distanceuo	pycsw:Dist	apiso:Dist	gmd:identi gmd:MD_Dat gmd:spatia gmd:MD_Res gmd:distan gco:Distan @uom			fixed to m	
time_begin	pycsw:Temp	apiso:Temp	gmd:identi gmd:MD_Dat gmd:extent gmd:EX_Ext gmd:tempor gmd:EX_Tem gmd:extent gml:TimePe gml:beginP		properties extent. temporal. interval[0	properties start_date	
time_end	pycsw:Temp	apiso:Temp	gmd:identi gmd:MD_Dat gmd:extent gmd:EX_Ext gmd:tempor gmd:EX_Tem gmd:extent gml:TimePe gml:endPos		properties extent. temporal. interval[1	properties end_dateti	
servicetyp	pycsw:Serv	apiso:Serv	gmd:identi srv:SV_Ser srv:servic gco:LocalN				
servicetyp	pycsw:Serv	apiso:Serv	gmd:identi srv:SV_Ser srv:servic gco:Charac				

continues on next page

Table 1 – continued from previous page

Database column	pycsw mapping name	Queryable name	ISO 19115 (XPath)	CSW Record/Dub Core (XPath)	OGC API - Records (JSON-Path)	STAC (JSON-Path)	Description
operation	pycsw:Oper	apiso:Oper	gmd:identi srv:SV_Ser srv:contai srv:SV_Ope srv:operat gco:Charac				
couplingty	pycsw:Coup	apiso:Coup	gmd:identi srv:SV_Ser srv:coupli srv:SV_Cou				
operateson	pycsw:Oper	apiso:Oper	gmd:identi srv:SV_Ser srv:operat gmd:MD_Dat gmd:citati gmd:CI_Cit gmd:identi gmd:RS_Ide gmd:code/ gco:Charac				
operateson	pycsw:Oper	apiso:Oper	gmd:identi srv:SV_Ser srv:couple srv:SV_Cou srv:identi gco:Charac				
operateson	pycsw:Oper	apiso:Oper	gmd:identi srv:SV_Ser srv:couple srv:SV_Cou srv:operat gco:Charac				
degree	pycsw:Degr	apiso:Degr	gmd:dataQu gmd:DQ_Dat gmd:report gmd:DQ_Dom gmd:result gmd:DQ_Con gmd:pass/ gco:Boolea				
accesscons	pycsw:Acce	apiso:Acce	gmd:identi gmd:MD_Dat gmd:resour gmd:MD_Leg gmd:access gmd:MD_Res	dc:rights			

continues on next page

Table 1 – continued from previous page

Database column	pycsw mapping name	Queryable name	ISO 19115 (XPath)	CSW Record/Dub Core (XPath)	OGC API - Records (JSON-Path)	STAC (JSON-Path)	Description
otherconst	pycsw:Othe	apiso:Othe	gmd:identi gmd:MD_Dat gmd:resour gmd:MD_Leg gmd:otherC gco:Charac		properties license		
classifica	pycsw:Clas	apiso:Clas	gmd:identi gmd:MD_Dat gmd:resour gmd:MD_Leg gmd:access gmd:MD_Cla				
conditiona	pycsw:Cond	apiso:Cond	gmd:identi gmd:MD_Dat gmd:useLim gco:Charac				
lineage	pycsw:Line	apiso:Line	gmd:dataQu gmd:DQ_Dat gmd:lineag gmd:LI_Lin gmd:statem gco:Charac				
responsibl	pycsw:Resp	apiso:Resp	gmd:contac gmd:CI_Res gmd:role/ gmd:CI_Rol				
specificat	pycsw:Spec	apiso:Spec	gmd:dataQu gmd:DQ_Dat gmd:report gmd:DQ_Dom gmd:result gmd:DQ_Con gmd:specif gmd:CI_Cit gmd:title/ gco:Charac				

continues on next page

Table 1 – continued from previous page

Database column	pycsw mapping name	Queryable name	ISO 19115 (XPath)	CSW Record/Dub Core (XPath)	OGC API - Records (JSON-Path)	STAC (JSON-Path)	Description
specificat	pycsw:Spec	apiso:Spec	gmd:dataQu gmd:DQ_Dat gmd:report gmd:DQ_Dom gmd:result gmd:DQ_Con gmd:specif gmd:CI_Cit gmd:date/ gmd:CI_Dat gmd:date/ gco:Date				
specificat	pycsw:Spec	apiso:Spec	gmd:dataQu gmd:DQ_Dat gmd:report gmd:DQ_Dom gmd:result gmd:DQ_Con gmd:specif gmd:CI_Cit gmd:date/ gmd:CI_Dat gmd:dateTy gmd:CI_Dat				
creator	pycsw:Crea	apiso:Crea	gmd:identi gmd:MD_Dat gmd:pointO gmd:CI_Res gmd:organi gmd:CI_Rol @codeListV gco:Charac	dc:creator	properties providers		
publisher	pycsw:Publ	apiso:Publ	gmd:identi gmd:MD_Dat gmd:pointO gmd:CI_Res gmd:organi gmd:CI_Rol @codeListV gco:Charac	dc:publish	properties providers		

continues on next page

Table 1 – continued from previous page

Database column	pycsw mapping name	Queryable name	ISO 19115 (XPath)	CSW Record/DublinCore (XPath)	OGC API - Records (JSON-Path)	STAC (JSON-Path)	Description
contributo	pycsw:Cont	apiso:Cont	gmd:identi gmd:MD_Dat gmd:pointO gmd:CI_Res gmd:organi gmd:CI_Rol @codeListV gco:Charac	dc:contrib	properties providers		
relation	pycsw:Rela	apiso:Rela	gmd:identi gmd:MD_Dat gmd:aggreg	dc:relatio			
platform	pycsw:Plat	apiso:Plat	gmi:acquis gmi:MI_Acq gmi:platfo gmi:MI_Pla gmi:identi			platform or properties platform	
instrument	pycsw:Inst	apiso:Inst	gmi:acquis gmi:MI_Acq gmi:platfo gmi:MI_Pla gmi:instru gmi:MI_Ins gmi:identi			instrument or properties instrument	
sensortype	pycsw:Sens	apiso:Sens	gmi:acquis gmi:MI_Acq gmi:platfo gmi:MI_Pla gmi:instru gmi:MI_Ins gmi:type				
cloudcover	pycsw:Clou	apiso:Clou	gmd:conten gmd:MD_Ima gmd:cloudC				
bands	pycsw:Band	apiso:Band	gmd:conten gmd:MD_Ima gmd:dimens MD_Band/ @id				JSON list of band information, see also *Existing Repository Requirements*

continues on next page

Table 1 – continued from previous page

Database column	pycsw mapping name	Queryable name	ISO 19115 (XPath)	CSW Record/Dub Core (XPath)	OGC API - Records (JSON-Path)	STAC (JSON-Path)	Description
links	pycsw:Link				links	links, assets	List of dicts with properties: name, description, protocol, url, see also *Existing Repository Requirements*
contacts	pycsw:Cont		//gmd:CI_Res		properties providers		List of dicts with properties: name, organization, address, postcode, city, region, country, email, phone, fax, onlineresource, position, role

CHAPTER

ELEVEN

OGC API - RECORDS SUPPORT

11.1 Versions

pycsw supports OGC API - Records - Part 1: Core, version 1.0 by default.

11.2 Request Examples

As the OGC successor to CSW, OGC API - Records is a change in paradigm rooted in lowering the barrier to entry, being webby/of the web, and focusing on developer experience/adoption. JSON and HTML output formats are both supported via the `f` parameter.

```
# landing page
http://localhost:8000/
# landing page explictly as JSON
http://localhost:8000/?f=json
# landing page as HTML
http://localhost:8000/?f=html
# conformance classes
http://localhost:8000/conformance
# all collections
http://localhost:8000/collections
# default collection
http://localhost:8000/collections/metadata:main
# default collection queryables
http://localhost:8000/collections/metadata:main/queryables

# collection queries
# query parameters can be combined (exclusive/AND)

# collection query, all records
http://localhost:8000/collections/metadata:main/items
# collection query, full text search
http://localhost:8000/collections/metadata:main/items?q=lorem
# collection query, full text search (multiple terms result in an exclusive (AND) search
http://localhost:8000/collections/metadata:main/items?q=lorem dolor
# collection query, spatial query
http://localhost:8000/collections/metadata:main/items?bbox=-142,42,-52,84
# collection query, temporal query
http://localhost:8000/collections/metadata:main/items?datetime=2001-10-30/2007-10-30
# collection query, temporal query, before
```

(continues on next page)

