pycsw Workshop

version 1.4

Jeff McKenna, Tom Kralidis, Angelos Tzotsos

June 13, 2013

Contents

pycsw Workshop

Welcome to the pycsw workshop. This workshop is a hands-on workshop that will give you an introduction to the popular pycsw metadata publishing software.

Note

The workshop instructions are also available as a single PDF document 

Background

Home

All workshop materials are maintained openly through a GitHub repository: https://github.com/geopython/pycsw-workshop. Contributions are always welcome.

The canonical location of the live workshop is always at http://geopython.github.io/pycsw-workshop/.

History

Initial workshop structure was created by Jeff McKenna of Gateway Geomatics.

Acknowledgements

The initial pycsw workshop materials were created through funding provided by the Oregon Coastal Management Program, through an FGDC CAP grant, in 2013.

Assumptions

As this workshop is designed for use with the OSGeo-Live virtual machine, basic knowledge of Unix commands is assumed.

License

Workshop Requirements

The pycsw workshop requires the following installed locally:

- 7-Zip (ability to extract *.7z* files)

- VirtualBox (virtual machine software, ability to load virtual disk *.vmdk files)
- OSGeo-Live Virtual Machine (which contains pycsw)

> ***Note***
>
> We recommend using the OSGeo-Live Virtual Machine method, although OSGeo-Live is available also through a bootable DVD or USB drive.

1. Install VirtualBox

- download the VirtualBox platform package for your local machine
- run the installer, and select the default setup options (approve any device security questions)

2. Download OSGeo-Live

> ***Caution!***
>
> You'll need a minimum of 10GB of free hard disk space, as well as a machine with 2GB of RAM.

Download

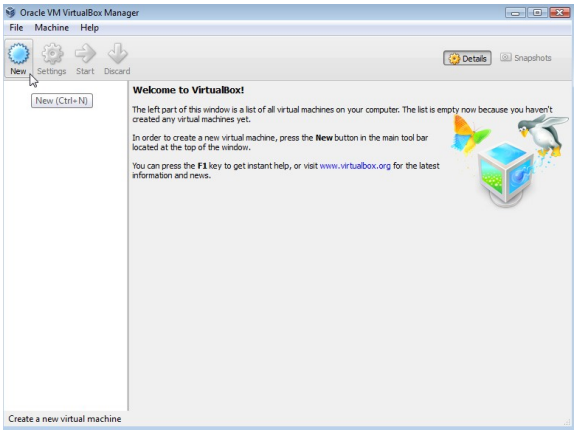
- download the OSGeo-Live Virtual Machine (*.7z) file. It will likely take you ~1 hour to download the 3GB file. There are several sites you can download this from:
  - official site
  - UC Davis mirror
  - National Technical University of Athens mirror

Extract

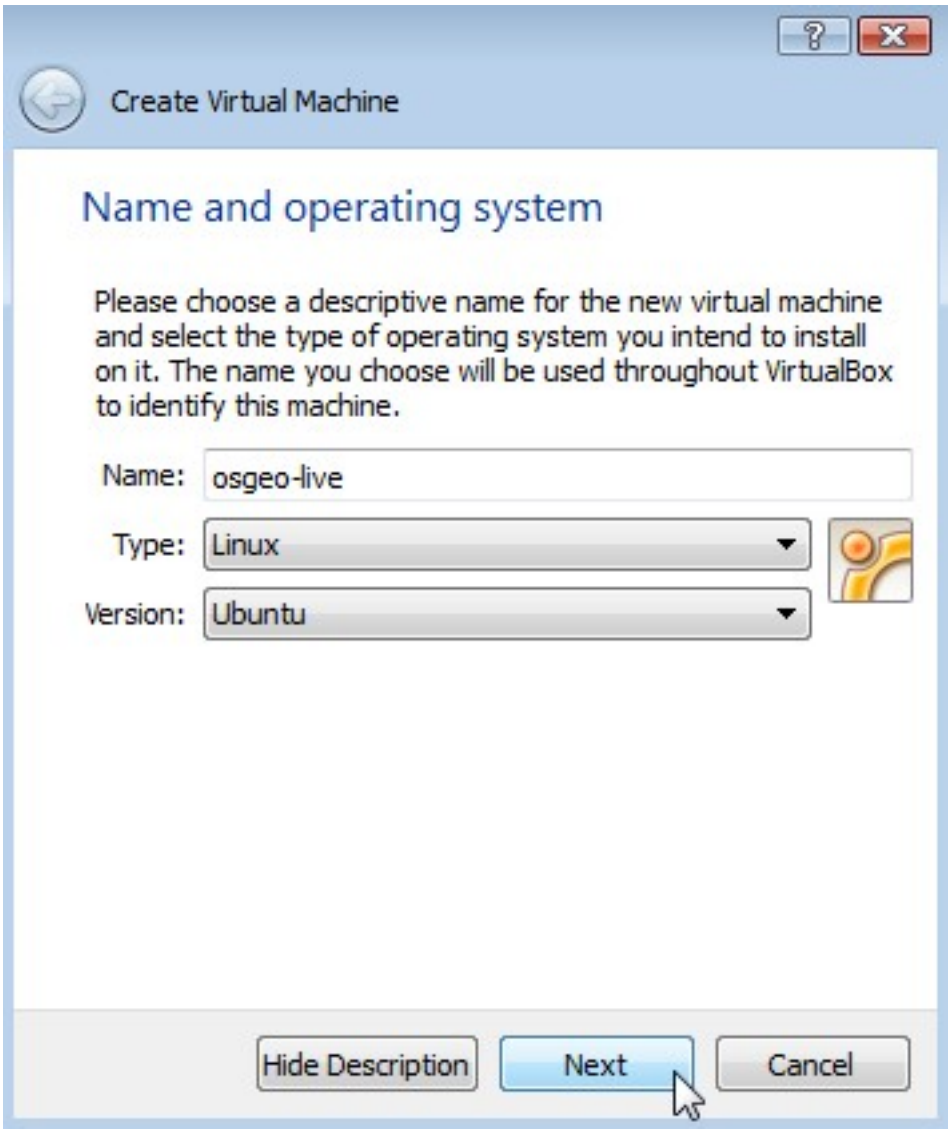
- using 7-Zip, open the *.7z* archive and extract the *.vmdk* file onto your hard disk (the extracted file is ~10GB in size)

3. Create Virtual Machine

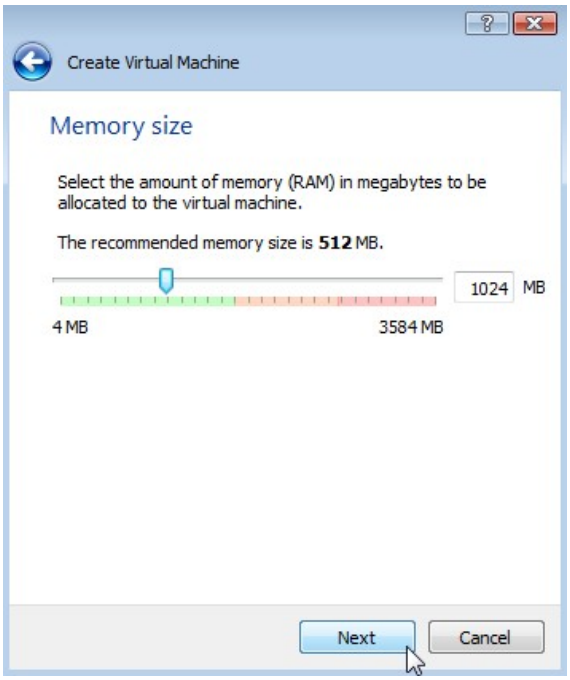
- start VirtualBox ("Oracle VM VirtualBox)
- click on the *New* button to create a VM



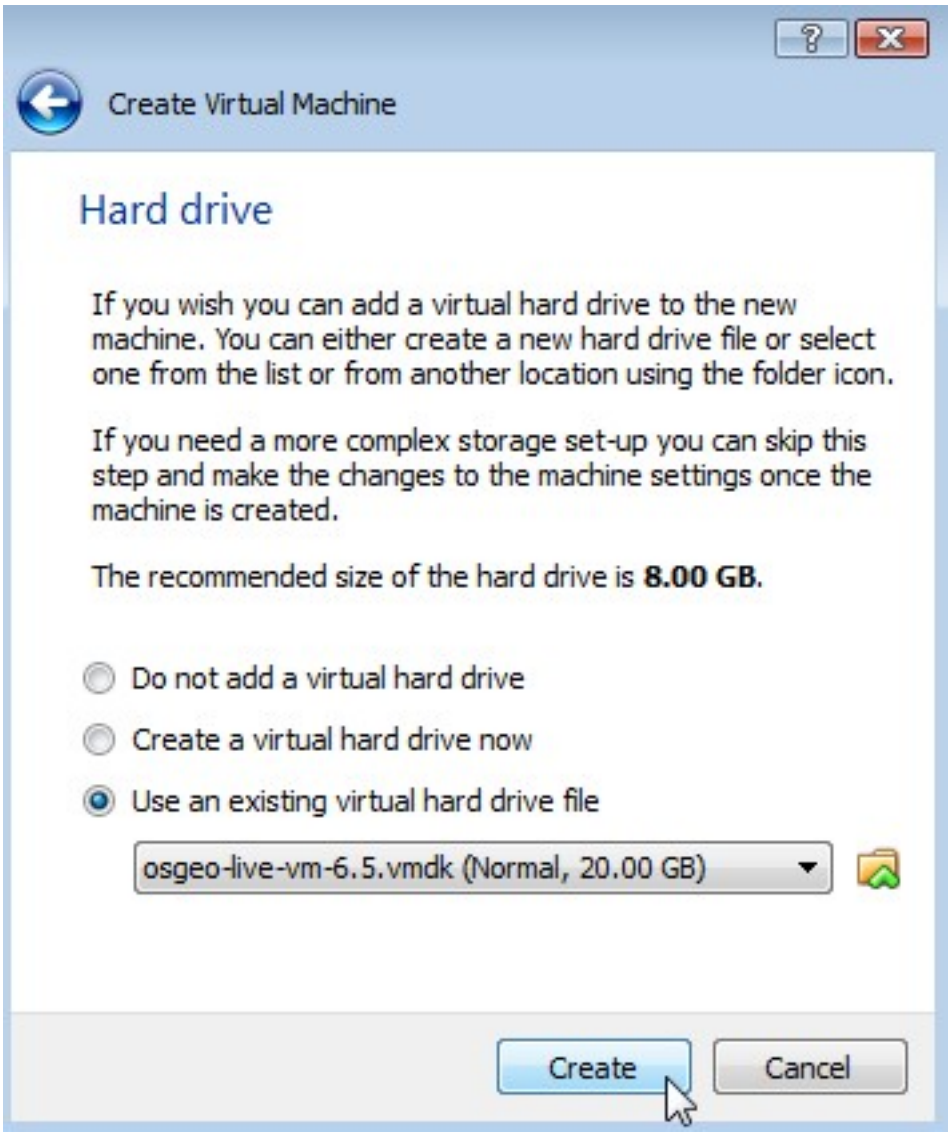
- enter "osgeo-live" for the name, and select *Type: Linux* and *Version: Ubuntu*



