pycsw Documentation

Release 2.4.1

Tom Kralidis

2019-08-31

Contents

Author Tom Kralidis

Contact tomkralidis at gmail.com

Release 2.4.1

Date 2019-08-31

CHAPTER 1

Introduction

pycsw is an OGC CSW server implementation written in Python.

CHAPTER 2

Features

- 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
- supports ISO, Dublin Core, DIF, FGDC, Atom and GM03 metadata models
- CGI or WSGI deployment
- Python 2 and 3 compatible
- simple configuration
- transactional capabilities (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 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
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

2.2 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*

2.3 Supported Output Formats

- XML (default)
- JSON

2.4 Supported Output Schemas

- Dublin Core
- ISO 19139
- FGDC CSDGM
- NASA DIF
- Atom
- GM03

2.5 Supported Sorting Functionality

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

2.6 Supported Filters

2.6.1 Full Text Search

- csw:AnyText

2.6.2 Geometry Operands

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

Note: Coordinate transformations are supported

2.6.3 Spatial Operators

- BBOX
- Beyond
- Contains
- Crosses
- Disjoint

- DWithin
- Equals
- Intersects
- Overlaps
- Touches
- Within

2.6.4 Logical Operators

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

2.6.5 Functions

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

CHAPTER 3

Installation

3.1 System Requirements

pycsw is written in Python, and works with (tested) version 2.7, 3.4 and 3.5

pycsw requires the following Python supporting libraries:

- lxml for XML support
- SQLAlchemy for database bindings
- pyproj for coordinate transformations
- Shapely for spatial query / geometry support
- OWSLib for CSW client and metadata parser
- six for Python 2/3 compatibility
- xmltodict for working with XML similar to working with JSON
- geolinks for dealing with geospatial links

Note: You can install these dependencies via easy_install or 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
$ pip install -e . && pip install -r requirements-standalone.txt
$ cp default-sample.cfg default.cfg
$ vi default.cfg
# adjust paths in
# - server.home
# - repository.database
# set server.url to http://localhost:8000/
$ python pycsw/wsgi.py
$ curl http://localhost:8000/?service=CSW&version=2.0.2&request=GetCapabilities
```

3.2.2 The Quick and Dirty Way

```
$ git clone git://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 git://github.com/geopython/pycsw.git
$ python setup.py build
$ python 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)

```
# easy_install or pip will do the trick
$ easy_install pycsw
# or
$ pip 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 http://download.opensuse.org/repositories/Application:/Geo/openSUSE_Leap_
↪42.1/ 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 http://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:

```
#!/Python27/python -u
```

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

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.cfg` 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.cfg` 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 ~ "\.(cfg)$">
 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:

```
$ python ./pycsw/wsgi.py
Serving on port 8000...
```

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

CHAPTER 4

Docker

pycsw is available as a Docker image. The image is hosted on the Docker Hub.

Assuming you already have docker installed, you can get a pycsw instance up and running by issuing the following command:

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

Docker will retrieve the pycsw image from Docker Hub (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.

4.1 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.

4.2 Using pycsw-admin

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

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

4.3 Running custom pycsw containers

4.3.1 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.cfg`.

Supplying the configuration file via bind mount:

```
docker run \
    --name pycsw \
    --detach \
    --volume <path-to-local-pycsw.cfg>:/etc/pycsw/pycsw.cfg \
    --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.cfg>
docker service create \
    --name pycsw \
    --secret src=pycsw-config,target=/etc/pycsw/pycsw.cfg \
    --publish 8000:8000
    geopython/pycsw
```

4.3.2 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
```

4.3.3 PostgreSQL repositories

Specifying a PostgreSQL repository is just a matter of configuring a custom pycsw.cfg 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.

4.4 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/python3.5/site-packages/pycsw \
    --volume ${PWD}/docs:/home/pycsw/docs \
    --volume ${PWD}/VERSION.txt:/home/pycsw/VERSION.txt \
    --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 requirements-dev.txt

# run tests (for example unit tests)
docker exec -ti pycsw-dev py.test -m unit

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

Note: Please note that the pycsw image only uses python 3.5 and that it also 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 5

Configuration

pycsw's runtime configuration is defined by `default.cfg`. pycsw ships with a sample configuration (`default-sample.cfg`). Copy the file to `default.cfg` 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
- **loglevel**: the logging level (see http://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)
- **federatedcatalogues**: comma delimited list of CSW 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
- **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`

- **profiles**: comma delimited list of profiles to load at runtime (default is none). See *Profile Plugins*
- **smtp_host**: SMTP host for processing `csw:ResponseHandler` parameter via outgoing email requests (default is `localhost`)
- **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.