(continued from previous page)

```
http://localhost:8000/collections/metadata:main/items?datetime=../2007-10-30
# collection query, temporal query, after
http://localhost:8000/collections/metadata:main/items?datetime=2007-10-30/..
# collection query, property query
http://localhost:8000/collections/metadata:main/items?title=Lorem%20ipsum
# collection query, CQL filter
http://localhost:8000/collections/metadata:main/items?filter-lang=cql-text&filter=title␣
↪LIKE '%lorem%'
# collection query, limiting results
http://localhost:8000/collections/metadata:main/items?limit=1
# collection query, paging
http://localhost:8000/collections/metadata:main/items?limit=10&offset=10
# collection query, paging and sorting (default ascending)
http://localhost:8000/collections/metadata:main/items?limit=10&offset=10&sortby=title
# collection query, paging and sorting (descending)
http://localhost:8000/collections/metadata:main/items?limit=10&offset=10&sortby=-title
# collection query as CQL JSON (HTTP POST), as curl request
curl http://localhost:8000/collections/metadata:main/items --request POST -H "Content-
↪Type: application/json" --data '{ "eq": [{ "property": "title" }, "Lorem ipsum"]}'
# collection query as CQL JSON (HTTP POST), limiting results, as curl request
curl http://localhost:8000/collections/metadata:main/items?limit=0 --request POST -H
↪"Content-Type: application/json" --data '{ "eq": [{ "property": "title" }, "Lorem ipsum
↪"]}'

# collection item as GeoJSON
http://localhost:8000/collections/metadata:main/items/{itemId}
# collection item as HTML
http://localhost:8000/collections/metadata:main/items/{itemId}?f=html
# collection item as XML
http://localhost:8000/collections/metadata:main/items/{itemId}?f=xml
```

11.3 Virtual Collections

In OGC API - Records, pycsw's global repository is named *metadata:main*, which serves all metadata records from a given pycsw configuration.

OGC API - Records support exposes parent metadata as distinct collections, reducing the barrier for users querying on a specific collection, for multiple collections. This functionality is implemented by default and does not require additional setup/configuration by the user. More information on this feature can be found in RFC 10: OGC API - Records virtual collections support.

CHAPTER

TWELVE

CSW SUPPORT

12.1 Versions

pycsw supports both CSW 2.0.2 and 3.0.0 versions by default. In alignment with the CSW specifications, the default version returned is the latest supported version. That is, pycsw will always behave like a 3.0.0 CSW unless the client explicitly requests a 2.0.2 CSW.

The sample URLs below provide examples of how requests behaves against various/missing/default version parameters.

```
http://localhost/csw  # returns 3.0.0 Capabilities
http://localhost/csw?service=CSW&request=GetCapabilities  # returns 3.0.0 Capabilities
http://localhost/csw?service=CSW&version=2.0.2&request=GetCapabilities  # returns 2.0.2␣
↪Capabilities
http://localhost/csw?service=CSW&version=3.0.0&request=GetCapabilities  # returns 3.0.0␣
↪Capabilities
```

12.2 Request Examples

The best place to look for sample requests is within the *tests/* directory, which provides numerous examples of all supported APIs and requests.

Additional examples:

- Data.gov CSW HowTo v2.0
- pycsw Quickstart on OSGeoLive

CHAPTER

THIRTEEN

PUBLISH-SUBSCRIBE INTEGRATION (PUB/SUB)

pycsw supports Publish-Subscribe (Pub/Sub) integration by implementing the OGC API Publish-Subscribe Workflow - Part 1: Core (draft) specification.

Pub/Sub integration can be enabled by defining a broker that pycsw can use to publish notifications on given topics using CloudEvents (as per the specification).

When enabled, core functionality of Pub/Sub includes:

- displaying the broker link in the OGC API - Records landing (using the `rel=hub` link relation)
- sending a notification message on metadata transactions (create, replace, update, delete)

The following message queuing protocols are supported:

13.1 MQTT

Example directive:

```
pubsub:
    broker:
        type: mqtt
        url: mqtt://localhost:1883
        channel: messages/a/data  # optional
        show_link: false  # default true
```

13.2 HTTP

Example directive:

```
pubsub:
    broker:
        type: http
        url: https://ntfy.sh
        channel: messages-a-data  # optional
        show_link: true  # default true
```

Note

For any Pub/Sub endpoints requiring authentication, encode the `url` value as follows:

- `mqtt://username:password@localhost:1883`

- `https://username:password@localhost`

As with any section of the pycsw configuration, environment variables may be used as needed, for example to set username/password information in a URL. If `pubsub.broker.url` contains authentication, and `pubsub.broker.show_link` is `true`, the authentication information will be stripped from the URL before displaying it on the landing page.

Note

If a `channel` is defined, it is used as a prefix to the relevant OGC API endpoint is used.

If a `channel` is not defined, only the relevant OGC API endpoint is used.

CHAPTER

FOURTEEN

SPATIOTEMPORAL ASSET CATALOG (STAC) API SUPPORT

14.1 Versions

pycsw supports SpatioTemporal Asset Catalog API version v1.0.0 by default.

pycsw implements provides STAC support in the following manner:

- a pycsw repository is equivalent to a STAC collection
- pycsw metadata records are equivalent to STAC items

The STAC specification is designed with the same principles as OGC API - Records.

14.2 Implementation

The following design patterns are put forth for STAC support:

14.2.1 Collections

- any pycsw record that is ingested as a STAC Collection will appear on `/stac/collections` as a collection

14.2.2 Search

- In addition to OGC API - Records `/collections/metadata:main/items` search, STAC API specific searches are realized via `/stac/search` which conforms to the STAC API items-search conformance class.

14.2.3 Filtering

STAC API filtering is realized by OGC Common Query Language (CQL2) via HTTP GET and POST. For GET requests, the *filter*= query parameter can be used. For POST requests, a JSON payload with a *filter* property expressing the CQL2 can be used.

14.2.4 Links and Assets

STAC support will render links as follows:

- links that are enclosures will be encoded as STAC assets (in `assets`)
- all other links remain as record links (in `links`)

14.2.5 Transactions

STAC Transactions are supported as per the following STAC API specifications:

- STAC API - Transaction Extension Specification.
- STAC API - Collection Transaction Extension.

14.3 Request Examples

```
# landing page
http://localhost:8000/stac

# collections
http://localhost:8000/stac/collections

# collection queries
# query parameters can be combined (exclusive/AND)

# landing page
http://localhost:8000/stac
# OpenAPI
http://localhost:8000/stac/openapi
# collections
http://localhost:8000/stac/collections
# collections query, full text search
http://localhost:8000/stac/collections?q=sentinel
# collections query, spatial query
http://localhost:8000/stac/collections?bbox=-142,42,-52,84
# collections query, full text search and spatial query
http://localhost:8000/stac/collections?q=sentinel,bbox=-142,42,-52,84
# collections query, limiting results
http://localhost:8000/stac/collections?limit=2
# collections query, spatial query
# single collection
http://localhost:8000/stac/collections/metadata:main
# collection queryables, all records
http://localhost:8000/stac/queryables
# collection query, all records
http://localhost:8000/stac/search
# collection query, full text search
http://localhost:8000/stac/search?q=lorem
# collection query, spatial query
http://localhost:8000/stac/search?bbox=-142,42,-52,84
# collection query, temporal query
http://localhost:8000/stac/search?datetime=2001-10-30/2007-10-30
# collection query, temporal query, before
http://localhost:8000/stac/search?datetime=../2007-10-30
# collection query, temporal query, after
http://localhost:8000/stac/search?datetime=2007-10-30/..
# collection query, property query
http://localhost:8000/stac/search?title=Lorem%20ipsum
# collection query, CQL filter
http://localhost:8000/stac/search?filter=title like "%lorem%"
```

(continues on next page)

(continued from previous page)

```
# collection query, limiting results
http://localhost:8000/stac/search?limit=1
# collection filter query, limiting results
http://localhost:8000/stac/search?limit=1&collections=landsat
# collection ids filter query, limiting results
http://localhost:8000/stac/search?limit=1&ids=id1,id2
# collection query, paging
http://localhost:8000/stac/search?limit=10&offset=10
# collection query, paging and sorting (default ascending)
http://localhost:8000/stac/search?limit=10&offset=10&sortby=title
# collection query, paging and sorting (descending)
http://localhost:8000/stac/search?limit=10&offset=10&sortby=-title
# collection item as GeoJSON
http://localhost:8000/stac/collections/metadata:main/items/{itemId}

# CQL2 JSON (as curl commands)

# search by creation date
curl --location --request POST 'http://localhost:8000/stac/search' \
    --header 'Content-Type: application/query-cql-json' \
    --data-raw '{
        "filter-lang": "cql2-json",
        "filter": {
            "op": "<=",
            "args": [
                {
                    "property": "date_creation"
                },
                "2025-12-15"
            ]
        }
    }'

# search by creation date
curl --location --request POST 'http://localhost:8000/stac/search' \
    --header 'Content-Type: application/query-cql-json' \
    --data-raw '{
      "filter-lang": "cql2-json",
      "filter": {
          "op": "and",
          "args": [
              {
                  "op": "in",
                  "args": [
                      {
                          "property": "parentidentifier"
                      },
                      [
                          "sentinel-2-l2a"
                      ]
                  ]
              }
```

(continues on next page)

(continued from previous page)

```
            ]
        }
    }'
```

CHAPTER

FIFTEEN

DISTRIBUTED SEARCHING

Note

- in CSW mode, distributed search must be configured against remote CSW services
- in OGC API - Records mode, distributed search must be configured against remote OGC API - Records services

Note

Your server must be able to make outgoing HTTP requests for this functionality.