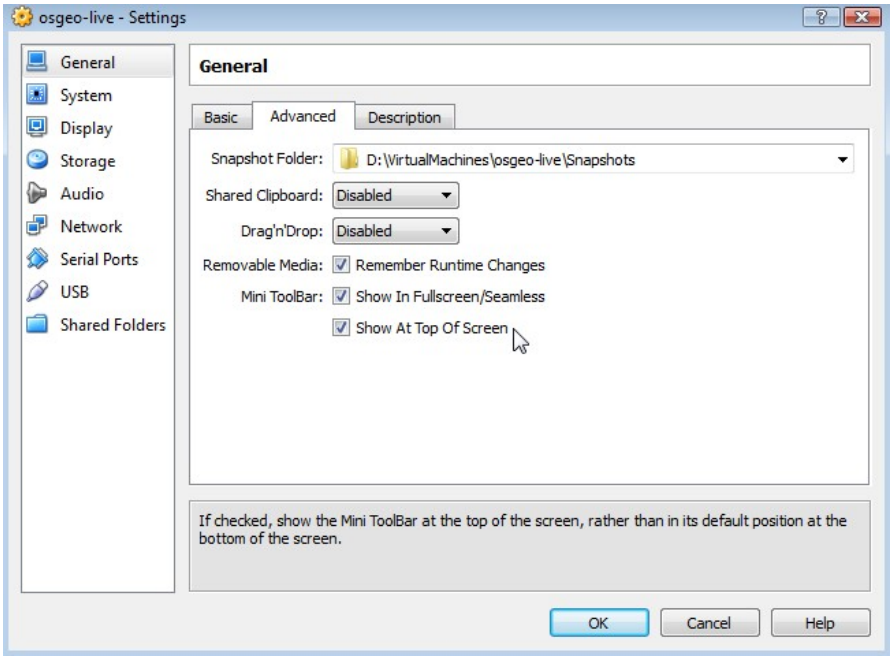
- In the next screen set the memory to 1024 MB (or more if your host computer has more than 4GB).



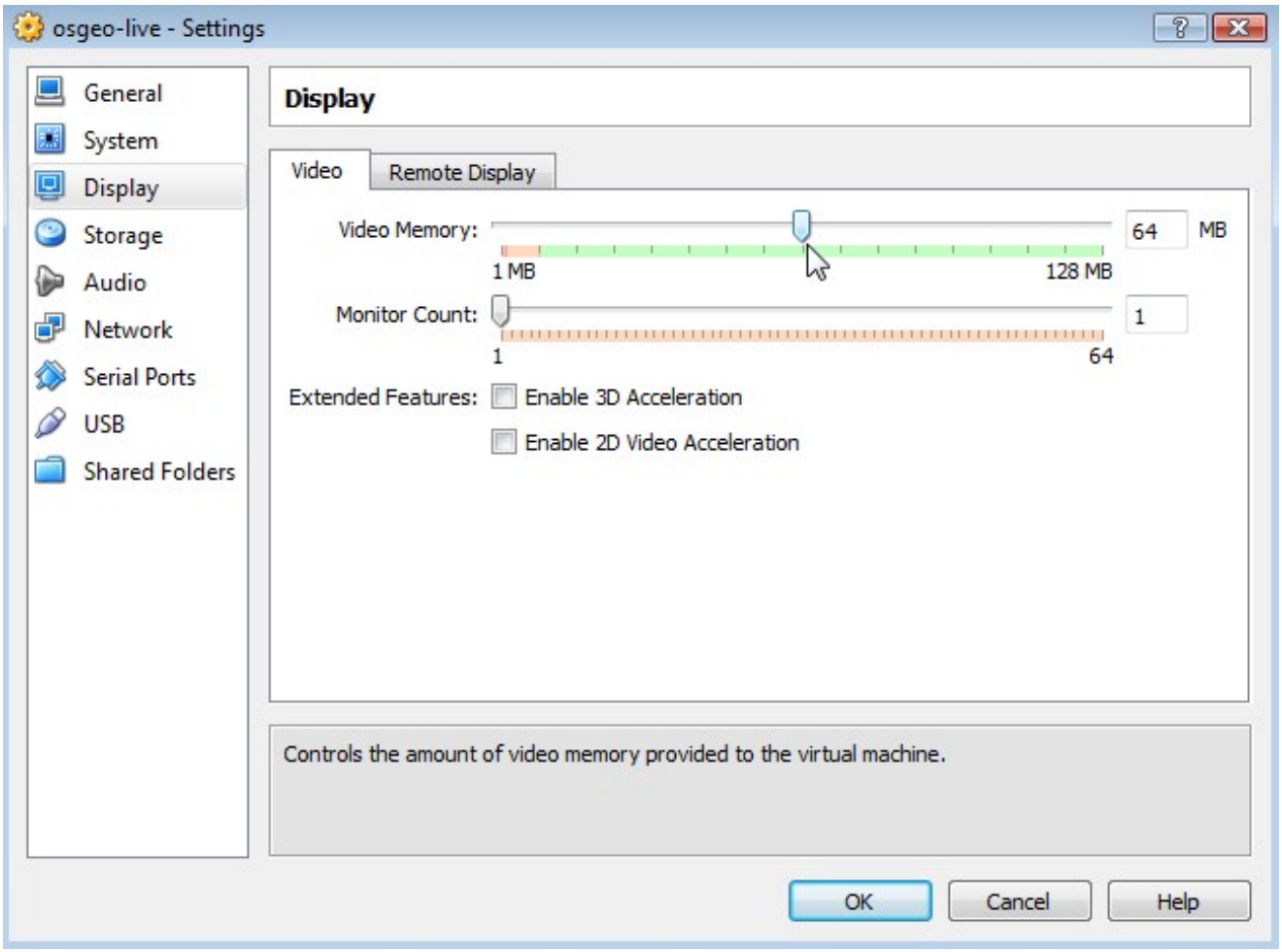
- Continue to the next screen and choose "Use existing hard disk". Then click on the button (a folder icon) to browse to where you saved the *.vmdk file. Select this file, press *Next* and *Create*.



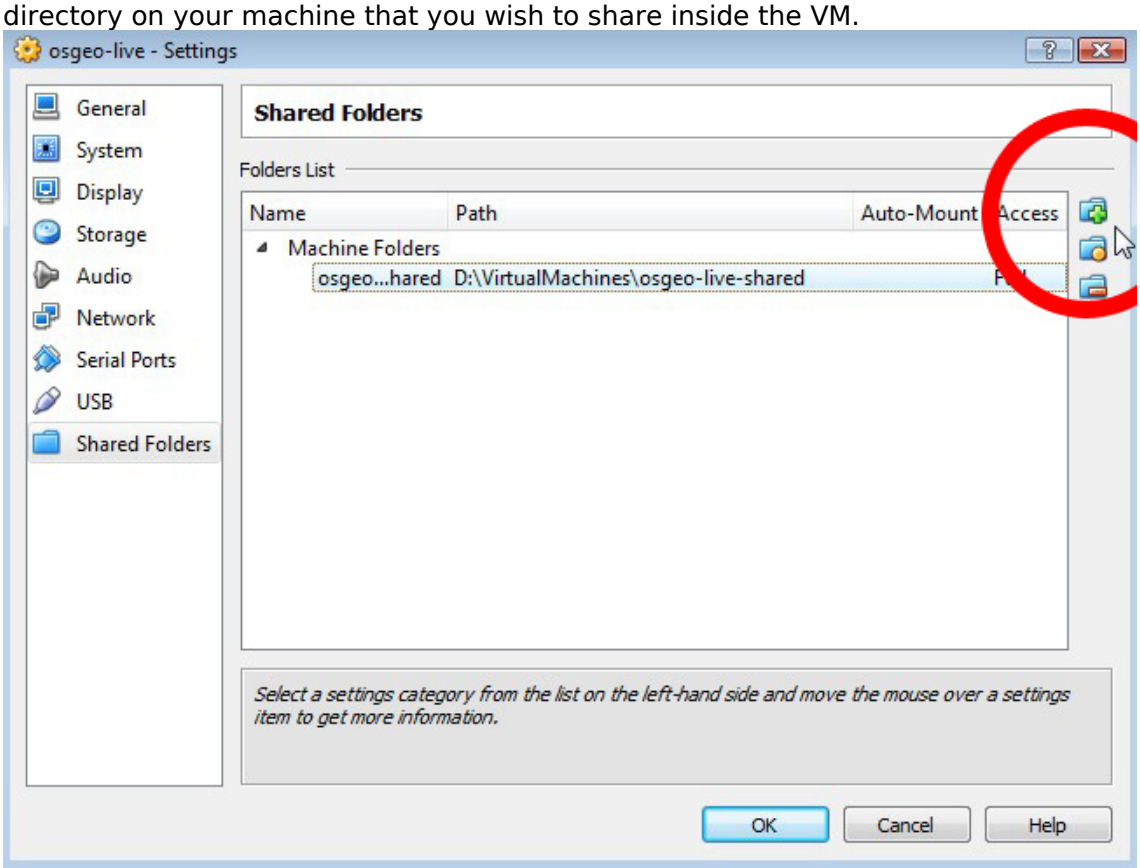
- Once the VM is created, click on the Settings button. In the "General" section, go to the Advanced tab, and click to select "Show at top of screen" for the Mini toolbar.