[manager]

- **transactions**: whether to enable transactions (`true` or `false`). Default is `false` (see *Transactions*)
- **allowed_ips**: comma delimited 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*)
- **csw_harvest_pagesize**: when harvesting other CSW servers, the number of records per request to page by (default is 10)

[metadata:main]

- **identification_title**: the title of the service
- **identification_abstract**: some descriptive text about the service
- **identification_keywords**: comma delimited list of keywords about the service
- **identification_keywords_type**: keyword type as per the ISO 19115 MD_KeywordTypeCode codelist). Accepted values are `discipline`, `temporal`, `place`, `theme`, `stratum`
- **identification_fees**: fees associated with the service
- **identification_accessconstraints**: access constraints associated with the service
- **provider_name**: the name of the service provider
- **provider_url**: the URL of the service provider
- **contact_name**: the name of the provider contact
- **contact_position**: the position title of the provider contact
- **contact_address**: the address of the provider contact
- **contact_city**: the city of the provider contact
- **contact_stateorprovince**: the province or territory of the provider contact
- **contact_postalcode**: the postal code of the provider contact
- **contact_country**: the country of the provider contact
- **contact_phone**: the phone number of the provider contact
- **contact_fax**: the facsimile number of the provider contact
- **contact_email**: the email address of the provider contact
- **contact_url**: the URL to more information about the provider contact
- **contact_hours**: the hours of service to contact the provider
- **contact_instructions**: the how to contact the provider contact
- **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 http://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*)

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

5.1 MaxRecords Handling

The The following describes how `maxRecords` is handled by the configuration when handling `GetRecords` requests:

server.maxrecords	GetRecords.maxRecords	Result
none set	none passed	10 (CSW default)
20	14	20
20	none passed	20
none set	100	100
20	200	20

5.2 Alternate Configurations

By default, pycsw loads `default.cfg` 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.cfg` to the base URL of the service (e.g. `http://localhost/pycsw/csw.py?config=tests/suites/default/default.cfg&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.

5.2.1 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

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

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

Note: Apache must be restarted after changes to `httpd.conf`

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.cfg

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

CHAPTER 6

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 -h` to see all administration operations and parameters

6.1 Metadata Repository Setup

pycsw supports the following databases:

- SQLite3
- PostgreSQL
- PostgreSQL with PostGIS enabled
- MySQL

Note: The easiest and fastest way to deploy pycsw is to use SQLite3 as the backend.

Note: PostgreSQL support includes support for PostGIS functions if enabled

Note: If PostGIS (1.x or 2.x) 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

6.2 Supported Information Models

By default, pycsw supports the `csw:Record` information model.

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

6.3 Setting up the Database

```
$ pycsw-admin.py -c setup_db -f default.cfg
```

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 OGR:

```
$ 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.

6.4 Loading Records

```
$ pycsw-admin.py -c load_records -f default.cfg -p /path/to/records
```

This will import all `*.xml` records from `/path/to/records` into the database specified in `default.cfg` (`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*).

6.5 Exporting the Repository

```
$ pycsw-admin.py -c export_records -f default.cfg -p /path/to/output_dir
```

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

6.6 Optimizing the Database

```
$ pycsw-admin.py -c optimize_db -f default.cfg
```

Note: This feature is relevant only for PostgreSQL and MySQL

6.7 Deleting Records from the Repository

```
$ pycsw-admin.py -c delete_records -f default.cfg
```

This will empty the repository of all records.

6.8 Database Specific Notes

6.8.1 PostgreSQL

- if PostGIS is not enabled, pycsw makes uses of PL/Python functions. To enable PostgreSQL support, the database user must be able to create functions within the database. In case of recent PostgreSQL versions (9.x), the PL/Python extension must be enabled prior to pycsw setup
- 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.admin.setup_db`. Any query against *csw:AnyText* or *apiso:AnyText* will process using PostgreSQL FTS handling

6.8.2 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.admin.setup_db`). Also the native geometries must be filled manually from the WKT field. Next versions of pycsw will automate this process

6.9 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/config.py:MD_CORE_MODEL`).

To override the default settings:

- define a custom database mapping based on `etc/mappings.py`
- in `default.cfg`, 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 Configuration*, and *Open Data Catalog Configuration* for further examples.

6.9.1 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
- `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:Links`: structure of links in the format "name,description,protocol,url[^,,,[^,,,]]"

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`.

CHAPTER 7

CSW Support

7.1 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
```

7.2 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 8

Distributed Searching

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

pycsw has the ability to perform distributed searching against other CSW servers. Distributed searching is disabled by default; to enable, `server.federatedcatalogues` 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.