15.1 CSW 2 / 3

pycsw has the ability to perform distributed searching against other CSW servers. Distributed searching is disabled by default; to enable, `distributedsearch` must be set. A CSW client must issue a GetRecords request with `csw:DistributedSearch` specified, along with an optional `hopCount` attribute (see subclause 10.8.4.13 of the CSW specification). When enabled, pycsw will search all specified catalogues and return a unified set of search results to the client. Due to the distributed nature of this functionality, requests will take extra time to process compared to queries against the local repository.

15.1.1 Scenario: Federated Search

pycsw deployment with 3 configurations (CSW-1, CSW-2, CSW-3), subsequently providing three (3) endpoints. Each endpoint is based on an opaque metadata repository (based on theme/place/discipline, etc.). Goal is to perform a single search against all endpoints.

pycsw realizes this functionality by supporting *alternate configurations*, and exposes the additional CSW endpoint(s) with the following design pattern:

CSW-1: `http://localhost/pycsw/csw.py?config=CSW-1.yml`

CSW-2: `http://localhost/pycsw/csw.py?config=CSW-2.yml`

CSW-3: `http://localhost/pycsw/csw.py?config=CSW-3.yml`

...where the `*.yml` configuration files are configured for each respective metadata repository. The above CSW endpoints can be interacted with as usual.

To federate the discovery of the three (3) portals into a unified search, pycsw realizes this functionality by deploying an additional configuration which acts as the superset of CSW-1, CSW-2, CSW-3:

CSW-all: `http://localhost/pycsw/csw.py?config=CSW-all.yml`

This allows the client to invoke one (1) CSW GetRecords request, in which the CSW endpoint spawns the same GetRecords request to 1..n distributed CSW endpoints. Distributed CSW endpoints are advertised in CSW Capabilities XML via `ows:Constraint`:

```
<ows:OperationsMetadata>
...
    <ows:Constraint name="FederatedCatalogues">
        <ows:Value>http://localhost/pycsw/csw.py?config=CSW-1.yml</ows:Value>
        <ows:Value>http://localhost/pycsw/csw.py?config=CSW-2.yml</ows:Value>
        <ows:Value>http://localhost/pycsw/csw.py?config=CSW-3.yml</ows:Value>
    </ows:Constraint>
...
</ows:OperationsMetadata>
```

...which advertises which CSW endpoint(s) the CSW server will spawn if a distributed search is requested by the client.

in the CSW-all configuration:

```
distributedsearch:
    merge_results: false
    catalogues:
        - id: fedcat01
          type: CSW
          title: Federated catalogue 1
          url: http://localhost/pycsw/csw.py?config=CSW-1.yml
        - id: fedcat02
          type: CSW
          title: Federated catalogue 2
          url: http://localhost/pycsw/csw.py?config=CSW-2.yml
        - id: fedcat03
          type: CSW
          title: Federated catalogue 3
          url: http://localhost/pycsw/csw.py?config=CSW-3.yml
```

At which point a CSW client request to CSW-all with `distributedsearch=TRUE`, while specifying an optional `hopCount`. Query network topology:

```
     AnyClient
         ^
         |
         v
      CSW-all
         ^
         |
         v
  /-------------\
  ^       ^     ^
  |       |     |
  v       v     v
CSW-1   CSW-2  CSW-3
```

As a result, a pycsw deployment in this scenario may be approached on a per 'theme' basis, or at an aggregate level.

All interaction in this scenario is local to the pycsw installation, so network performance would not be problematic.

A very important facet of distributed search is as per Annex B of OGC:CSW 2.0.2. Given that all the CSW endpoints

are managed locally, duplicates and infinite looping are not deemed to present an issue.

15.2 OGC API - Records

Experimental support for distibuted searching is available in pycsw's OGC API - Records support to allow for searching remote services. The implementation uses the same approach as described above, operating in OGC API - Records mode as per OGC API - Records - Part 4: Federated Search (draft).

> **Note**
>
> The `distributedsearch.catalogues` directives must point to an OGC API - Records **collections** endpoint.

```
distributedsearch
    catalogues:
        - id: fedcat01
          type: OARec
          title: Federated catalogue 1
          url: https://example.org/collections/collection1
        - id: fedcat02
          type: OARec
          title: Federated catalogue 2
          url: https://example.org/collections/collection2
```

With the above configured, a distributed search can be invoked as follows:

http://localhost/collections/metadata:main/items?distributedSearch=true

15.2.1 Merging results

When *distributedsearch.merge_results* exists and is set to `true`, pycsw will merge all results in `federatedSearchResults`. To prevent identifier collision, merged federated search results will have identifiers prefixed by their catalogue `id` (as defined in `distributedsearch.catalogues[*].id`. In addition, a `federatedCatalogueId` property is added to the feature with the catalogue id.

15.3 STAC API

Experimental support for distibuted searching is available in pycsw's STAC API support to allow for searching remote services. The implementation uses the same approach as described above.

> **Note**
>
> The `distributedsearch.catalogues` directives must point to a STAC API endpoint.

```
distributedsearch:
    catalogues:
        - id: fedcat03
          type: STAC-API
          title: Copernicus Data Space Ecosystem (CDSE) asset-level STAC catalogue
          url: https://stac.dataspace.copernicus.eu/v1
```

(continues on next page)

(continued from previous page)

```
        collections:
            - daymet-annual-pr
```

Note

To constrain STAC API distributed search to specific collections, define one to many in the *collections* (array) directive.

With the above configured, a distributed search can be invoked as follows:

http://localhost/stac/search?distributedSearch=true

15.3.1 Merging results

Merging behaviour is implemented in the same manner as OGC API - Records support.

CHAPTER

SIXTEEN

SEARCH/RETRIEVAL VIA URL (SRU) SUPPORT

pycsw supports the Search/Retrieval via URL search protocol implementation as per subclause 8.4 of the OpenGIS Catalogue Service Implementation Specification.

SRU support is enabled by default. There are two ways to access SRU depending on the deployment pattern chosen.

16.1 OGC API - Records deployment

```
http://localhost:8000/sru
```

16.2 CSW legacy deployment

HTTP GET requests must be specified with `mode=sru` for SRU requests, e.g.:

```
http://localhost/pycsw/csw.py?mode=sru&operation=searchRetrieve&query=foo
```

See https://www.loc.gov/standards/sru/misc/simple.html for example SRU requests.

CHAPTER

SEVENTEEN

OPENSEARCH SUPPORT

pycsw OpenSearch support is enabled by default. There are two ways to access OpenSearch depending on the deployment pattern chosen.

17.1 OGC API - Records deployment

```
http://localhost:8000/opensearch
```

17.2 CSW legacy deployment

HTTP requests must be specified with `mode=opensearch` in the base URL for OpenSearch requests, e.g.:

```
http://localhost/pycsw/csw.py?mode=opensearch&service=CSW&version=2.0.2&
↪request=GetCapabilities
```

This will return the Description document which can then be autodiscovered.

17.3 OGC OpenSearch Geo and Time Extensions 1.0

pycsw supports the OGC OpenSearch Geo and Time Extensions 1.0 standard via the following conformance classes:

- Core (GeoSpatial Service) `{searchTerms}`, `{geo:box}`, `{startIndex}`, `{count}`
- Temporal Search core `{time:start}`, `{time:end}`

17.3.1 Supported Query Parameters

- `q`
- `time`
- `bbox`

17.4 OGC OpenSearch Extension for Earth Observation

pycsw supports the OGC OpenSearch Extension for Earth Observation standard via the following conformance classes:

- Core

17.4.1 Supported Query Parameters

- eo:cloudCover
- eo:instrument
- eo:orbitDirection
- eo:orbitNumber
- eo:platform
- eo:processingLevel
- eo:productType
- eo:sensorType
- eo:snowCover
- eo:spectralRange
- eo:illuminationElevationAngle

17.4.2 Mapping of non-19115 Queryable Mappings

The following queryables are implemented as faceted keywords given they are not implemented in generic geospatial metadata standards:

- eo:productType
- eo:orbitNumber
- eo:orbitDirection
- eo:snowCover
- eo:processingLevel

This means metadata ingested into pycsw must have these fields implemented as keywords, as per the examples below:

```
<!-- ISO 19115 -->
<gmd:keyword>
  <gco:CharacterString>eo:productType:S2MSI2A</gco:CharacterString>
</gmd:keyword>
<gmd:keyword>
  <gco:CharacterString>eo:orbitNumber:50</gco:CharacterString>
</gmd:keyword>
<gmd:keyword>
  <gco:CharacterString>eo:orbitDirection:DESCENDING</gco:CharacterString>
</gmd:keyword>
<gmd:keyword>
  <gco:CharacterString>eo:snowCover:0.0</gco:CharacterString>
</gmd:keyword>
<gmd:keyword>
  <gco:CharacterString>eo:procesingLevel:Level-2A</gco:CharacterString>
</gmd:keyword>
```

```
<!-- Dublin Core -->
<dc:subject>eo:productType:S2MSI2A</dc:subject>
<dc:subject>eo:orbitNumber:50</dc:subject>
<dc:subject>eo:orbitDirection:DESCENDING</dc:subject>
```

(continues on next page)

(continued from previous page)

```
<dc:subject>eo:snowCover:0.0</dc:subject>
<dc:subject>eo:procesingLevel:Level-2A</dc:subject>
```

17.5 OpenSearch Temporal Queries Implementation

By default, pycsw's OpenSearch temporal support will query the Dublin Core `dc:date` property as a time instant/single point in time. To enable temporal extent search, set `profiles=apiso` which will query the temporal extents of a metadata record (`apiso:TempExtent_begin` and `apiso:TempExtent_end`).