- In the "Display" section and increase video memory to 32 or 64 MB.



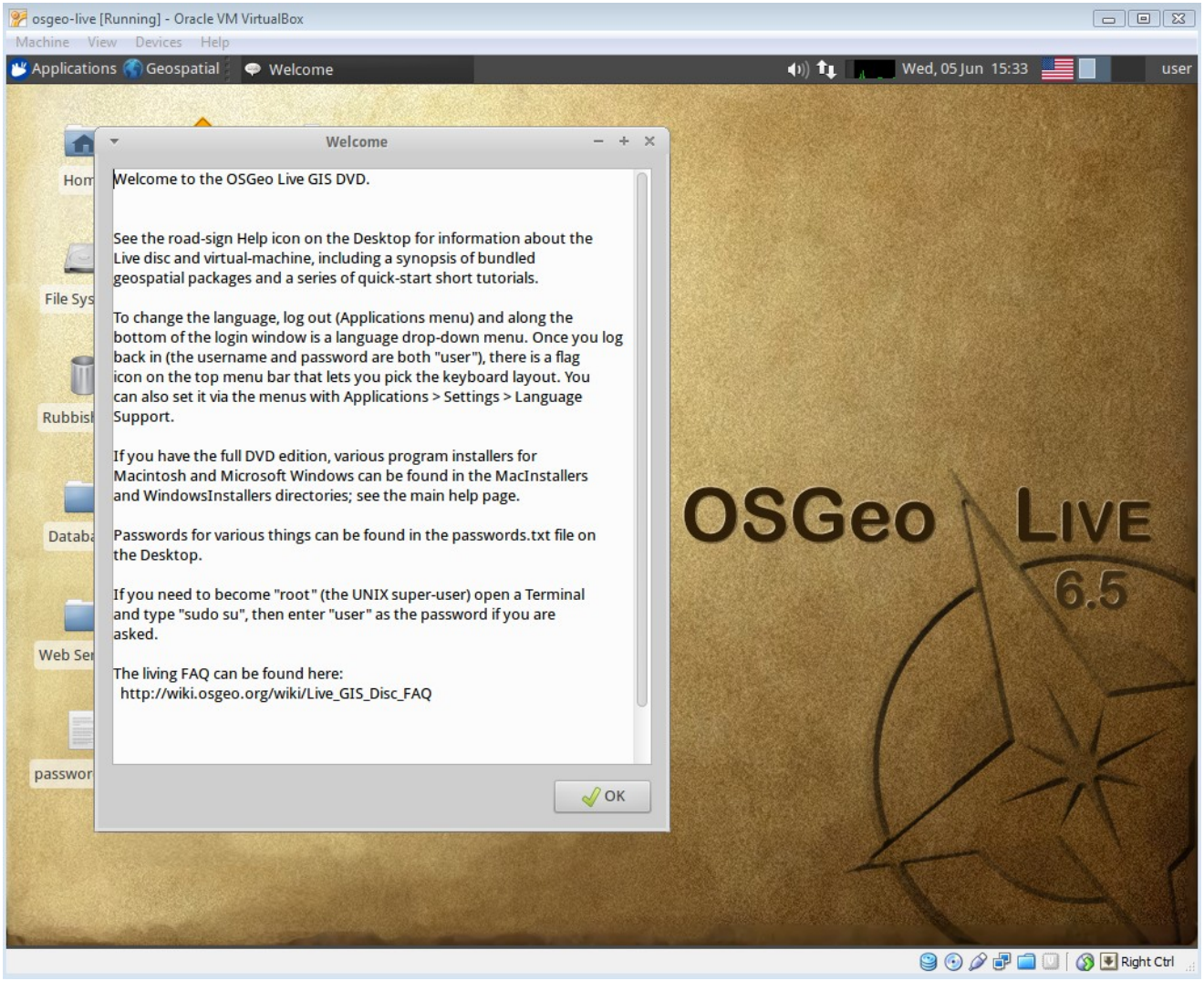
- In the "Shared Folders" section, click the "Add folder" (green + icon on the right) to find a directory on your machine that you wish to share inside the VM.



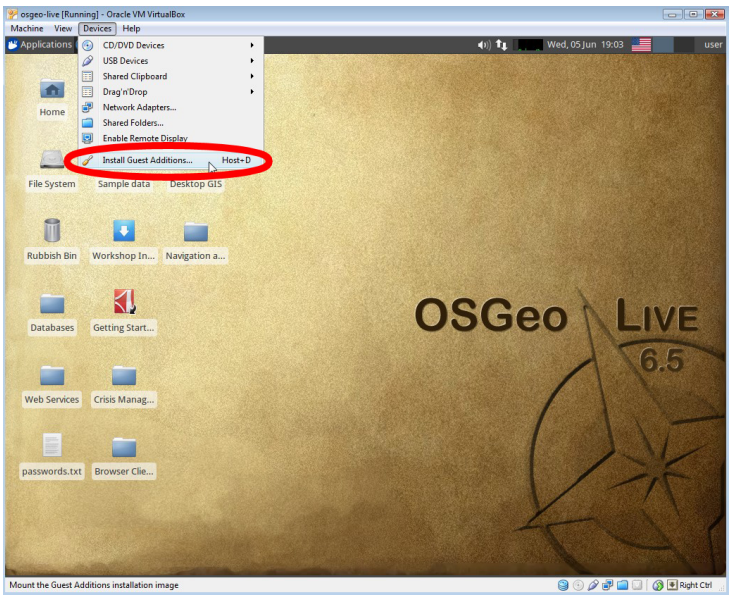
Once the "Folder path" and "Folder name" are defined, click OK, and close the settings window.

4. Run the Virtual Machine

- Now bootup the VM by clicking the Start (green arrow) button. OK any warning messages.



- To improve video performance and enable the shared folders, open the Devices menu and click "Install Guest Additions".



- Next, on the desktop you will see an icon named "VBOXADDITIONS_4.2.12_84980", click it (this mounts the drive). You can then close this window.
- Open a Terminal window (in top left click "Applications" / "Accessories" / "Terminal Emulator"
- In the Terminal, execute the following:

```
user@osgeolive:~$ sudo apt-get install linux-headers-`uname -r`

   password: user

user@osgeolive:~$ cd /media/VBOXADDITIONS_4.2.12_84980

user@osgeolive:/media/VBOXADDITIONS_4.2.12_84980$ sudo ./VBoxLinuxAdditions.run
```

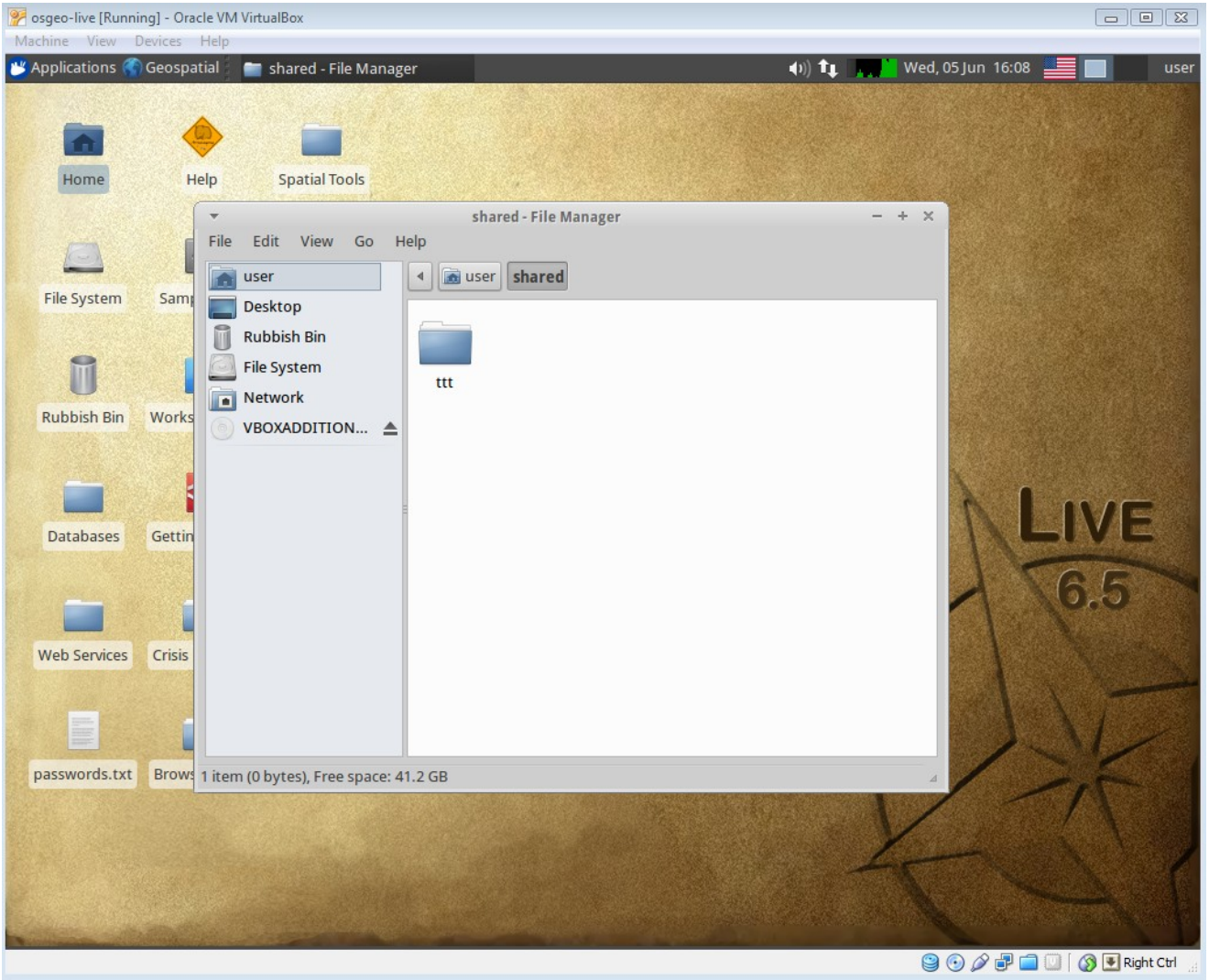
- reboot the machine (click on "user" in top-right of desktop, and select "Reboot")

- Open a Terminal window again, and execute the following (where "osgeo-live-shared" is the name you entered earlier in the Settings for the shared folder):

```
user@osgeolive:~$ mkdir shared

user@osgeolive:~$ sudo mount -t vboxsf -o uid=user,rw osgeo-live-shared /home/user/shared
```

You can now create a test folder on your local machine (in my case "ttt") and then view it within the virtual machine.