8.1 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.cfg`

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

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

...where the `*.cfg` 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.cfg`

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.cfg</ows:Value>
        <ows:Value>http://localhost/pycsw/csw.py?config=CSW-2.cfg</ows:Value>
        <ows:Value>http://localhost/pycsw/csw.py?config=CSW-3.cfg</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:

```
[server]
...
federatedcatalogues=http://localhost/pycsw/csw.py?config=CSW-1.cfg,http://localhost/
↪pycsw/csw.py?config=CSW-2.cfg,http://localhost/pycsw/csw.py?config=CSW-3.cfg
```

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.

CHAPTER 9

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. 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 http://www.loc.gov/standards/sru/simple.html for example SRU requests.

CHAPTER 10

OpenSearch Support

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}`

OpenSearch support is enabled by default. 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.

CHAPTER 11

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. 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 12

JSON Support

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 13

SOAP

pycsw 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 14

XML Sitemaps

XML Sitemaps can be generated by running:

```
$ pycsw-admin.py -c gen_sitemap -f default.cfg -o 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 15

Transactions

pycsw 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`.

15.1 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:urn`		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.

15.2 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://`) 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*.

15.2.1 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.

15.3 Transactions

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://`) if the Transaction request specifies `csw:ResponseHandler`.

The *Testing* contain CSW-T request examples.

CHAPTER 16

Repository Filters

pycsw has the ability to perform server side repository / database filters as a means to mask all CSW 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.

16.1 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 CSW 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]
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 17

Profile Plugins

17.1 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.

17.2 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.
```

17.3 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
```

(continues on next page)

(continued from previous page)

```
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'],
        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`.

17.4 Enabling Profiles

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

17.5 Testing

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

CHAPTER 18

Supported Profiles

18.1 ISO Metadata Application Profile (1.0.0)

18.1.1 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.

18.1.2 Configuration

No extra configuration is required.

18.1.3 Querying

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

18.1.4 Enabling APISO Support

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

18.1.5 Testing

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

18.2 INSPIRE Extension

18.2.1 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`.

18.2.2 Configuration

[metadata: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**: a comma-seperated keyword 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`)

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

18.3.1 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.

18.3.2 Configuration

No extra configuration is required.

18.3.3 Querying

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

18.3.4 Enabling ebRIM Support

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

18.3.5 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 19

Repository Plugins

19.1 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 CSW and other APIs atop a given application.

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

19.2 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

19.3 Configuration

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

```
[repository]
mappings='path.to.repo_plugin.MyRepository'
```

CHAPTER 20

Output Schema Plugins

20.1 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.

20.2 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
```

20.3 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.

20.4 Testing

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

CHAPTER 21

GeoNode Configuration

GeoNode (http://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).

21.1 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 http://docs.geonode.org/en/latest/developers/reference/django-apps.html#id1 for customizing pycsw within GeoNode.

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

CHAPTER 22

HHypermap Configuration

HHypermap (Harvard Hypermap) Registry (https://github.com/cga-harvard/HHypermap) 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.

22.1 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 23

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).

23.1 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 24

CKAN Configuration

CKAN (http://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.

24.1 CKAN Setup

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

CHAPTER 25

API

Python applications can integrate pycsw into their custom workflows. This allows for seamless integate within frameworks like 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

25.1 Simple Flask Example

```
import logging

from flask import Flask, request

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
```

(continues on next page)

(continued from previous page)

```
    # pycsw_config: either a ConfigParser object or a 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 26

Testing

Pycsw uses pytest for managing its automated tests. There are a number of test suites that perform mostly functional testing. These tests ensure that pycsw 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 Travis continuous integration server against all pushes and pull requests to the code repository.

26.1 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.

26.2 Functional test suites

Currently 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.

Each test follows the same workflow:

- Create a new pycsw instance with a custom configuration and data repository for each suite of tests;
- Perform a series of GET and POST requests to the running pycsw instance;
- Compare the results of each request against a previously prepared expected result. If the test result matches the expected outcome the test passes, otherwise it fails.