At the HTTP API level, time is supported via either `time=t1/t2` or `start=t1&stop=t2`. If the `time` parameter is present, it will override the `start` and `stop` parameters respectively.

CHAPTER

EIGHTEEN

OAI-PMH SUPPORT

pycsw supports the The Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) standard.

OAI-PMH OpenSearch support is enabled by default. There are two ways to access OAI-PMH depending on the deployment pattern chosen.

18.1 OGC API - Records deployment

```
http://localhost:8000/oaipmh
```

18.2 CSW legacy deployment

HTTP requests must be specified with `mode=oaipmh` in the base URL for OAI-PMH requests, e.g.:

```
http://localhost/pycsw/csw.py?mode=oaipmh&verb=Identify
```

See http://www.openarchives.org/OAI/openarchivesprotocol.html for more information on OAI-PMH as well as request / reponse examples.

CHAPTER

NINETEEN

JSON SUPPORT

19.1 OGC API - Records

pycsw fully supports the OGC API - Records JSON conformance class, which is the default representation provided.

19.2 CSW

pycsw supports JSON support for `DescribeRecord`, `GetRecords` and `GetRecordById` requests. Adding `outputFormat=application/json` to your CSW request will return the response as a JSON representation.

CHAPTER

TWENTY

SOAP

pycsw's CSW implementation supports handling of SOAP encoded requests and responses as per subclause 10.3.2 of OGC:CSW 2.0.2. SOAP request examples can be found in `tests/index.html`.

CHAPTER

TWENTYONE

XML SITEMAPS

XML Sitemaps can be generated by running:

```
pycsw-admin.py gen-sitemap --config default.yml --output sitemap.xml
```

The `sitemap.xml` file should be saved to an an area on your web server (parallel to or above your pycsw install location) to enable web crawlers to index your repository.

CHAPTER

TWENTYTWO

TRANSACTIONS USING CSW

pycsw's CSW implementation has the ability to process CSW Harvest and Transaction requests (CSW-T). Transactions are disabled by default; to enable, `manager.transactions` must be set to `true`. Access to transactional functionality is limited to IP addresses which must be set in `manager.allowed_ips`.

22.1 Supported Resource Types

For transactions and harvesting, pycsw supports the following metadata resource types by default:

Resource Type	Namespace	Transaction	Harvest
Dublin Core	`http://www.opengis.net/cat/csw/2.0.2`	yes	yes
FGDC	`http://www.opengis.net/cat/csw/csdgm`	yes	yes
GM03	`http://www.interlis.ch/INTERLIS2.3`	yes	yes
ISO 19139	`http://www.isotc211.org/2005/gmd`	yes	yes
ISO GMI	`http://www.isotc211.org/2005/gmi`	yes	yes
OGC:CSW 2.0.2	`http://www.opengis.net/cat/csw/2.0.2`		yes
OGC:WMS 1.1.1/1.3.0	`http://www.opengis.net/wms`		yes
OGC:WMTS 1.0.0	`http://www.opengis.net/wmts/1.0`		yes
OGC:WFS 1.0.0/1.1.0/2.0.0	`http://www.opengis.net/wfs`		yes
OGC:WCS 1.0.0	`http://www.opengis.net/wcs`		yes
OGC:WPS 1.0.0	`http://www.opengis.net/wps/1.0.0`		yes
OGC:SOS 1.0.0	`http://www.opengis.net/sos/1.0`		yes
OGC:SOS 2.0.0	`http://www.opengis.net/sos/2.0`		yes
WAF	`urn:geoss:waf`		yes

Additional metadata models are supported by enabling the appropriate *Profile Plugins*.

Note

For transactions to be functional when using SQLite3, the SQLite3 database file (**and its parent directory**) must be fully writable. For example:

```
$ mkdir /path/data
$ chmod 777 /path/data
$ chmod 666 test.db
$ mv test.db /path/data
```

For CSW-T deployments, it is strongly advised that this directory reside in an area that is not accessible by HTTP.

22.2 Harvesting

> **Note**
>
> Your server must be able to make outgoing HTTP requests for this functionality.

pycsw supports the CSW-T `Harvest` operation. Records which are harvested require to setup a cronjob to periodically refresh records in the local repository. A sample cronjob is available in `etc/harvest-all.cron` which points to `pycsw-admin.py` (you must specify the correct path to your configuration). Harvest operation results can be sent by email (via `mailto:`) or ftp (via `ftp://`) or ftps (via `ftps://`) if the Harvest request specifies `csw:ResponseHandler`.

> **Note**
>
> For `csw:ResponseHandler` values using the `mailto:` protocol, you must have `server.smtp_host` set in your *configuration*.

22.2.1 OGC Web Services

When harvesting OGC web services, requests can provide the base URL of the service as part of the Harvest request. pycsw will construct a `GetCapabilities` request dynamically.

When harvesting other CSW servers, pycsw pages through the entire CSW in default increments of 10. This value can be modified via the `manager.csw_harvest_pagesize` *configuration* option. It is strongly advised to use the `csw:ResponseHandler` parameter for harvesting large CSW catalogues to prevent HTTP timeouts.

22.3 Transaction operations

pycsw supports 3 modes of the `Transaction` operation (`Insert`, `Update`, `Delete`):

- **Insert**: full XML documents can be inserted as per CSW-T
- **Update**: updates can be made as full record updates or record properties against a `csw:Constraint`
- **Delete**: deletes can be made against a `csw:Constraint`

Transaction operation results can be sent by email (via `mailto:`) or ftp (via `ftp://`) or ftps (via `ftps://`) if the Transaction request specifies `csw:ResponseHandler`.

The *Testing* contain CSW-T request examples.

CHAPTER

TWENTYTHREE

TRANSACTIONS USING OGC API - RECORDS

pycsw's OGC API - Records support provides transactional capabilities via the OGC API - Features - Part 4: Create, Replace, Update and Delete draft specification, which follows RESTful patterns for insert/update/delete of resources.

23.1 Supported Resource Types

All resource types supported by CSW Transactions are supported via OGC API - Records transactional workflow. Note that the HTTP `Content-Type` header MUST be set according to the media type of the given resource (i.e. `application/json`, `application/xml`, etc.).

23.2 Transaction operations

The below examples demonstrate transactional workflow using pycsw's OGC API - Records endpoint:

```
# insert GeoJSON metadata
curl -v -H "Content-Type: application/geo+json" -XPOST http://localhost:8000/collections/
↪metadata:main/items -d @foorecord.json

# update metadata
curl -v -H "Content-Type: application/geo+json" -XPUT http://localhost:8000/collections/
↪metadata:main/items/foorecord -d @foorecord.json

# delete metadata
curl -v -XDELETE http://localhost:8000/collections/metadata:main/items/foorecord

# insert XML metadata
curl -v -H "Content-Type: application/xml" -XPOST http://localhost:8000/collections/
↪metadata:main/items -d @foorecord.xml
```

23.3 Harvesting

Harvesting is not yet supported via OGC API - Records.

CHAPTER

TWENTYFOUR

TRANSACTIONS USING STAC API

pycsw's STAC API support provides transactional capabilities via the STAC API - Transaction Extension Specification and STAC API - Collection Transaction Extension specifications, which follows RESTful patterns for insert/update/delete of resources.

24.1 Supported Resource Types

STAC Collections, Items and Item Collections are supported via OGC API - Records transactional workflow. Note that the HTTP `Content-Type` header MUST be set to (i.e. `application/json`).