Metadata Background

Metadata is often described as "data about data", or the *who*, *what*, *where*, and *when*. In the spatial world, for each dataset we maintain, we should record information about the data such as:

- general description
- location
- usage restrictions
- projection
- technical contact
- date created
- date modified
- version

Benefits

Maintaining metadata for your datasets is important for several reasons:

1. Internal: local management
    - tracking dataset management
    - scheduling data updates
2. External: discovery
    - allowing your dataset to be used outside your organization

Standardized Metadata

With the growth of geographic information systems (GIS) in the 80's and 90's, geographic datasets became a requirement for decision makers across the world. The expansion of the Internet to share information through the late 90's and 2000's has now brought 'discovery' of geographic data into the hands of the average citizen.

Note

Metadata standards have been introduced since the mid-90's with the goals of:

- outlining specific required parameters
- common terminology
- consistency
- interoperability

Dublin Core

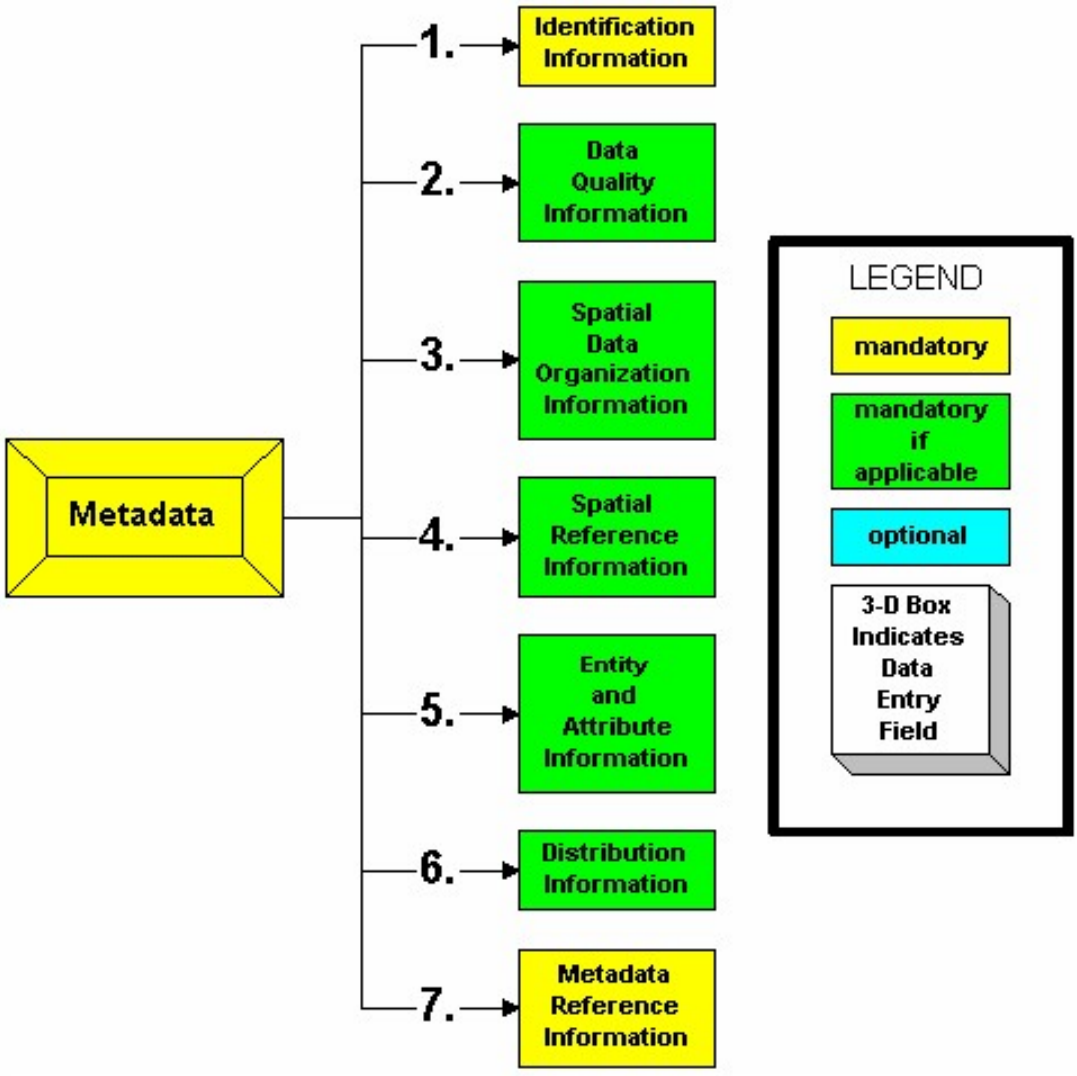
- named from workshop in Dublin, Ohio in 1995
- established a core/common group of 15 metadata elements

Example:

```
<head profile="http://dublincore.org">
..
<meta name="DC.Identifier" schema="DCterms:URI"
content="http://tutorialsonline.info/Common/DublinCore.html" />
<meta name="DC.Format" schema="DCterms:IMT" content="text/html" /> <meta name="DC.Title" xml
<meta name="DC.Creator" content="Alan Kelsey" />
<meta name="DC.Subject" xml:lang="EN" content="Dublin Core Meta Tags" />
<meta name="DC.Publisher" content="Alan Kelsey, Ltd." />
<meta name="DC.Publisher.Address" content="alan@tutorialsonline.info" />
<meta name="DC.Contributor" content="Alan Kelsey" />
<meta name="DC.Date" scheme="ISO8601" content="2007-01-06" />
<meta name="DC.Type" content="text/html" />
<meta name="DC.Description" xml:lang="EN"
  content="Learning Advanced Web Design can be fun and easy! Look at a site designed specifi
<meta name="DC.Identifier" content="http://tutorialsonline.info/Common/DublinCore.html" />
<meta name="DC.Relation" content="TutorialOnline.info" scheme="IsPartOf" />
<meta name="DC.Coverage" content="Hennepin Technical College" />
<meta name="DC.Rights" content="Copyright 2011, Alan Kelsey, Ltd.  All rights reserved." />
<meta name="DC.Date.X-MetadataLastModified" scheme="ISO8601" content="2007-01-06" />
<meta name="DC.Language" scheme="dcterms:RFC1766" content="EN" />
```

FGDC Content Standard for Digital Geospatial Metadata (CSDGM)

- approved by the U.S. Federal Geographic Data Committee originally in 1994
- composed of Sections, Compound Elements, Data Elements



ISO 19115

- International Standards Organization's TC211 committee created this in 2003
- consisting of more than 400 "Core", "Mandatory", and "Optional" elements

Table 3 — Core metadata for geographic datasets

Dataset title (M) (MD_Metadata > MD_DataIdentification.citation > CI_Citation.title)	**Spatial representation type** (O) (MD_Metadata > MD_DataIdentification.spatialRepresentationType)
Dataset reference date (M) (MD_Metadata > MD_DataIdentification.citation > CI_Citation.date)	**Reference system** (O) (MD_Metadata > MD_ReferenceSystem)
Dataset responsible party (O) (MD_Metadata > MD_DataIdentification.pointOfContact > CI_ResponsibleParty)	**Lineage** (O) (MD_Metadata > DQ_DataQuality.lineage > LI_Lineage)
Geographic location of the dataset (by four coordinates or by geographic identifier) (C) (MD_Metadata > MD_DataIdentification.extent > EX_Extent > EX_GeographicExtent > EX_GeographicBoundingBox or EX_GeographicDescription)	**On-line resource** (O) (MD_Metadata > MD_Distribution > MD_DigitalTransferOption.onLine > CI_OnlineResource)
Dataset language (M) (MD_Metadata > MD_DataIdentification.language)	**Metadata file identifier** (O) (MD_Metadata.fileIdentifier)
Dataset character set (C) (MD_Metadata > MD_DataIdentification.characterSet)	**Metadata standard name** (O) (MD_Metadata.metadataStandardName)
Dataset topic category (M) (MD_Metadata > MD_DataIdentification.topicCategory)	**Metadata standard version** (O) (MD_Metadata.metadataStandardVersion)
Spatial resolution of the dataset (O) (MD_Metadata > MD_DataIdentification.spatialResolution > MD_Resolution.equivalentScale or MD_Resolution.distance)	**Metadata language** (C) (MD_Metadata.language)
Abstract describing the dataset (M) (MD_Metadata > MD_DataIdentification.abstract)	**Metadata character set** (C) (MD_Metadata.characterSet)
Distribution format (O) (MD_Metadata > MD_Distribution > MD_Format.name and MD_Format.version)	**Metadata point of contact** (M) (MD_Metadata.contact > CI_ResponsibleParty)
Additional extent information for the dataset (vertical and temporal) (O) (MD_Metadata > MD_DataIdentification.extent > EX_Extent > EX_TemporalExtent or EX_VerticalExtent)	**Metadata date stamp** (M) (MD_Metadata.dateStamp)

ISO 19139

- the XML implementation schema for ISO 19115 specifying the metadata record format
- may be used to describe, validate, and exchange geospatial metadata prepared in XML

OGC CSW Specification

The Open Geospatial Consortium (OGC) OpenGIS Catalog Service Implementation Specification, currently at version 2.0.2, is a standard for discovering and retrieving spatial data and metadata. Catalog Services for the Web (CSW) is a profile/part of the Catalog Service Implementation Specificaton that allows for publishing and searching of metadata.

