# Contents

1 Introduction 3

2 Features 5
   2.1 Standards Support ........................... 6
   2.2 Supported Operations .......................... 6
   2.3 Supported Output Formats ...................... 6
   2.4 Supported Output Schemas ..................... 6
   2.5 Supported Sorting Functionality ............... 7
   2.6 Supported Filters ........................... 7

3 Installation 9
   3.1 System Requirements ........................... 9
   3.2 Installing from Source .......................... 9
   3.3 Installing from the Python Package Index (PyPi) 10
   3.4 Installing from OpenSUSE Build Service ........ 10
   3.5 Installing on Ubuntu/Xubuntu/Kubuntu .......... 11
   3.6 Running on Windows ............................ 11
   3.7 Security ....................................... 11
   3.8 Running on WSGI ............................... 11

4 Configuration 13
   4.1 MaxRecords Handling ........................... 15
   4.2 Alternate Configurations ....................... 15

5 Administration 17
   5.1 Metadata Repository Setup ..................... 17
   5.2 Supported Information Models .................. 17
   5.3 Setting up the Database ....................... 18
   5.4 Loading Records ............................... 18
   5.5 Exporting the Repository ...................... 18
   5.6 Optimizing the Database ...................... 18
   5.7 Deleting Records from the Repository ........ 18
   5.8 Database Specific Notes ....................... 19
   5.9 Mapping to an Existing Repository .......... 19

6 Distributed Searching 21
   6.1 Scenario: Federated Search ................... 21

7 Search/Retrieval via URL (SRU) Support 23
23 Contributing to pycsw

23.1 GitHub ......................................................... 57
23.2 Code Overview .................................................. 57
23.3 Documentation .................................................. 57
23.4 Bugs .............................................................. 57
23.5 Forking pycsw .................................................... 57
23.6 Development ..................................................... 58
23.7 GitHub Commit Access .......................................... 59

24 License ......................................................... 61

25 Committers ..................................................... 63
Author  Tom Kralidis

Contact  tomkralidis at gmail.com

Release  1.10.1

Date  2014-12-29
Introduction

pycsw is an OGC CSW server implementation written in Python.
• certified OGC Compliant and OGC Reference Implementation
• 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 and Atom metadata models
• CGI or WSGI deployment
• simple configuration
• transactional capabilities (CSW-T)
• flexible repository configuration
• GeoNode connectivity
• Open Data Catalog connectivity
• CKAN connectivity
• federated catalogue distributed searching
• realtime XML Schema validation
• extensible profile plugin architecture
2.1 Standards Support

<table>
<thead>
<tr>
<th>Standard</th>
<th>Version(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OGC CSW</td>
<td>2.0.2</td>
</tr>
<tr>
<td>OGC Filter</td>
<td>1.1.0</td>
</tr>
<tr>
<td>OGC OWS Common</td>
<td>1.0.0</td>
</tr>
<tr>
<td>OGC GML</td>
<td>3.1.1</td>
</tr>
<tr>
<td>OGC SFSQL</td>
<td>1.2.1</td>
</tr>
<tr>
<td>Dublin Core</td>
<td>1.1</td>
</tr>
<tr>
<td>SOAP</td>
<td>1.2</td>
</tr>
<tr>
<td>ISO 19115</td>
<td>2003</td>
</tr>
<tr>
<td>ISO 19139</td>
<td>2007</td>
</tr>
<tr>
<td>ISO 19119</td>
<td>2005</td>
</tr>
<tr>
<td>NASA DIF</td>
<td>9.7</td>
</tr>
<tr>
<td>FGDC CSDGM</td>
<td>1998</td>
</tr>
<tr>
<td>SRU</td>
<td>1.1</td>
</tr>
<tr>
<td>OGC OpenSearch</td>
<td>1.0</td>
</tr>
<tr>
<td>OAI-PMH</td>
<td>2.0</td>
</tr>
</tbody>
</table>

2.2 Supported Operations

<table>
<thead>
<tr>
<th>Request</th>
<th>Optionality</th>
<th>Supported</th>
<th>HTTP method binding(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetCapabilities</td>
<td>mandatory</td>
<td>yes</td>
<td>GET (KVP) / POST (XML) / SOAP</td>
</tr>
<tr>
<td>DescribeRecord</td>
<td>mandatory</td>
<td>yes</td>
<td>GET (KVP) / POST (XML) / SOAP</td>
</tr>
<tr>
<td>GetRecords</td>
<td>mandatory</td>
<td>yes</td>
<td>GET (KVP) / POST (XML) / SOAP</td>
</tr>
<tr>
<td>GetRecordById</td>
<td>optional</td>
<td>yes</td>
<td>GET (KVP) / POST (XML) / SOAP</td>
</tr>
<tr>
<td>GetRepositoryItem</td>
<td>optional</td>
<td>yes</td>
<td>GET (KVP) / POST (XML) / SOAP</td>
</tr>
<tr>
<td>GetDomain</td>
<td>optional</td>
<td>yes</td>
<td>GET (KVP) / POST (XML) / SOAP</td>
</tr>
<tr>
<td>Harvest</td>
<td>optional</td>
<td>yes</td>
<td>GET (KVP) / POST (XML) / SOAP</td>
</tr>
<tr>
<td>Transaction</td>
<td>optional</td>
<td>yes</td>
<td>POST (XML) / SOAP</td>
</tr>
</tbody>
</table>