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

- A mandatory `default.cfg` 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

26.2.1 Test identifiers

Each test has an identifier that is built using the following rule:

<test_function>[<suite_name>_<http_method>_<test_name>]

For example:

test_suites[default_post_GetRecords-end]

26.2.2 Functional tests' implementation

Functional tests are generated for each suite directory present under *tests/functionaltests/suites*. 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* function. This parametrization causes the generation of a test for each of the GET and POST requests defined in a suite's directory.

26.2.3 Adding New Tests

To add tests to an existing suite:

- for HTTP POST tests, add XML documents to `tests/functionaltests/suites/<suite>/post`
- for HTTP GET tests, add tests (one per line) to `tests/functionaltests/suites/<suite>/get/requests.txt`

To add a new test suite:

- Create a new directory under `tests/functionaltests/suites` (e.g. `foo`)
- Create a new configuration in `tests/suites/foo/default.cfg`
- Populate HTTP POST requests in `tests/suites/foo/post`
- Populate HTTP GET requests in `tests/suites/foo/get/requests.txt`

- If the test suite requires test data, create `tests/suites/foo/data` and store XML files there. These will be inserted in the test catalogue at test runtime
- Use pytest or tox as described above in order to run the tests

The new test suite database will be created automatically and used as part of tests.

26.3 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)
py.test -m unit
```

26.4 Running tests

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

```
py.test
```

This command will run all tests and report on the number of successes, failures and also the time it took to run them. The *py.test* 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:

```
py.test --help
```

26.4.1 Running specific suites and test cases

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

- `unit` - run only inut tests
- `functional`- run onyl functional tests

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

```
py.test -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.

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

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

26.4.2 Exiting fast

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

```
py.test --exitfirst
```

26.4.3 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.

```
py.test --verbose
py.test --pycsw-loglevel=debug
py.test -v --capture=no --pycsw-loglevel=debug
```

26.4.4 Comparing results with difflib instead of XML c14n

The functional tests compare results with their expected values by using [XML canonicalisation - XML c14n](https://www.w3.org/TR/xml-c14n/). Alternatively, you can call py.test 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.

```
py.test -m functional -k 'harvesting' --functional-prefer-diffs
```

26.4.5 Saving test results for disk

The result of each functional 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.

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

26.4.6 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.

```
py.test --cov pycsw
```

26.4.7 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

```
py.test --timeout=1.5
```

26.4.8 Linting with flake8

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

```
py.test --flake8
```

26.4.9 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 *py.test* runner by using the invocation *tox <tox arguments> – <py.test arguments>*. Examples:

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

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

# run tests only with python2.7 and using sqlite as backend
tox -e py27-sqlite

# run only csw30 suite tests with python3.5 and postgresql as backend
tox -e py35-postgresql -- -k 'csw30'
```

26.4.10 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:

```
$ paver gen_tests_html
```

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

CHAPTER 27

pycsw Migration Guide

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

27.1 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 28

Cataloguing and Metadata Tools

28.1 CSW Clients

- Geoportal CSW Clients
- OWSLib
- MetaSearch (QGIS plugin)

28.2 CSW Servers

- deegree
- eXcat
- GeoNetwork opensource

28.3 Metadata Editing Tools

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

CHAPTER 29

Support

29.1 Community

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

CHAPTER 30

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

30.1 Code of Conduct

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

30.2 Contributions and Licensing

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

30.2.1 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

30.2.2 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

30.2.3 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.
```

30.3 GitHub

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

30.4 Code Overview

- the pycsw wiki documents an overview of the codebase

30.5 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.

30.6 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 -c get_sysprof` and copy/paste the output into your issue.

30.7 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.

30.8 Development

30.8.1 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)
- for trivial commits that do not need Travis CI to run, include `[ci skip]` as part of the commit message
- 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

30.8.2 Coding Guidelines

- pycsw instead of PyCSW, pyCSW, Pycsw
- always code with PEP 8 conventions
- always run source code through `pep8` 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
- the pycsw wiki documents developer tasks for things like releasing documentation, testing, etc.

30.8.3 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
#
```

(continues on next page)

(continued from previous page)

```
# 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 31

License

The MIT License (MIT)

Copyright (c) 2010-2015 Tom Kralidis Copyright (c) 2011-2015 Angelos Tzotsos Copyright (c) 2012-2015 Adam Hinz Copyright (c) 2015 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.

31.1 Documentation

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

CHAPTER 32

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