Operations

```
−<ows:OperationsMetadata>
  +<ows:Operation name="GetCapabilities"></ows:Operation>
  +<ows:Operation name="GetRepositoryItem"></ows:Operation>
  +<ows:Operation name="DescribeRecord"></ows:Operation>
  +<ows:Operation name="GetDomain"></ows:Operation>
  +<ows:Operation name="GetRecordById"></ows:Operation>
  +<ows:Operation name="GetRecords"></ows:Operation>
  +<ows:Parameter name="version"></ows:Parameter>
  +<ows:Parameter name="service"></ows:Parameter>
  +<ows:Constraint name="XPathQueryables"></ows:Constraint>
  +<ows:Constraint name="PostEncoding"></ows:Constraint>
  +<inspire_ds:ExtendedCapabilities xsi:schemaLocation="http://inspire.ec.europa.eu/
    </inspire_ds:ExtendedCapabilities>
</ows:OperationsMetadata>
```

CSW defines several possible operations to discover and retrieve metadata, and groups these operations into 3 "classes":

Service Class

- GetCapabilities (mandatory) - allow clients to retrieve information describing the service instance

Discovery Class

- DescribeRecord (mandatory) - allows a client to discover elements of the information model supported by the target catalog service
- GetRecords (mandatory) - get metadata records
- GetRecordById (optional) - get metadata records by ID
- GetDomain (optional) - obtain runtime information about the range of values of a metadata record element or request parameter.

Management Class

- Harvest (optional) - references the data to be inserted or updated in the catalog
- Transaction (optional) - defines an interface for creating, modifying and deleting catalog records.

Example Live Requests

- GetCapabilities
- DescribeRecord
- GetRecords
- GetRecordById
- GetDomain
- Harvest
- Transaction

Introduction to pycsw

pycsw is a lightweight metadata publisher, written in Python. It is easily configured, and can plug into your architecture.

The following sections will introduce your to the powers of pycsw.

Goals

Initially conceived in 2010, the overal vision of the development team was to:

1. **Create an Open Source standalone metadata publisher in Python**

Many other metadata publishing options exist, but mostly in Java. Python is an Open Source scripting language, that is supported on all major platforms, and is very popular in the geospatial world today.

2. **Make it lightweight and easy to configure: focused on one task, publishing**

   Keep the goals of the project to simple metadata publishing; don't get into other tasks such as metadata editing and acquisition. Rather than a bloated "kitchen sink" concept. limit the software to easy metadata publishing.

3. **Design so additional metadata formats can be easily supported.**

   Although initially the core metadata model was Dublic Core, design the sofware so that many other metadata formats can be plugged in.

Note

The initial pycsw 'manifesto' is still available here. It is a nice way to understand the project's vision.

Features / Design

pycsw is not a metadata editor.

pycsw is a fully compliant CSW server.

pycsw is flexible and headless; can be integrated into workflows without getting in the way, as well as seamless integration with Python environments.

pycsw is able to pull and store ("harvest") layer information from other remote OGC services (WMS, WFS, WCS, WPS, WAF, CSW, SOS).

pycsw avoids any stylesheet conversions by storing metadata elements in a local database.

pycsw allows for remote updates to the local repository ("transactions"), through CSW-T.

pycsw allows for additional profiles/metadata formats to be plugged in.

pycsw includes a commandline utility to administer the metadata repository.

.

Note

A detailed list of features and standards is maintained in the pycsw documentation here.

Component Architecture

Use Cases

These are common situations where pycsw thrives:

Case 1: Publishing against established metadata management workflow

- existing desktop GIS environment
- already utilizing custom environments / processes / automation
- implement pycsw on top of environment to publish
- **Examples: ArcGIS, GeoNode**

Case 2: Publishing against existing metadata database

- bind pycsw to database with property -> column mappings

Installing

Required Software

pycsw
: Source home is on GitHub.

Python
: Core language of pycsw. Both 3.x and 2.x series are supported.

Database
: Used to store metadata elements. (PostgreSQL/PostGIS, SQLite, MySQL)

SQLAlchemy
: Used to bind database models to Python classes.

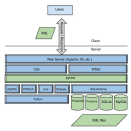
OWSLib
: Used to parse XML formats.

lxml
: Used to parse requests.

pyproj
: Used to handle coordinate transformations.

Shapely
: Used for spatial operations.

Software Architecture



OSGeo Live DVD

The 4 minute install

```
$ virtualenv pycsw && cd pycsw && . bin/activate
$ git clone git@github.com:geopython/pycsw.git && cd pycsw
$ pip install -e . && pip install -r requirements.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
```

Windows

Note

The following steps were performed on a Windows Vista machine.

- Install Python:

    - in this case Python was already installed locally (2.6 series)
    - http://python.org/download/
- Install Package Management Tools
    - download http://python-distribute.org/distribute_setup.py
    - C:\Python26\python.exe distribute_setup.py
- Install Virtual Environment
    - highly recommended
    - read more on its usage: https://pypi.python.org/pypi/virtualenv
    - C:\Python26\Scripts\easy_install.exe virtualenv
        - Create virtual environment
            - C:\Python26\Scripts\virtualenv.exe pycsw-workshop
            - cd pycsw-workshop
            - scripts\activate.exe
- Install pycsw Dependencies
    - C:\Python26\Scripts\easy_install.exe lxml==2.2.8
    - C:\Python26\Scripts\easy_install.exe pyproj==1.9.2
    - C:\Python26\Scripts\easy_install.exe Shapely==1.2.17
    - C:\Python26\Scripts\easy_install.exe SQLAlchemy>=0.6
    - C:\Python26\Scripts\easy_install.exe OWSLib==0.7.1

sw

oad pycsw from http://pycsw.org/download.html and unzip into pycsw-workshop/ folder

csw-1.4.1

default-sample.cfg default.cfg

efault-sample.cfg

```
ome (e.g. home=C:/Python26/Scripts/pycsw-workshop/pycsw-1.4.1)
rl to url=http://localhost:8000/
ry.database (e.g. database=sqlite:///C:\\Python26\\Scripts\\pycsw-workshop\\pycsw-1.4.1\\data\\cite
```

re on the *database*= syntax for PostgreSQL here, SQLite here, or MySQL here.

- Start CSW server
    - python.exe csw.wsgi
    - open web browser to `http://localhost:8000/?service=CSW&version=2.0.2&request=GetCapabilities`

Configuring

- copy `default-sample.cfg` to `default.cfg`
- key configuration parameters
    - `server.home`: absolute path of where pycsw is located
    - `server.url`: the public base URL of the service, which is advertised in Capabilities XML

- `manager.transactions`: `true` or `false` (more on CSW-T later)
- `metadata:main`: service metadata!
- `repository.database`: database connection string

> ***Note***
>
> Read more on the *database*= syntax for PostgreSQL here, SQLite here, or MySQL here.

Exercises

Install pycsw

If you are using OSGeoLive, pycsw is already included. To check your current version:

```
$ sudo apt-cache show python-pycsw
```

To install pycsw on a fresh Ubuntu installation:

```
$ sudo apt-get install python-pycsw pycsw-cgi
```

This will also install dependencies (lxml, Shapely, pyproj, SQLAlchemy, OWSLib)

Upgrade pycsw

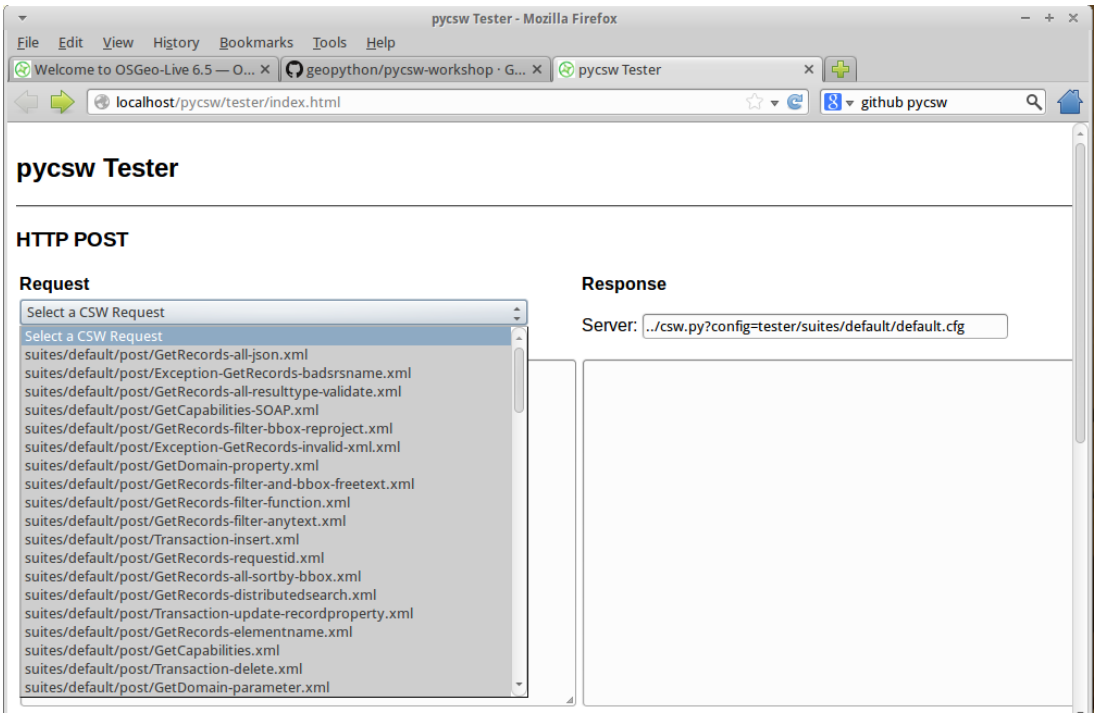
In order to upgrade pycsw to current stable version:

```
$ sudo apt-get upgrade python-pycsw pycsw-cgi
```