24.2 Transaction operations

The below examples demonstrate transactional workflow using pycsw's OGC API - Records endpoint:

```
# insert STAC Item
curl -v -H "Content-Type: application/json" -XPOST http://localhost:8000/stac/
↪collections/metadata:main/items -d @fooitem.json

# update STAC Item
curl -v -H "Content-Type: application/json" -XPUT http://localhost:8000/stac/collections/
↪metadata:main/items/fooitem -d @fooitem.json

# delete STAC Item
curl -v -XDELETE http://localhost:8000/stac/collections/metadata:main/items/fooitem

# insert STAC Item Collection
curl -v -H "Content-Type: application/json" -XPOST http://localhost:8000/stac/
↪collections/metadata:main/items -d @fooitemcollection.json

# insert STAC Collection
curl -v -H "Content-Type: application/json" -XPOST http://localhost:8000/stac/
↪collections -d @foocollection.json

# update STAC Collection
curl -v -H "Content-Type: application/json" -XPUT http://localhost:8000/stac/collections/
↪foocollection -d @foocollection.json

# delete STAC Collection
curl -v -XDELETE http://localhost:8000/stac/collections/foocollection
```

CHAPTER

TWENTYFIVE

REPOSITORY FILTERS

pycsw has the ability to perform server side repository / database filters as a means to mask all requests to query against a specific subset of the metadata repository, thus providing the ability to deploy multiple pycsw instances pointing to the same database in different ways via the `repository.filter` configuration option.

Repository filters are a convenient way to subset your repository at the server level without the hassle of creating proper database views. For large repositories, it may be better to subset at the database level for performance.

25.1 Scenario: One Database, Many Views

Imagine a sample database table of records (subset below for brevity):

identifier	parentidentifier	title	abstract
1	33	foo1	bar1
2	33	foo2	bar2
3	55	foo3	bar3
4	55	foo1	bar1
5	21	foo5	bar5
5	21	foo6	bar6

A default pycsw instance (with no `repository.filters` option) will always process requests against the entire table. So a CSW *GetRecords* filter like:

```
<ogc:Filter>
    <ogc:PropertyIsEqualTo>
        <ogc:PropertyName>apiso:Title</ogc:PropertyName>
        <ogc:Literal>foo1</ogc:Literal>
    </ogc:PropertyIsEqualTo>
</ogc:Filter>
```

. . . will return:

identifier	parentidentifier	title	abstract
1	33	foo1	bar1
4	55	foo1	bar1

Suppose you wanted to deploy another pycsw instance which serves metadata from the same database, but only from a specific subset. Here we set the `repository.filter` option:

```
repository:
    database: sqlite:///records.db
    filter: pycsw:ParentIdentifier = '33'
```

The same CSW *GetRecords* filter as per above then yields the following results:

identifier	parentidentifier	title	abstract
1	33	foo1	bar1

Another example:

```
repository:0
    database: sqlite:///records.db
    filter: "pycsw:ParentIdentifier != '33'"
```

The same CSW *GetRecords* filter as per above then yields the following results:

identifier	parentidentifier	title	abstract
4	55	foo1	bar1

The `repository.filter` option accepts all core queryables set in the pycsw core model (see `pycsw.config.StaticContext.md_core_model` for the complete list).

CHAPTER

TWENTYSIX

PROFILE PLUGINS

26.1 Overview

pycsw allows for the implementation of profiles to the core standard. Profiles allow specification of additional metadata format types (i.e. ISO 19139:2007, NASA DIF, INSPIRE, etc.) to the repository, which can be queried and presented to the client. pycsw supports a plugin architecture which allows for runtime loading of Python code.

All profiles must be placed in the `pycsw/plugins/profiles` directory.

26.2 Requirements

```
pycsw/
  plugins/
  __init__.py # empty
  profiles/ # directory to store profiles
    __init__.py # empty
    profile.py # defines abstract profile object (properties and methods) and functions␣
↪to load plugins
    apiso/ # profile directory
      __init__.py # empty
      apiso.py # profile code
      ... # supporting files, etc.
```

26.3 Abstract Base Class Definition

All profile code must be instantiated as a subclass of `profile.Profile`. Below is an example to add a `Foo` profile:

```
from pycsw.plugins.profiles import profile

class FooProfile(profile.Profile):
    profile.Profile.__init__(self,
        name='foo',
        version='1.0.3',
        title='My Foo Profile',
        url='http://example.org/fooprofile/docs',
        namespace='http://example.org/foons',
        typename='foo:RootElement',
        outputschema=http://example.org/foons',
        prefixes=['foo'],
```

(continues on next page)

(continued from previous page)

```
        model=model,
        core_namespaces=namespaces,
        added_namespaces={'foo': 'http://example.org/foons'}
        repository=REPOSITORY['foo:RootElement'])
```

Your profile plugin class (`FooProfile`) must implement all methods as per `profile.Profile`. Profile methods must always return `lxml.etree.Element` types, or `None`.

26.4 Enabling Profiles

All profiles are disabled by default. To specify profiles at runtime, set the `profiles` value in the *Configuration* to the name of the package (in the `pycsw/plugins/profiles` directory). To enable multiple profiles, specify as a list (see *Configuration*).

26.5 Testing

Profiles must add examples to the *Testing* interface, which must provide example requests specific to the profile.

CHAPTER

TWENTYSEVEN

SUPPORTED PROFILES

27.1 ISO Metadata Application Profile (1.0.0)

27.1.1 Overview

The ISO Metadata Application Profile (APISO) is a profile of CSW 2.0.2 which enables discovery of geospatial metadata following ISO 19139:2007 and ISO 19119:2005/PDAM 1.

27.1.2 Configuration

No extra configuration is required.

27.1.3 Querying

- **typename**: `gmd:MD_Metadata`
- **outputschema**: `http://www.isotc211.org/2005/gmd`

27.1.4 Enabling APISO Support

To enable APISO support, add `apiso` to `profiles` as specified in *Configuration*.

27.1.5 Testing

A testing interface is available in `tests/index.html` which contains tests specific to APISO to demonstrate functionality. See *Testing* for more information.

27.2 INSPIRE Extension

27.2.1 Overview

APISO includes an extension for enabling INSPIRE Discovery Services 3.0 support. To enable the INSPIRE extension to APISO, create a `[metadata:inspire]` section in the main configuration with `enabled` set to `true`.

27.2.2 Configuration

INSPIRE configuration is specified within `metadata.inspire`:

inspire

- **enabled**: whether to enable the INSPIRE extension (`true` or `false`)
- **languages_supported**: supported languages (see http://inspire.ec.europa.eu/schemas/common/1.0/enums/enum_eng.xsd, simpleType `euLanguageISO6392B`)

- **default_language**: the default language (see http://inspire.ec.europa.eu/schemas/common/1.0/enums/enum_eng.xsd, simpleType `euLanguageISO6392B`)
- **date**: date of INSPIRE metadata offering (in ISO 8601 format)
- **gemet_keywords**: list of GEMET INSPIRE theme keywords about the service (see http://inspire.ec.europa.eu/schemas/common/1.0/enums/enum_eng.xsd, complexType `inspireTheme_eng`)
- **conformity_service**: the level of INSPIRE conformance for spatial data sets and services (`conformant`, `notConformant`, `notEvaluated`)
- **contact_organization**: the organization name responsible for the INSPIRE metadata
- **contact_email**: the email address of entity responsible for the INSPIRE metadata
- **temp_extent**: temporal extent of the service (in ISO 8601 format). Either a single date (i.e. `yyyy-mm-dd`), or an extent (i.e. `yyyy-mm-dd/yyyy-mm-dd`)

27.3 CSW-ebRIM Registry Service - Part 1: ebRIM profile of CSW

27.3.1 Overview

The CSW-ebRIM Registry Service is a profile of CSW 2.0.2 which enables discovery of geospatial metadata following the ebXML information model.

27.3.2 Configuration

No extra configuration is required.

27.3.3 Querying

- **typename**: `rim:RegistryObject`
- **outputschema**: `urn:oasis:names:tc:ebxml-regrep:xsd:rim:3.0`

27.3.4 Enabling ebRIM Support

To enable ebRIM support, add `ebrim` to `profiles` as specified in *Configuration*.

27.3.5 Testing

A testing interface is available in `tests/index.html` which contains tests specific to ebRIM to demonstrate functionality. See *Testing* for more information.

CHAPTER

TWENTYEIGHT

REPOSITORY PLUGINS

28.1 Overview

pycsw allows for the implementation of custom repositories in order to connect to a backend different from the pycsw's default. This is especially useful when downstream applications manage their own metadata model/database/document store and want pycsw to connect to it directly instead of using pycsw's default model, thus creating duplicate repositories which then require syncronization/accounting. Repository plugins enable a single metadata backend which is independent from the pycsw setup. pycsw thereby becomes a pure wrapper around a given backend in providing OGC API - Records, CSW and other APIs atop a given application.

All outputschemas must be placed in the `pycsw/plugins/outputschemas` directory.

28.2 Requirements

Repository plugins:

- can be developed and referenced / connected external to pycsw
- must be accessible within the `PYTHONPATH` of a given application
- must implement pycsw's `pycsw.core.repository.Repository` properties and methods
- must be specified in the pycsw *Configuration* as a class reference (e.g. `path.to.repo_plugin.MyRepository`)
- must minimally implement the `query_insert`, `query_domain`, `query_ids`, and `query` methods

28.3 Configuration

- set pycsw's `repository.source` setting to the class which implements the custom repository:

```
repository:
    mappings: 'path.to.repo_plugin.MyRepository'
```

CHAPTER

TWENTYNINE

OUTPUT SCHEMA PLUGINS

29.1 Overview

pycsw allows for extending the implementation of output schemas to the core standard. outputschemas allow for a client to request metadata in a specific format (ISO, Dublin Core, FGDC, NASA DIF Atom and GM03 are default).