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

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

• 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
3.1 System Requirements

pycsw is written in Python, and works with (tested) version 2.6 and 2.7

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

Note: You can install these dependencies via easy_install or pip

Note: For GeoNode or Open Data Catalog 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 csw.wsgi
$ 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:

```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 *csw.wsgi* 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 OBS package in openSUSE 12.3, one can run the following commands as user *root*:

```
# zypper -ar http://download.opensuse.org/repositories/Application:/Geo/openSUSE_12.3:/ GEO
# zypper -ar http://download.opensuse.org/repositories/devel:/languages:/python/openSUSE_12.3:/ python
# zypper refresh
# zypper install python-pycsw pycsw-cgi
```

For earlier openSUSE versions change 12.3 with 12.2. For future openSUSE version use *Factory*. An alternative method is to use the One-Click Installer.
3.5 Installing on Ubuntu/Xubuntu/Kubuntu

In order to install pyCsw to an Ubuntu based distribution, one can run the following commands:

```sh
# sudo add-apt-repository ppa:pycsw/stable
# sudo apt-get update
# sudo apt-get install python-pycsw pycsw-cgi
```

An alternative method is to use the OSGeoLive installation script located in `pycsw/etc/dist/osgeolive`:

```sh
# cd pycsw/etc/dist
# sudo ./install_pycsw.sh
```

The script installs the dependencies (Apache, lxml, sqlalchemy, shapely, pyproj) and then pyCsw to `/var/www`.

3.6 Running on Windows

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

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

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

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 `csw.wsgi` in your WSGI server environment. Below is an example of configuring with Apache:

```sh
WSGIDaemonProcess host1 home=/var/www/pycsw processes=2
WSGIScriptAlias /pycsw-wsgi /var/www/pycsw/csw.wsgi
<Directory /var/www/pycsw>
  Order deny,allow
  Allow from all
</Directory>
```
or use the WSGI reference implementation:

```
$ python ./csw.wsgi
Serving on port 8000...
```

which will publish pycsw to http://localhost:8000/
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](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](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]**
pycsw Documentation, Release 1.10.1

- **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.

### 4.1 MaxRecords Handling

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

<table>
<thead>
<tr>
<th>server.maxrecords</th>
<th>GetRecords.maxRecords</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>none set</td>
<td>none passed</td>
<td>10 (CSW default)</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>20</td>
</tr>
<tr>
<td>20</td>
<td>none passed</td>
<td>20</td>
</tr>
<tr>
<td>none set</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>20</td>
<td>200</td>
<td>20</td>
</tr>
</tbody>
</table>

### 4.2 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 specifying 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`).

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.

#### 4.2.1 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`:

```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
```
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

### 5.1 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

### 5.2 Supported Information Models

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

**Note:** See Profile Plugins for information on enabling profiles
5.3 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.

5.4 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.

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

5.5 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.

5.6 Optimizing the Database

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

Note: This feature is relevant only for PostgreSQL and MySQL.

5.7 Deleting Records from the Repository

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

This will empty the repository of all records.
5.8 Database Specific Notes

5.8.1 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

5.8.2 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

5.9 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

See the GeoNode Configuration and Open Data Catalog Configuration for further examples.

5.9.1 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`. 
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.

### 6.1 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`:

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

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

- Core (GeoSpatial Service) `{searchTerms},{geo:box}`
- 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.
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 / response examples.
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.
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 area on your web server (parallel to or above your pycsw install location) to enable web crawlers to index your repository.
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`.

### 12.1 Supported Resource Types

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

<table>
<thead>
<tr>
<th>Resource Type</th>
<th>Namespace</th>
<th>Transaction</th>
<th>Harvest</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dublin Core</td>
<td><a href="http://www.opengis.net/cat/csw/2.0.2">http://www.opengis.net/cat/csw/2.0.2</a></td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>FGDC</td>
<td><a href="http://www.opengis.net/cat/csw/csdgm">http://www.opengis.net/cat/csw/csdgm</a></td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>ISO 19139</td>
<td><a href="http://www.isotc211.org/2005/gmd">http://www.isotc211.org/2005/gmd</a></td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>ISO GMI</td>
<td><a href="http://www.isotc211.org/2005/gmi">http://www.isotc211.org/2005/gmi</a></td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>OGC:CSW 2.0.2</td>
<td><a href="http://www.opengis.net/cat/csw/2.0.2">http://www.opengis.net/cat/csw/2.0.2</a></td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>OGC:WMS 1.1.1</td>
<td><a href="http://www.opengis.net/wms">http://www.opengis.net/wms</a></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>OGC:WFS 1.1.0</td>
<td><a href="http://www.opengis.net/wfs">http://www.opengis.net/wfs</a></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>OGC:WCS 1.0.0</td>
<td><a href="http://www.opengis.net/wcs">http://www.opengis.net/wcs</a></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>OGC:SOS 1.0.0</td>
<td><a href="http://www.opengis.net/sos/1.0">http://www.opengis.net/sos/1.0</a></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>OGC:SOS 2.0.0</td>
<td><a href="http://www.opengis.net/sos/2.0">http://www.opengis.net/sos/2.0</a></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>WAF</td>
<td>urn:geoss:urn</td>
<td></td>
<td>yes</td>
</tr>
</tbody>
</table>

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:

```bash
$ 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.

### 12.2 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.

12.2.1 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.

12.3 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 Tester contain CSW-T request examples.
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.

13.1 Scenario: One Database, Many Views

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

<table>
<thead>
<tr>
<th>identifier</th>
<th>parentidentifier</th>
<th>title</th>
<th>abstract</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>33</td>
<td>foo1</td>
<td>bar1</td>
</tr>
<tr>
<td>2</td>
<td>33</td>
<td>foo2</td>
<td>bar2</td>
</tr>
<tr>
<td>3</td>
<td>55</td>
<td>foo3</td>
<td>bar3</td>
</tr>
<tr>
<td>4</td>
<td>55</td>
<td>foo1</td>
<td>bar1</td>
</tr>
<tr>
<td>5</td>
<td>21</td>
<td>foo5</td>
<td>bar5</td>
</tr>
<tr>
<td>5</td>
<td>21</td>
<td>foo6</td>
<td>bar6</td>
</tr>
</tbody>
</table>

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

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

...will return:

<table>
<thead>
<tr>
<th>identifier</th>
<th>parentidentifier</th>
<th>title</th>
<th>abstract</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>33</td>
<td>foo1</td>
<td>bar1</td>
</tr>
<tr>
<td>4</td>
<td>55</td>
<td>foo1</td>
<td>bar1</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>identifier</th>
<th>parentidentifier</th>
<th>title</th>
<th>abstract</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>33</td>
<td>foo1</td>
<td>bar1</td>
</tr>
</tbody>
</table>

Another example:

```python
[repository]
database=sqlite:///records.db
filter=pycsw:ParentIdentifier != '33'
```

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

<table>
<thead>
<tr>
<th>identifier</th>
<th>parentidentifier</th>
<th>title</th>
<th>abstract</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>55</td>
<td>foo1</td>
<td>bar1</td>
</tr>
</tbody>
</table>

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).
14.1 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.

14.2 Requirements

pycsw/
  plugins/
    __init__.py # empty
  profiles/ # directory to store profiles
    __init__.py # empty
  apiso/ # profile directory
    __init__.py # empty
    apiso.py # profile code
    ... # supporting files, etc.

14.3 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:

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

### 14.4 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).

### 14.5 Testing

Profiles must add examples to the Tester interface, which must provide example requests specific to the profile.
15.1 ISO Metadata Application Profile (1.0.0)

15.1.1 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.

15.1.2 Configuration

No extra configuration is required.

15.1.3 Querying

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

15.1.4 Enabling APISO Support

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

15.1.5 Testing

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

15.2 INSPIRE Extension

15.2.1 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**.
15.2.2 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)

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

15.3.1 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.

15.3.2 Configuration

No extra configuration is required.

15.3.3 Querying

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

15.3.4 Enabling ebRIM Support

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

15.3.5 Testing

A testing interface is available in `tests/index.html` which contains tests specific to ebRIM to demonstrate functionality. See `Tester` for more information.
16.1 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 and Atom are default).

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

16.2 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

16.3 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.

16.4 Testing

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

17.1 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.
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).

18.1 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.

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

### 19.1 CKAN Setup

Installation and configuration Instructions are provided as part of the ckanext-spatial documentation.
CHAPTER 20

Testing

20.1 OGC CITE

Compliance benchmarking is done via the OGC Compliance & Interoperability Testing & Evaluation Initiative. The pycsw wiki documents testing procedures and status.