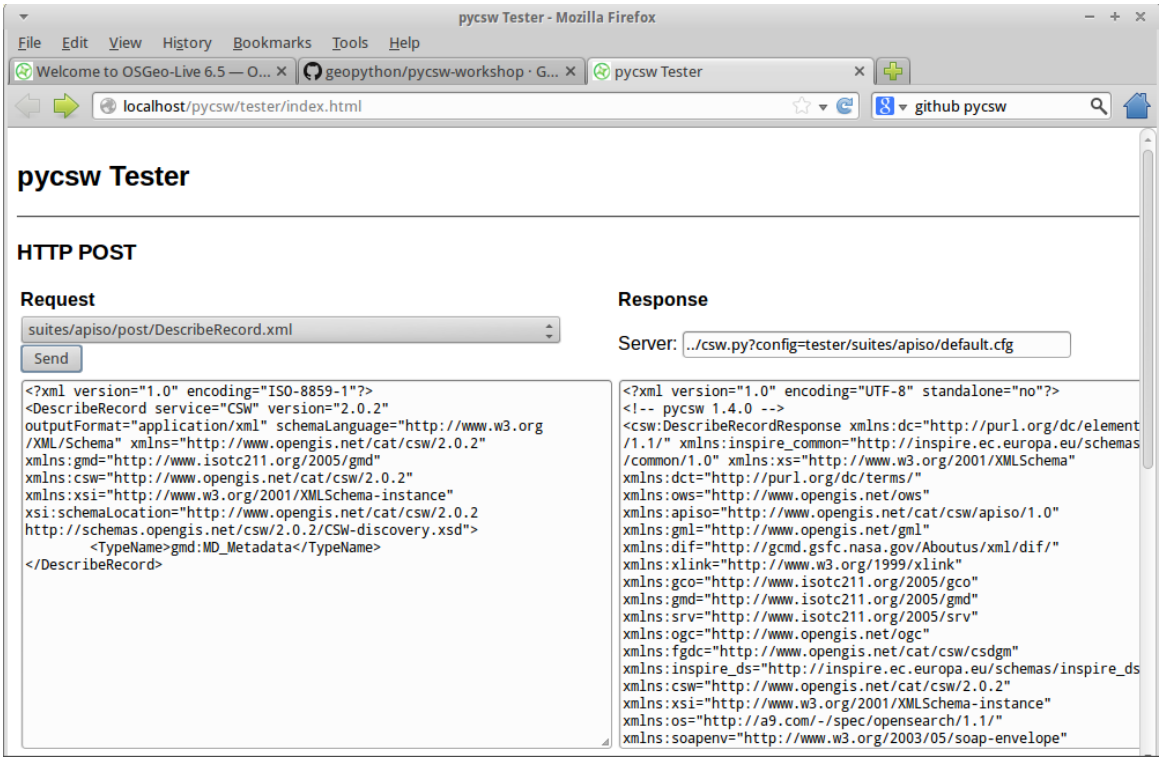
Tester Application

To run the pycsw tester, use the pycsw launcher from the Web Services group, or open Firefox and navigate to `http://localhost/pycsw/tester/index.html`

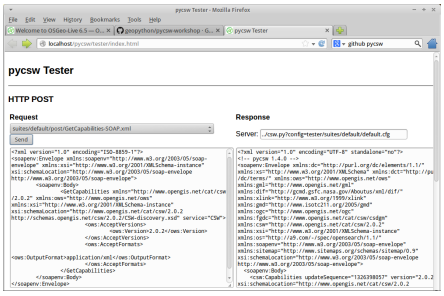
By selecting the left drop-down list, the user can see various predefined POST requests, encoded as XML, that can be sent to pycsw:



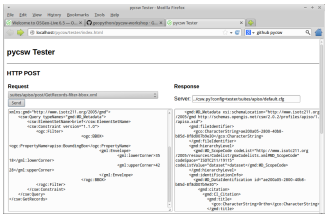
For example, by selecting "apiso/DescribeRecord", and pushing the "Send" button, a description of the ISO Application Profile record is presented on the right panel:



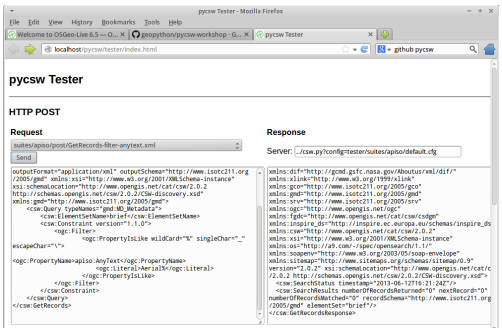
By selecting "GetCapabilities-SOAP" and pushing the "Send" button, a SOAP request is sent to the server to advertise their web service capabilities:



Also, the user can search for data records, performing a spatial bounding box query, by selecting "GetRecords-filter-bbox" and editing the coordinates in the XML request:



By selecting "GetRecords-filter-anytext" and pushing the "Send" button, a full text search request is sent to the server:



The user can go through all the available requests and perform various requests from this testing application.

Capabilities Document

The capabilities of the pycsw installation can be found at `http://localhost/pycsw/csw.py?service=CSW&version=2.0.2&request=GetCapabilities`.

Setup service metadata

pycsw's runtime configuration is defined by `default.cfg`. pycsw ships with a sample configuration (`default-sample.cfg`).

To edit the web service metadata, included in the capabilities document, the user can modify the file `/var/www/pycsw/default.cfg` under the tag `[metadata:main]`.

[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`

Create new database

pycsw supports the following databases:

- SQLite3
- PostgreSQL
- PostgreSQL with PostGIS enabled

- MySQL

In order to create a new SQLite database we need to:

1. Edit `default.cfg`:

[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`)

2. Setup the database:

```
$ cd /var/www/pycsw
$ export PYTHONPATH=`pwd`
$ python ./sbin/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***
>
> Don't forget to test the configuration by sending a GetCapabilities request to the pycsw server.

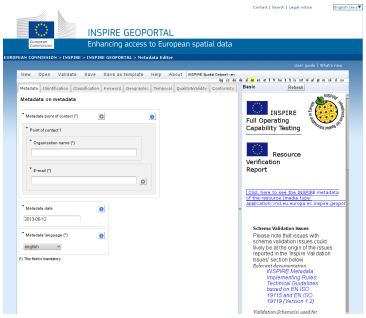
Load demo metadata

We will use some demo GIS data from the GeoNode project. The following will download the data archive and load all metadata xml files into our new database.

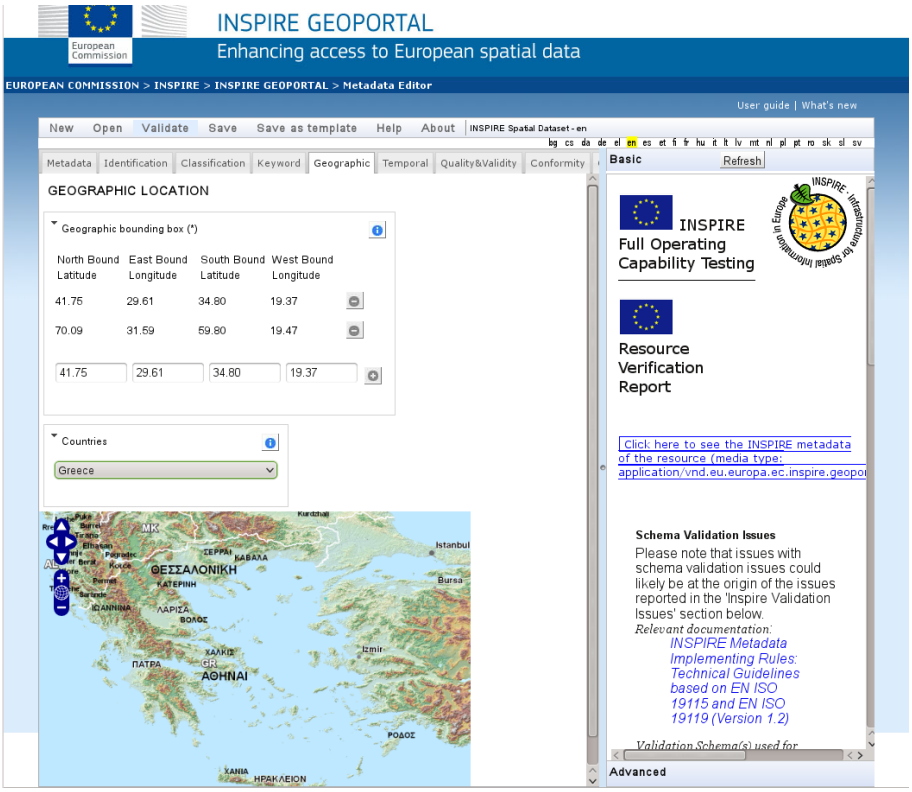
```
$ cd ~
$ wget https://github.com/GeoNode/gisdata/archive/master.zip
$ unzip master.zip
$ cd /var/www/pycsw
$ sudo ./sbin/pycsw-admin.py -c load_records -f default.cfg -p ~/gisdata-master/gisdata/meta
```

Metadata Creation

Metadata files for a spatial dataset or a spatial web service can be created with a plain XML editor. Usually a specialized application is used to create metadata XML files, e.g. through the open source implementation of inspire directive for metadata, European Open Source Metadata Editor (EUOSME). This application can be found at `http://inspire-geoportal.ec.europa.eu/editor/`. Source code is available at `https://joinup.ec.europa.eu/svn/euosme/trunk`

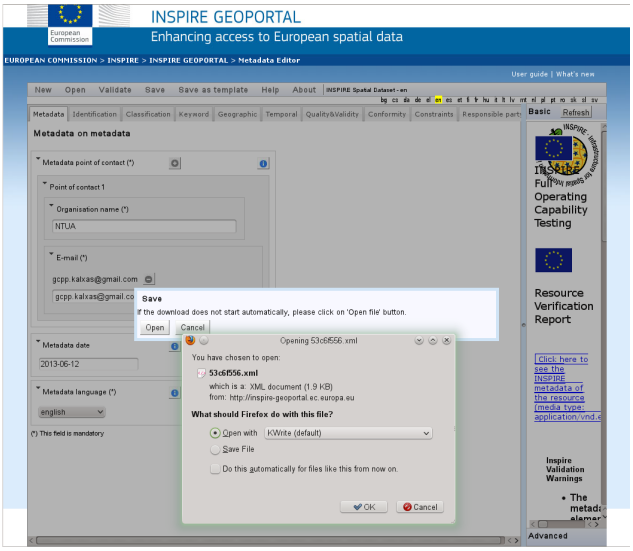


The user fills the mandatory metadata fields going through the application tabs, adding information like the name of the data owner, keywords, resource location on the web, geographic location (using a bounding box or the name of a country) etc.



After the user has added all the information available, must push the validation button on top of the page, so that a check to be performed for consistency with the INSPIRE directive.

After a sucessful validation, the XML file can be saved to the local disk and viewed through a text editor or a browser.



Metadata editing alternatives include:

- CatMDEdit
- GIMED

> ***Note***
>
> Try to create new XML files and then try to load them into the pycsw server as shown previously.

Metadata Harvesting

We can harvest a working WMS service with the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<Harvest xmlns="http://www.opengis.net/cat/csw/2.0.2" xmlns:xsi="http://www.w3.org/2001/XMLS
  <Source>http://webservices.nationalatlas.gov/wms/1million</Source>
  <ResourceType>http://www.opengis.net/wms</ResourceType>
  <ResourceFormat>application/xml</ResourceFormat>
</Harvest>
```