All outputschemas must be placed in the `pycsw/plugins/outputschemas` directory.

29.2 Requirements

```
pycsw/
  plugins/
  __init__.py # empty
  outputschemas/
    __init__.py # __all__ is a list of all provided outputschemas
    atom.py # default
    dif.py # default
    fgdc.py # default
    gm03.py # default
```

29.3 Implementing a new outputschema

Create a file in `pycsw/plugins/outputschemas`, which defines the following:

- `NAMESPACE`: the default namespace of the outputschema which will be advertised
- `NAMESPACE`: dict of all applicable namespaces to outputschema
- `XPATH_MAPPINGS`: dict of pycsw core queryables mapped to the equivalent XPath of the outputschema
- `write_record`: function which returns a record as an `lxml.etree.Element` object

Add the name of the file to `__init__.py:__all__`. The new outputschema is now supported in pycsw.

29.4 Testing

New outputschemas must add examples to the *Testing* interface, which must provide example requests specific to the profile.

CHAPTER

THIRTY

XSLT SUPPORT

By default, pycsw performs metadata transformations using a generic framework that attempts to cover generic use cases. pycsw users can also specify custom XSLT transformations for specific use cases or communities.

To specify a custom XSLT transformation, you must map to input and output outputschemas supported by pycsw, where the input outputschema must match the metadata as ingested and stored in the repository.

```
xslt:
    - input_os: http://www.opengis.net/cat/csw/2.0.2
      output_os: http://www.isotc211.org/2005/gmd
      transform: tests/functionaltests/suites/xslt/custom.xslt
```

The `xslt` directive must point to a valid XSLT document on disk.

Note

You may also use environment variables to point to XSLT files.

CHAPTER

THIRTYONE

HTML TEMPLATING

pycsw uses Jinja as its templating engine to render HTML and Flask to provide route paths of the API that returns HTTP responses. For complete details on how to use these modules, refer to the Jinja documentation and the Flask documentation.

The default pycsw configuration has `server.templates` commented out and defaults to the pycsw `pycsw/templates` and `pycsw/static` folder. To point to a different set of template configuration, you can edit your configuration as follows:

```
server:
  templates:
    path: /path/to/jinja2/templates/folder # jinja2 template HTML files
    static: /path/to/static/folder # css, js, images and other static files referenced␣
↪by the template
```

Note: the URL path to your static folder will always be `/static` in your deployed web instance of pycsw.

Your templates folder should mimic the same file names and structure of the default pycsw templates. Otherwise, you will need to modify `api.py` accordingly.

Note that you need only copy and edit the templates you are interested in updating. For example, if you are only interested in updating the `landing_page.html` template, then create your own version of only that same file. When pycsw detects that a custom HTML template is being used, it will look for the custom template in `server.templates.path`. If it does not exist, pycsw will render the default HTML template for the given endpoint/request.

Linking to a static file in your HTML templates can be done using Jinja syntax and the exposed `config['server']['url']`:

```
<!-- CSS example -->
<link rel="stylesheet" href="{{ config['server']['url'] }}/static/css/default.css">
<!-- JS example -->
<script src="{{ config['server']['url'] }}/static/js/main.js"></script>
<!-- Image example with metadata -->
<img src="{{ config['server']['url'] }}/static/img/logo.png" title="{{ config['metadata
↪']['identification']['title'] }}" />
```

CHAPTER

THIRTYTWO

GEONODE CONFIGURATION

GeoNode (https://geonode.org/) is a platform for the management and publication of geospatial data. It brings together mature and stable open-source software projects under a consistent and easy-to-use interface allowing users, with little training, to quickly and easily share data and create interactive maps. GeoNode provides a cost-effective and scalable tool for developing information management systems. GeoNode uses CSW as a cataloguing mechanism to query and present geospatial metadata.

pycsw supports binding to an existing GeoNode repository for metadata query. The binding is read-only (transactions are not in scope, as GeoNode manages repository metadata changes in the application proper).

32.1 GeoNode Setup

pycsw is enabled and configured by default in GeoNode, so there are no additional steps required once GeoNode is setup. See the `CATALOGUE` and `PYCSW` settings.py entries at for customizing pycsw within GeoNode.

The GeoNode plugin is managed outside of pycsw within the GeoNode project.

CHAPTER

THIRTYTHREE

HHYPERMAP-REGISTRY CONFIGURATION

HHypermap (Harvard Hypermap) Registry (https://github.com/cga-harvard/Hypermap-Registry) is an application that manages OWS, Esri REST, and other types of map service harvesting, and maintains uptime statistics for services and layers. HHypermap Registry will publish to HHypermap Search (based on Lucene) which provides a fast search and visualization environment for spatio-temporal materials.

HHypermap uses CSW as a cataloguing mechanism to ingest, query and present geospatial metadata.

pycsw supports binding to an existing HHypermap repository for metadata query.

33.1 HHypermap Setup

pycsw is enabled and configured by default in HHypermap, so there are no additional steps required once HHypermap is setup. See the `REGISTRY_PYCSW` hypermap/settings.py entries for customizing pycsw within HHypermap.

The HHypermap plugin is managed outside of pycsw within the HHypermap project. HHypermap settings must ensure that `REGISTRY_PYCSW['repository']['source']` is set to `hypermap.search.pycsw_repository`.

CHAPTER

THIRTYFOUR

OPEN DATA CATALOG CONFIGURATION

Open Data Catalog (https://github.com/azavea/Open-Data-Catalog/) is an open data catalog based on Django, Python and PostgreSQL. It was originally developed for OpenDataPhilly.org, a portal that provides access to open data sets, applications, and APIs related to the Philadelphia region. The Open Data Catalog is a generalized version of the original source code with a simple skin. It is intended to display information and links to publicly available data in an easily searchable format. The code also includes options for data owners to submit data for consideration and for registered public users to nominate a type of data they would like to see openly available to the public.

pycsw supports binding to an existing Open Data Catalog repository for metadata query. The binding is read-only (transactions are not in scope, as Open Data Catalog manages repository metadata changes in the application proper).

34.1 Open Data Catalog Setup

Open Data Catalog provides CSW functionality using pycsw out of the box (installing ODC will also install pycsw). Settings are defined in https://github.com/azavea/Open-Data-Catalog/blob/master/OpenDataCatalog/settings.py#L165.

ODC settings must ensure that `REGISTRY_PYCSW['repository']['source']` is set to``hypermap.search.pycsw_repository``.

At this point, pycsw is able to read from the Open Data Catalog repository using the Django ORM.

CHAPTER

THIRTYFIVE

CKAN CONFIGURATION

CKAN (https://ckan.org) is a powerful data management system that makes data accessible – by providing tools to streamline publishing, sharing, finding and using data. CKAN is aimed at data publishers (national and regional governments, companies and organizations) wanting to make their data open and available.

ckanext-spatial is CKAN's geospatial extension. The extension adds a spatial field to the default CKAN dataset schema, using PostGIS as the backend. This allows to perform spatial queries and display the dataset extent on the frontend. It also provides harvesters to import geospatial metadata into CKAN from other sources, as well as commands to support the CSW standard. Finally, it also includes plugins to preview spatial formats such as GeoJSON.

35.1 CKAN Setup

Installation and configuration Instructions are provided as part of the ckanext-spatial documentation.

CHAPTER

THIRTYSIX

API

Python applications can integrate pycsw into their custom workflows. This allows for seamless integate within frameworks such as Flask and Django.

Below are examples of where using the API (as opposed to the default WSGI/CGI services could be used:

- configuration based on a Python dict, or stored in a database
- downstream request environment / framework (Flask, Django)
- authentication or authorization logic
- forcing CSW version 2.0.2 as default

36.1 OGC API - Records Flask Example

See https://github.com/geopython/pycsw/blob/master/pycsw/wsgi_flask.py for how to implement a Flask wrapper atop all pycsw supported APIs. Note the use of Flask blueprints to enable integration with downstream Flask applications.

36.2 Simple Flask blueprint example

```
from flask import Flask, redirect

from pycsw.wsgi_flask import BLUEPRINT as pycsw_blueprint

app = Flask(__name__, static_url_path='/static')

app.url_map.strict_slashes = False
app.register_blueprint(pycsw_blueprint, url_prefix='/oapi')

@app.route('/')
def hello_world():
    return "Hello, World!"
```

In the above example, all pycsw endpoints are made available under `http://localhost:8000/oapi`.