20.2 Tester

The pycsw tests framework (in tests) is a collection of testsuites to perform automated regression testing of the codebase. Test are run against all pushes to the GitHub repository via Travis CI.

20.2.1 Running Locally

The tests framework can be run from tests using Paver (see pavement.py) tasks for convenience:

```
$ cd /path/to/pycsw
# run all tests (starts up http://localhost:8000)
$ paver test
# run tests only against specific testsuites
$ paver test -s apiso,fgdc
# run all tests, including harvesting (this is turned off by default given the volatility of remote services)
$ paver test -r
```

The tests perform HTTP GET and POST requests against http://localhost:8000. The expected output for each test can be found in expected. Results are categorized as passed, failed, or initialized. A summary of results is output at the end of the run.

20.2.2 Failed Tests

If a given test has failed, the output is saved in results. The resulting failure can be analyzed by running diff tests/expected/name_of_test.xml tests/results/name_of_test.xml to find variances. The Paver task returns a status code which indicates the number of tests which have failed (i.e. echo $?).
20.2.3 Test Suites

The tests framework is run against a series of 'suites' (in tests/suites), each of which specifies a given configuration to test various functionality of the codebase. Each suite is structured as follows:

- tests/suites/suite/default.cfg: the configuration for the suite
- tests/suites/suite/post: directory of XML documents for HTTP POST requests
- tests/suites/suite/get/requests.txt: directory and text file of KVP for HTTP GET requests
- tests/suites/suite/data: directory of sample XML data required for the test suite. Database and test data are setup/loaded automatically as part of testing

When the tests are invoked, the following operations are run:

- pycsw configuration is set to tests/suites/suite/default.cfg
- HTTP POST requests are run against tests/suites/suite/post/*.xml
- HTTP GET requests are run against each request in tests/suites/suite/get/requests.txt

The CSV format of tests/suites/suite/get/requests.txt is testname,request, with one line for each test. The testname value is a unique test name (this value sets the name of the output file in the test results). The request value is the HTTP GET request. The PYCSW_SERVER token is replaced at runtime with the URL to the pycsw install.

20.2.4 Adding New Tests

To add tests to an existing suite:

- for HTTP POST tests, add XML documents to tests/suites/suite/post
- for HTTP GET tests, add tests (one per line) to tests/suites/suite/get/requests.txt
- run paver test

To add a new test suite:

- create a new directory under tests/suites (e.g. foo)
- create a new configuration in tests/suites/foo/default.cfg
  - Ensure that all file paths are relative to path/to/pycsw
  - Ensure that repository.database points to an SQLite3 database called tests/suites/foo/data/records.db. The database must be called records.db and the directory tests/suites/foo/data must exist
- populate HTTP POST requests in tests/suites/foo/post
- populate HTTP GET requests in tests/suites/foo/get/requests.txt
- if the testsuite requires test data, create tests/suites/foo/data are store XML file there
- run paver test (or paver test -s foo to test only the new test suite)

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

20.2.5 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 21

Cataloguing and Metadata Tools

21.1 CSW Clients

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

21.2 CSW Servers

- degree
- eXcat
- GeoNetwork opensource

21.3 Metadata Editing Tools

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

22.1 Community

Please see the Community page for information on the pycsw community, getting support, and how to get involved.
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, pycsw community is a great place to get an idea of how to connect and participate in pycsw community and development.

23.1 GitHub

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

23.2 Code Overview

• the pycsw wiki documents an overview of the codebase

23.3 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.

23.4 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.

23.5 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.
23.6 Development

23.6.1 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 Tester tests before they are committed
- enhancements which can be demonstrated from the pycsw Tester should be accompanied by example CSW request XML

23.6.2 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.

23.6.3 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 72-foo
$ git checkout 72-foo
#
# make code/doc changes
```
#
$ git commit -am 'fix xyz (#72-foo)'
$ git push origin 72-foo

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

## 23.7 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 shall be listed in https://github.com/geopython/pycsw/blob/master/COMMITTERS.txt
The MIT License (MIT)

Copyright (c) 2010-2013 Tom Kralidis

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.
## Committers

<table>
<thead>
<tr>
<th>Login(s)</th>
<th>Name</th>
<th>Email / Contact</th>
<th>Area(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tomkralidis</td>
<td>Tom Kralidis</td>
<td>tomkralidis at gmail.com</td>
<td>Overall</td>
</tr>
<tr>
<td>kalxas</td>
<td>Angelos Tzotsos</td>
<td>tzotsos at gmail.com</td>
<td>INSPIRE, APISO profiles, Packaging</td>
</tr>
<tr>
<td>adamhinz</td>
<td>Adam Hinz</td>
<td>hinz dot adam at gmail.com</td>
<td>WSGI/Server Deployment</td>
</tr>
</tbody>
</table>