Save the file to something like "post.xml" and run:

```
$ cd /var/www/pycsw
$ python ./sbin/pycsw-admin.py -c post_xml -u http://localhost/pycsw/csw.py -x ~/post.xml
```

Advanced pycsw

Contents:

CSW-T

- CSW class for management functions
- write access to repository
- insert, update, delete
- CSW has no authentication mechanism; left up to provider
- push (`Transaction`) or pull (`Harvest`) operations

Warning

Enabling CSW-T opens up for write access of your data via HTTP

Enabling CSW-T in pycsw

- set `manager.transactions` to `true`
- pycsw uses IP authentication
- limit read/write access to list of allowed IP addresses
  - set in configuration (`manager.allowed_ips`)
    - IP address like `192.168.0.11`
    - wildcards like `192.168.0.*`
    - CIDR like `192.168.100.0/24`

Transactions

- 'push' metadata into repository
- pycsw does no application level backup/versioning

> ***Note***
>
> Always optimize your database accordingly (`pycsw-admin.py -c optimize_db`, `VACUUM ANALYZE`, etc.) when making changes to the repository

Insert

- inserts a record into the repository
  - like `insert into table values (...);`
- pycsw honours the identifier in the metadata
  - if absent, pycsw sets identifier

- example: https://raw.github.com/geopython/pycsw/master/tests/suites/manager/post/Transaction-dc-01-insert.xml

> ***Note***
>
> Always ensure your metadata has an identifier

Update (full)

- full update of metadata record
- will update based on identifier found in XML
  - if no identifier, pycsw will insert as a new record

- example: https://raw.github.com/geopython/pycsw/master/tests/suites/manager/post/Transaction-dc-02-update-full.xml

Update (property)

- partial update of metadata record(s)
  - like `update table set foo="bar"`
- you can apply an OGC filter to make updates on specific records
  - like `update table set foo="bar" where identifier=12345`

- example: https://raw.github.com/geopython/pycsw/master/tests/suites/manager/post/Transaction-iso-03-update-recprop.xml

Delete

- deletes record(s) from the repository
  - like `delete from table where identifier=12345`

- example: https://raw.github.com/geopython/pycsw/master/tests/suites/manager/post/Transaction-iso-05-delete.xml

Harvesting

- 'pull' metadata into repository from remote URL
- pycsw supports many formats for harvesting

- WMS, WFS, WCS, WPS, WAF, SOS
- Dublin Core, FGDC, ISO, RDF
- even other CSW servers

> **Note**
>
> Always optimize your database accordingly (`pycsw-admin.py -c optimize_db`, `VACUUM ANALYZE`, etc.) when making changes to the repository

Exporting

- dump all records in pycsw repository
- use `pycsw-admin.py` to export all records to XML files on disk

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

- creates files in /tmp/metadata
- files are named by metadata record identifier
- want to import them back into another repository?

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

Tips and Tricks

Importing Metadata Recursively

- use the `pycsw-admin.py -r` switch

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

Making CSW XML POST requests

- different from traditional HTTP POST
  - no form key/value pairs
  - client opens HTTP connection and send XML directly

Using `pycsw-admin.py`:

```
$ pycsw-admin.py -c post_xml -u http://labs.gatewaygeomatics.com/csw -x /path/to/request.xml
```

Using curl:

```
$ curl -H "Content-Type: text/xml" -X POST -d @request_file.xml http://labs.gatewaygeomatics
```

JSON Output

- for DescribeRecord, GetRecordById, GetRecords
- set `outputformat` to `application/json` as part of request

Get Raw Metadata

- use `GetRepositoryItem` (based on ebRIM profile)

```
$ GET "http://labs.gatewaygeomatics.com/csw?service=CSW&version=2.0.2&request=GetRepository
```

Optimizing the Repository

```
$ pycsw-admin.py -c optimize_db
```

Dependency Tracing

- use `pycsw-admin.py -c get_sysprof`
- valuable when multiple versions of pycsw and / or supporting libraries are on the same system

Multiple 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`). 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.

Debugging Issues

- turn on logging (set `server.loglevel` to `DEBUG` and `server.logfile` to a writable file)
- set `server.pretty_print` to `true`
- monitor logfile when testing (i.e. `tailf /path/to/pycsw-log.txt`)
- report issues / bugs to pycsw issue tracker / mailing list
- specify environment and supporting libraries (i.e. `pycsw-admin.py -c get_sysprof`)

Community

- in OSGeo Incubation
- OGC Reference Implementation

Exercises

QGIS CSW Client Installation

The HTTP request/response mechanism is not friendly enough to the end user in order to perform queries to the Catalogue Service. For this workshop, we will use the *QGIS <http://www.qgis.org/>* OGC Catalogue Service Client plugin.

To install the plugin in OSGeoLive:

```
$ cd /usr/share/qgis/python/plugins
$ sudo svn co https://qgiscommunitypl.svn.sourceforge.net/svnroot/qgiscommunitypl/python/plu
```

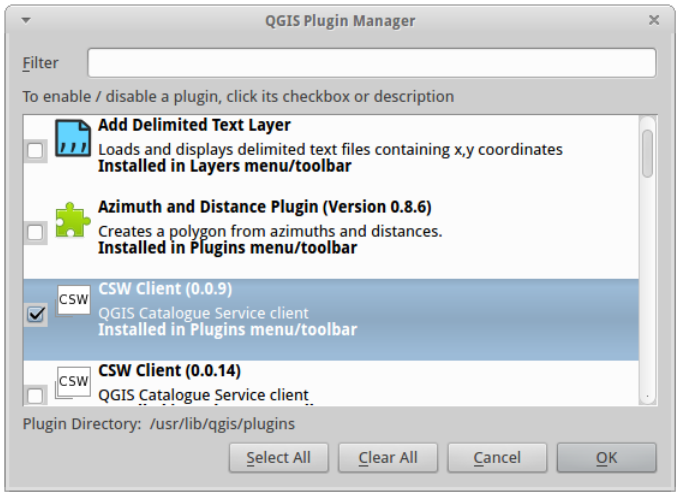
Note

Remember that the root password in OSGeoLive is "user".

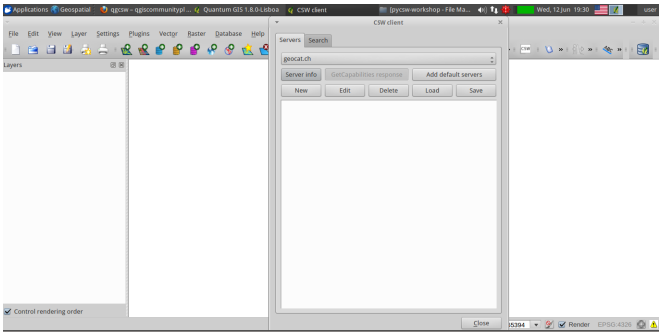
Data Discovery through QGIS

Start QGIS from the Desktop GIS group and go to "Manage Plugins".

Enable the CSW plugin (version 0.0.9) from the list



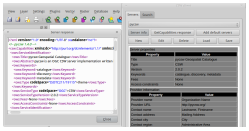
Then select the CSW button from the toolbar and launch CSW Client



Add the pycsw server by pressing the "New" button and type in `http://localhost/pycsw/csw.py`

The user can add some default servers using the "Add default servers" button and also get the capabilities of the server using "Server info" button

Raw server responces are available through the "GetCapabilities" button.

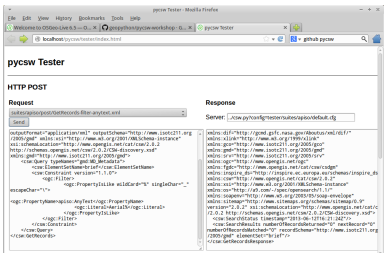


Note

The button name is not correct right now. It will show the last server responce even if it is not a GetCapabilities responce.

Perform search using the catalogue, either by a string value or with a bounding box

Discovery of data can be also performed through the Tester application by setting the appropriate requests eg. any text search with the string "imagery" leads to discovering XML metadata including the word "imagery".