36.3 Simple CSW Flask Example

```
import logging

from flask import Flask, request
```

(continues on next page)

(continued from previous page)

```
from pycsw import __version__ as pycsw_version
from pycsw.server import Csw

LOGGER = logging.getLogger(__name__)
APP = Flask(__name__)

@APP.route('/csw')
def csw_wrapper():
    """CSW wrapper"""

    LOGGER.info('Running pycsw %s', pycsw_version)

    pycsw_config = some_dict  # really comes from somewhere

    # initialize pycsw
    # pycsw_config: dict of the pycsw configuration
    #
    # env: dict of (HTTP) environment (defaults to os.environ)
    #
    # version: defaults to '3.0.0'
    my_csw = Csw(pycsw_config, request.environ, version='2.0.2')

    # dispatch the request
    http_status_code, response = my_csw.dispatch_wsgi()

    return response, http_status_code, {'Content-type': csw.contenttype}
```

CHAPTER

THIRTYSEVEN

TESTING

There are a number of test suites that perform mostly functional testing. These tests ensure that pycsw operates correctly and is compliant with the various supported standards. There is also a growing set of unit tests. These focus on smaller scope testing, in order to verify that individual bits of code are working as expected.

Tests can be run locally as part of the development cycle. They are also run on pycsw's GitHub Actions continuous integration setup against all pushes and pull requests to the code repository.

pycsw uses pytest for managing its automated tests.

Install pytest from the development requirements.

```
pip3 install -r requirements-dev.txt
```

37.1 OGC API - Records

Tests for OGC API - Records are located in `tests/functionaltests/suites/oarec`. They can be run as follows:

```
pytest tests/functionaltests/suites/oarec
```

37.2 OGC CSW

Tests for OGC CSW are located in `tests/functionaltests/suites/csw30`. They can be run as follows:

```
pytest tests/functionaltests/suites/csw30
```

37.3 OGC CITE

In addition to pycsw's own tests, all public releases are also tested via the OGC Compliance & Interoperability Testing & Evaluation Initiative (CITE). The pycsw wiki documents CITE testing procedures and status.

Tests for OGC CITE are located in `tests/functionaltests/suites/cite`. They can be run as follows:

```
pytest tests/functionaltests/suites/cite
```

37.4 Functional test suites

Most of pycsw's tests are functional tests. This means that each test case is based on the requirements mandated by the specifications of the various standards that pycsw implements. These tests focus on making sure that pycsw works as expected.

37.4.1 Suites for xml-based standards (CSW, ATOM, etc)

A number of different test suites exist under `tests/functionaltests/suites`. Each suite specifies the following structure:

- A mandatory `default.yml` file with the pycsw configuration that must be used by the test suite;
- A mandatory `expected/` directory containing the expected results for each request;
- An optional `data/` directory that contains `.xml` files with testing data that is to be loaded into the suite's database before running the tests. The presence of this directory and its contents have the following meaning for tests:
  - If `data/` directory is present and contains files, they will be loaded into a new database for running the tests of the suite;
  - If `data/` directory is present and does not contain any data files, a new empty database is used in the tests;
  - If `data/` directory is absent, the suite will use a database populated with test data from the `CITE` suite.
- An optional `get/requests.txt` file that holds request parameters used for making HTTP GET requests.

  Each line in the file must be formatted with the following scheme:

  test_id,request_query_string

  For example:

  TestGetCapabilities,service=CSW&version=2.0.2&request=GetCapabilities

  When tests are run, the *test_id* is used for naming each test and for finding the expected result.
- An optional `post/` directory that holds `.xml` files used for making HTTP POST requests

Test generation uses pytest's *pytest_generate_tests* function. This function is implemented in *tests/functionaltests/conftest.py*. It provides an automatic parametrization of the *tests/functionaltests/test_suites_functional:test_suites* test. This parametrization causes the generation of a test for each of the GET and POST requests defined in a suite's directory.

37.4.2 Suites for JSON-based standards (OGC API - Records, STAC API, etc)

These are implemented as simple pytest-based tests, for which no custom test generation function exists. They are simpler to generate - look into the implementation in *tests/functionaltests/suites/oarec* for examples.

37.5 Unit tests

pycsw also features unit tests. These deal with testing the expected behaviour of individual functions.

The usual implementation of unit tests is to import the function/method under test, run it with a set of known arguments and assert that the result matches the expected outcome.

Unit tests are defined in *pycsw/tests/unittests/<module_name>*.

pycsw's unit tests are marked with the *unit* marker. This makes it easy to run them in isolation:

```
# running only the unit tests (not the functional ones)
pytest -m unit
```

37.6 Running tests

Since pycsw uses pytest, tests are run with the `pytest` runner. A basic test run can be made with:

```
pytest
```

This command will run all tests and report on the number of successes, failures and also the time it took to run them. The *pytest* command accepts several additional parameters that can be used in order to customize the execution of tests. Look into pytest's invocation documentation for a more complete description. You can also get a description of the available parameters by running:

```
pytest --help
```

37.6.1 Running specific suites and test cases

pytest allows tagging tests with markers. These can be used to selectively run some tests. pycsw uses two markers:

- `unit` - run only input tests
- `functional`- run only functional tests

Markers can be specified by using the `-m <marker_name>` flag.

```
pytest -m functional  # run only functional tests
```

You can also use the `-k <name_expression>` flag to select which tests to run. Since each test's name includes the suite name, http method and an identifier for the test, it is easy to run only certain tests.

```
pytest -k "apiso and GetRecords"  # run only tests from the apiso suite that have␣
↪GetRecords in their name
pytest -k "post and GetRecords"  # run only tests that use HTTP POST and GetRecords in␣
↪their name
pytest -k "not harvesting"  # run all tests except those from the harvesting suite
```

The `-m` and `-k` flags can be combined.

37.6.2 Exiting fast

The `--exitfirst` (or `-x`) flag can be used to stop the test runner immediately as soon as a test case fails.

```
pytest --exitfirst
```

37.6.3 Seeing more output

There are three main ways to get more output from running tests:

- The `--verbose` (or `-v`) flag;
- The `--capture=no` flag - Messages sent to stdout by a test are not suppressed;
- The `--pycsw-loglevel` flag - Sets the log level of the pycsw instance under test. Set this value to `debug` in order to see all debug messages sent by pycsw while processing a request.

```
pytest --verbose
pytest --pycsw-loglevel=debug
pytest -v --capture=no --pycsw-loglevel=debug
```

37.6.4 Comparing xml-based suite results with difflib instead of XML c14n

Functional tests for XML-based suites compare results with their expected values by using XML canonicalisation - XML c14n. Alternatively, you can call pytest with the `--functional-prefer-diffs` flag. This will enable comparison based on Python's `difflib`. Comparison is made on a line-by-line basis and in case of failure, a unified diff will be printed to standard output.

```
pytest -m functional -k 'harvesting' --functional-prefer-diffs
```

37.6.5 Saving test results for disk

The result of each XML-based suite test can be saved to disk by using the `--functional-save-results-directory` option. Each result file is named after the test identifier it has when running with pytest.

```
pytest -m functional -k 'not harvesting' --functional-save-results-directory=/tmp/pycsw-
↪test-results
```

37.6.6 Test coverage

Use the *–cov pycsw* flag in order to see information on code coverage. It is possible to get output in a variety of formats.

```
pytest --cov pycsw
```

37.6.7 Specifying a timeout for tests

The *–timeout <seconds>* option can be used to specify that if a test takes more than *<seconds>* to run it is considered to have failed. Seconds can be a float, so it is possibe to specify sub-second timeouts

```
pytest --timeout=1.5
```

37.6.8 Linting with flake8

Use the *–flake8* flag to also check if the code complies with Python's style guide

```
pytest --flake8
```

37.6.9 Testing multiple Python versions

For testing multiple Python versions and configurations simultaneously you can use tox. pycsw includes a *tox.ini* file with a suitable configuration. It can be used to run tests against multiple Python versions and also multiple database backends. When running *tox* you can send arguments to the *pytest* runner by using the invocation *tox <tox arguments> – <pytest arguments>*. Examples:

```
# install tox on your system
sudo pip3 install tox

# run all tests on multiple Python versions against all databases,
# with default arguments
tox

# run tests only with python3.7 and using sqlite as backend
tox -e py37 -sqlite
```

(continues on next page)

(continued from previous page)

```
# run only csw30 suite tests with python3.7 and postgresql as backend
tox -e py37-postgresql -- -k 'csw30'
```

37.6.10 Web Testing

You can also use the pycsw tests via your web browser to perform sample requests against your pycsw install. The tests are is located in `tests/`. To generate the HTML page:

```
python3 gen_html.py > index.html
```

Then navigate to `http://host/path/to/pycsw/tests/index.html`.

CHAPTER

THIRTYEIGHT

PYCSW MIGRATION GUIDE

This page provides migration support across pycsw versions over time to help with pycsw change management.

38.1 pycsw 2.x to 3.0 Migration

- the default configuration is now in YAML format. See *Configuration* for more information. A helper script (`pycsw-admin.py migrate-config`) is included for updating from the previous configuration format
- the default endpoint for standalone deployments is now powered by `pycsw/wsgi_flask.py` (based on Flask) which supports ALL pycsw supported APIs. Make sure to use `requirements-standalone.txt` on top of `requirements.txt` to install Flask along with other standalone requirements
- the previously used `pycsw/wsgi.py` can still be used for CSW only deployments or for applications that need to integrate pycsw as a library (e.g. Django applications). PyPI installations still use `requirements.txt` which does not install Flask by default
- the default endpoint `/` is now OGC API - Records
- the CSW endpoint is now `/csw`
- the OAI-PMH endpoint is now `/oaipmh`
- the OpenSearch endpoint is now `/opensearch`
- the SRU endpoint is now `/sru`
- the `pycsw-admin.py` syntax has been updated
  - the `-c` flag has been replaced by subcommands (i.e. `pycsw-admin.py -c load_records` -> `pycsw-admin.py load-records`)
  - subcommands have been slugified (i.e. `load_records` -> `load-records`)
  - consult `--help` to use the updated CLI syntax
- use the following migration script to add new model fields

```
alter table records add column metadata TEXT;
alter table records add column metadata_type TEXT default 'application/xml';
alter table records add column edition TEXT;
alter table records add column contacts TEXT;
alter table records add column themes TEXT;
vacuum;
```

38.2 pycsw 1.x to 2.0 Migration

- the default CSW version is now 3.0.0. CSW clients need to explicitly specify `version=2.0.2` for CSW 2 behaviour. Also, pycsw administrators can use a WSGI wrapper to the pycsw API to force `version=2.0.2` on init of `pycsw.server.Csw` from the server. See *CSW Support* for more information.
- `pycsw.server.Csw.dispatch_wsgi()` previously returned the response content as a string. 2.0.0 introduces a compatability break to additionally return the HTTP status code along with the response as a list

```
from pycsw.server import Csw
my_csw = Csw(my_dict)  # add: env=some_environ_dict,  version='2.0.2' if preferred

# using pycsw 1.x
response = my_csw.dispatch_wsgi()

# using pycsw 2.0
http_status_code, response = my_csw.dispatch_wsgi()

# covering either pycsw version
content = csw.dispatch_wsgi()

# pycsw 2.0 has an API break:
# pycsw < 2.0: content = xml_response
# pycsw >= 2.0: content = [http_status_code, content]
# deal with the API break
if isinstance(content, list):  # pycsw 2.0+
    http_response_code, response = content
```

See *API* for more information.

CHAPTER

THIRTYNINE

CATALOGUING AND METADATA TOOLS

39.1 OGC API - Records Clients and Servers

- OGC API - Records official implementations

39.2 CSW Clients

- Geoportal CSW Clients
- OWSLib
- QGIS MetaSearch

39.3 CSW Servers

- deegree
- GeoNetwork opensource

39.4 Metadata Editing Tools

- pygeometa
- CatMDEdit
- EUOSME
- GIMED
- Metatools (QGIS plugin)

CHAPTER

FORTY

SUPPORT

40.1 Community

Please see the Community page for information on the pycsw community, getting support, and how to get involved.

CHAPTER

FORTYONE

CONTRIBUTING TO PYCSW

The pycsw project openly welcomes contributions (bug reports, bug fixes, code enhancements/features, etc.). This document will outline some guidelines on contributing to pycsw. As well, the pycsw community is a great place to get an idea of how to connect and participate in pycsw community and development.

pycsw has the following modes of contribution:

- GitHub Commit Access
- GitHub Pull Requests

41.1 Code of Conduct

Contributors to this project are expected to act respectfully toward others in accordance with the OSGeo Code of Conduct.

41.2 Contributions and Licensing

Contributors are asked to confirm that they comply with project license guidelines.

41.2.1 GitHub Commit Access

- proposals to provide developers with GitHub commit access shall be emailed to the pycsw-devel mailing list. Proposals shall be approved by the pycsw development team. Committers shall be added by the project admin
- removal of commit access shall be handled in the same manner
- each committer must send an email to the pycsw mailing list agreeing to the license guidelines (see *Contributions and Licensing Agreement Template*). **This is only required once**
- each committer shall be listed in https://github.com/geopython/pycsw/blob/master/COMMITTERS.txt

41.2.2 GitHub Pull Requests

- pull requests can provide agreement to license guidelines as text in the pull request or via email to the pycsw mailing list (see *Contributions and Licensing Agreement Template*). **This is only required for a contributor's first pull request. Subsequent pull requests do not require this step**
- pull requests may include copyright in the source code header by the contributor if the contribution is significant or the contributor wants to claim copyright on their contribution
- all contributors shall be listed at https://github.com/geopython/pycsw/graphs/contributors
- unclaimed copyright, by default, is assigned to the main copyright holders as specified in https://github.com/geopython/pycsw/blob/master/LICENSE.txt

41.2.3 Contributions and Licensing Agreement Template

```
Hi all, I'd like to contribute <feature X|bugfix Y|docs|something else> to pycsw.
I confirm that my contributions to pycsw will be compatible with the pycsw license
guidelines at the time of contribution.
```

41.3 GitHub

Code, tests, documentation, wiki and issue tracking are all managed on GitHub. Make sure you have a GitHub account.

41.4 Code Overview

- the pycsw wiki documents an overview of the codebase

41.5 Documentation

- documentation is managed in `docs/`, in reStructuredText format
- Sphinx is used to generate the documentation
- See the reStructuredText Primer on rST markup and syntax.

41.6 Bugs

pycsw's issue tracker is the place to report bugs or request enhancements. To submit a bug be sure to specify the pycsw version you are using, the appropriate component, a description of how to reproduce the bug, as well as what version of Python and platform. For convenience, you can run `pycsw-admin.py get-sysprof` and copy/paste the output into your issue.

41.7 Forking pycsw

Contributions are most easily managed via GitHub pull requests. Fork pycsw into your own GitHub repository to be able to commit your work and submit pull requests.

41.8 Development

41.8.1 GitHub Commit Guidelines

- enhancements and bug fixes should be identified with a GitHub issue
- commits should be granular enough for other developers to understand the nature / implications of the change(s)
- non-trivial Git commits shall be associated with a GitHub issue. As documentation can always be improved, tickets need not be opened for improving the docs
- Git commits shall include a description of changes
- Git commits shall include the GitHub issue number (i.e. `#1234`) in the Git commit log message
- all enhancements or bug fixes must successfully pass all *OGC CITE* tests before they are committed
- all enhancements or bug fixes must successfully pass all *Testing* tests before they are committed
- enhancements which can be demonstrated from the pycsw *Testing* should be accompanied by example CSW request XML

41.8.2 Coding Guidelines

- pycsw instead of PyCSW, pyCSW, Pycsw
- always code with PEP 8 conventions
- always run source code through flake8 and pylint, using all pylint defaults except for `C0111`. `sbin/pycsw-pylint.sh` is included for convenience
- for exceptions which make their way to OGC `ExceptionReport` XML, always specify the appropriate `locator` and `code` parameters

41.8.3 Submitting a Pull Request

This section will guide you through steps of working on pycsw. This section assumes you have forked pycsw into your own GitHub repository.

```
# setup a virtualenv
virtualenv mypycsw && cd mypycsw
. ./bin/activate
# clone the repository locally
git clone git@github.com:USERNAME/pycsw.git
cd pycsw
pip install -e . && pip install -r requirements-standalone.txt
# add the main pycsw master branch to keep up to date with upstream changes
git remote add upstream https://github.com/geopython/pycsw.git
git pull upstream master
# create a local branch off master
# The name of the branch should include the issue number if it exists
git branch issue-72
git checkout issue-72
#
# make code/doc changes
#
git commit -am 'fix xyz (#72)'
git push origin issue-72
```

Your changes are now visible on your pycsw repository on GitHub. You are now ready to create a pull request. A member of the pycsw team will review the pull request and provide feedback / suggestions if required. If changes are required, make them against the same branch and push as per above (all changes to the branch in the pull request apply).

The pull request will then be merged by the pycsw team. You can then delete your local branch (on GitHub), and then update your own repository to ensure your pycsw repository is up to date with pycsw master:

```
git checkout master
git pull upstream master
```

CHAPTER

FORTYTWO

LICENSE

CHAPTER

FORTYTHREE

THE MIT LICENSE (MIT)

Copyright © 2010-2026 Tom Kralidis Copyright © 2011-2026 Angelos Tzotsos Copyright © 2012-2015 Adam Hinz Copyright © 2015-2021 Ricardo Garcia Silva

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

43.1 Documentation

The documentation is released under the Creative Commons Attribution 4.0 International (CC BY 4.0) license.

CHAPTER

FORTYFOUR

COMMITTERS

Login(s)	Name	Email / Contact	Area(s)
tomkralidis	Tom Kralidis	tomkralidis at gmail.com	Overall
kalxas	Angelos Tzotsos	tzotsos at gmail.com	INSPIRE, APISO profiles, Packaging
adamhinz	Adam Hinz	hinz dot adam at gmail.com	WSGI/Server Deployment
ricardogsilva	Ricardo Garcia Silva	ricardo.garcia.silva at gmail.com	Overall