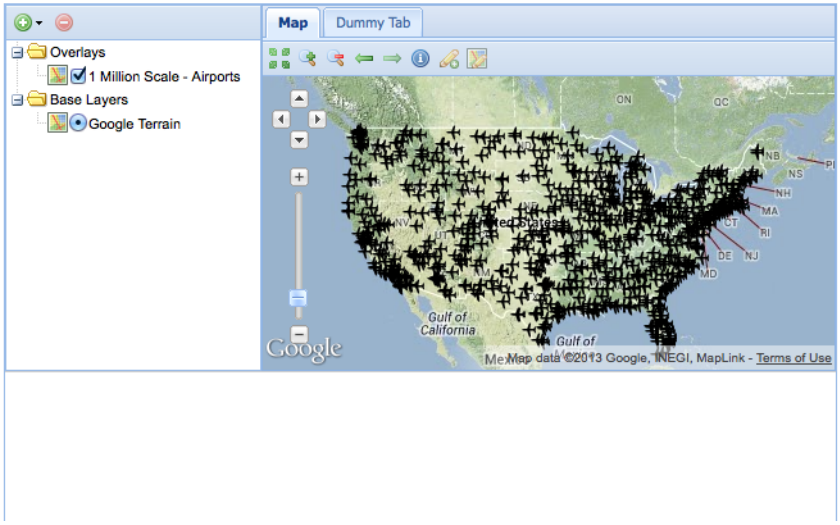


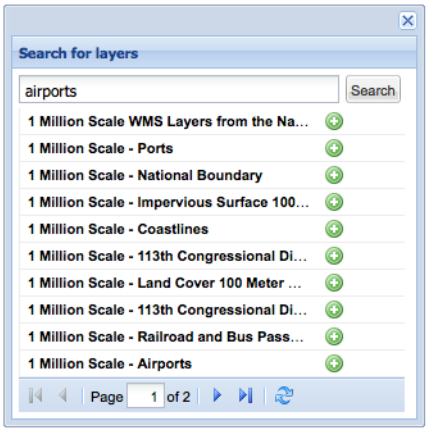
Data Discovery through GeoExt

Another way to use a pycsw server is through a web application, acting like a CSW client. Such functionality is available through OpenLayers and GeoExt Javascript libraries.

For this workshop we have hacked a small demonstration in GeoExt (thanks Bart van den Eijnden) using a demo pycsw installation at http://demo.pycsw.org/services/csw:

- Go to http://demo.pycsw.org/demos/gxp/examples/catalogue.html
- Click icon "find layers"
- Enter "airports" (without double quotes)
- Click "search" or hit Enter
- See results
- Click the "add to map" icon beside the last result on that result set ("1 Million Scale - Airports")
- See layer added to map panel





OWSLib CSW client installation

For this exercise we will use the OWSLib Python library.

OWSLib is a Python package for client programming with Open Geospatial Consortium (OGC) web service (hence OWS) interface standards, and their related content models.

OWSLib is already installed in OSGeoLive as a pycsw dependency.

For installation on another system:

```
$ easy_install OWSLib
```

or in Ubuntu:

```
$ sudo apt-get install python-owslib
```

Data Discovery through OWSLib and Python

Connect to local pycsw server, and inspect its properties:

```
>>> from owslib.csw import CatalogueServiceWeb
>>> csw = CatalogueServiceWeb('http://localhost/pycsw/csw.py')
>>> csw.identification.type
'CSW'
>>> [op.name for op in csw.operations]
['GetCapabilities', 'GetRecords', 'GetRecordById', 'DescribeRecord', 'GetDomain']
```

Get supported resultType's:

```
>>> csw.getdomain('GetRecords.resultType')
>>> csw.results
{'values': ['results', 'validate', 'hits'], 'parameter': 'GetRecords.resultType', 'type': 'c
>>>
```

Search for casino data:

```
>>> csw.getrecords(keywords=['casino'], maxrecords=20)
>>> csw.results
{'matches': 5, 'nextrecord': 0, 'returned': 5}
>>> for rec in csw.records:
...     print csw.records[rec].title
...
CasinoSites
Parcels_North
Parcels
Parcels_East
Parcels_South
>>>
```

Search for casino data in Canada:

```
>>> csw.getrecords(keywords=['casino'],bbox=[-141,42,-52,84])
>>> csw.results
{'matches': 0, 'nextrecord': 0, 'returned': 0}
>>>
```

Search for 'casino' or 'bar'

```
>>> csw.getrecords(keywords=['casino', 'bar'])
>>> csw.results
{'matches': 11, 'nextrecord': 11, 'returned': 10}
>>>
```

Search for a specific record:

```
>>> csw.getrecordbyid(id=['http://capita.wustl.edu/DataspaceMetadata_ISO/CIRA.VIEWS.dv.xml']
>>> csw.records['http://capita.wustl.edu/DataspaceMetadata_ISO/CIRA.VIEWS.dv.xml'].title
'VIEWS.dv'
```

Search with a CQL query

```
>>> csw.getrecords(cql='csw:AnyText like "%bar%"')
```

Transaction: insert

```
>>> from urllib2 import urlopen
>>> content = urlopen('http://demo.pycsw.org/waf/Clause_10.2.5.3.2_Example03_Full.xml').read
>>> csw.transaction(ttype='insert', typename='csw:Record', record=content)
>>> print csw.response
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<csw:TransactionResponse xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:inspire_common="h
  <csw:TransactionSummary>
    <csw:totalInserted>1</csw:totalInserted>
```

```
    <csw:totalUpdated>0</csw:totalUpdated>
    <csw:totalDeleted>0</csw:totalDeleted>
  </csw:TransactionSummary>
  <csw:InsertResult>
    <csw:BriefRecord>
      <dc:identifier>00180e67-b7cf-40a3-861d-b3a09337b174</dc:identifier>
      <dc:title>Image2000 Product 1 (at1) Multispectral</dc:title>
    </csw:BriefRecord>
  </csw:InsertResult>
</csw:TransactionResponse>
>>> # search
>>> csw.getrecords(keywords=['image2000'])
>>> print csw.response
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<csw:GetRecordsResponse xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:inspire_common="ht
  <csw:SearchStatus timestamp="2013-06-13T08:37:07Z"/>
  <csw:SearchResults nextRecord="0" numberOfRecordsMatched="1" numberOfRecordsReturned="1" r
    <csw:SummaryRecord>
      <dc:identifier>00180e67-b7cf-40a3-861d-b3a09337b174</dc:identifier>
      <dc:title>Image2000 Product 1 (at1) Multispectral</dc:title>
      <dc:type>dataset</dc:type>
      <dc:subject>imagery</dc:subject>
      <dc:subject>baseMaps</dc:subject>
      <dc:subject>earthCover</dc:subject>
      <dc:format>BIL</dc:format>
      <dct:modified>2004-10-04 00:00:00</dct:modified>
      <dct:abstract>IMAGE2000 product 1 individual orthorectified scenes. IMAGE2000 was  pro
    </csw:SummaryRecord>
  </csw:SearchResults>
</csw:GetRecordsResponse>
```

Transaction: update

```
>>> # upload an updated version of that metadata
>>> content2 = content.replace('Image2000', 'footitle')
>>> csw.transaction(ttype='update', record=content2)
>>> print csw.response  # raw
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<csw:TransactionResponse xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:inspire_common="h
  <csw:TransactionSummary>
    <csw:totalInserted>0</csw:totalInserted>
    <csw:totalUpdated>1</csw:totalUpdated>
    <csw:totalDeleted>0</csw:totalDeleted>
  </csw:TransactionSummary>
</csw:TransactionResponse>
>>> print csw.results
>>> {'deleted': 0, 'insertresults': [], 'updated': 1, 'inserted': 0, 'requestid': None}
```

Transaction: delete

```
>>> # delete record we inserted
>>> csw.transaction(ttype='delete', identifier='00180e67-b7cf-40a3-861d-b3a09337b174')
>>> c.results
{'deleted': 1, 'insertresults': [], 'updated': 0, 'inserted': 0, 'requestid': None}
```

Harvest a resource

```
>>> help(csw.harvest)

Help on method harvest in module owslib.csw:

harvest(self, source, resourcetype, resourceformat=None, harvestinterval=None, responsehandl
```

```
    Construct and process a Harvest request

  Parameters
  ----------

    - source: a URI to harvest
    - resourcetype: namespace identifying the type of resource
    - resourceformat: MIME type of the resource
    - harvestinterval: frequency of harvesting, in ISO8601
    - responsehandler: endpoint that CSW should responsd to with response
(END)
>>> csw.harvest('http://webservices.nationalatlas.gov/wms/1million', resourcetype='http://ww
>>> print csw.response
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<csw:HarvestResponse xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:inspire_common="http:
  <csw:TransactionResponse version="2.0.2">
    <csw:TransactionSummary>
      <csw:totalInserted>13</csw:totalInserted>
      <csw:totalUpdated>0</csw:totalUpdated>
      <csw:totalDeleted>0</csw:totalDeleted>
    </csw:TransactionSummary>
    <csw:InsertResult>
      <csw:BriefRecord>
        <dc:identifier>urn:uuid:8ed68189-c33e-43c8-9ac0-59ade5cbf682</dc:identifier>
        <dc:title>1 Million Scale WMS Layers from the National Atlas of the United States</d
      </csw:BriefRecord>
      <csw:BriefRecord>
        <dc:identifier>urn:uuid:8ed68189-c33e-43c8-9ac0-59ade5cbf682-ports1m</dc:identifier>
        <dc:title>1 Million Scale - Ports</dc:title>
      </csw:BriefRecord>
      <csw:BriefRecord>
        <dc:identifier>urn:uuid:8ed68189-c33e-43c8-9ac0-59ade5cbf682-national1m</dc:identifi
        <dc:title>1 Million Scale - National Boundary</dc:title>
      </csw:BriefRecord>
      <csw:BriefRecord>
        <dc:identifier>urn:uuid:8ed68189-c33e-43c8-9ac0-59ade5cbf682-impervious</dc:identifi
        <dc:title>1 Million Scale - Impervious Surface 100 Meter Resolution</dc:title>
      </csw:BriefRecord>
      <csw:BriefRecord>
        <dc:identifier>urn:uuid:8ed68189-c33e-43c8-9ac0-59ade5cbf682-coast1m</dc:identifier>
        <dc:title>1 Million Scale - Coastlines</dc:title>
      </csw:BriefRecord>
      <csw:BriefRecord>
        <dc:identifier>urn:uuid:8ed68189-c33e-43c8-9ac0-59ade5cbf682-cdl</dc:identifier>
        <dc:title>1 Million Scale - 113th Congressional Districts</dc:title>
      </csw:BriefRecord>
      <csw:BriefRecord>
        <dc:identifier>urn:uuid:8ed68189-c33e-43c8-9ac0-59ade5cbf682-landcov100m</dc:identif
        <dc:title>1 Million Scale - Land Cover 100 Meter Resolution</dc:title>
      </csw:BriefRecord>
      <csw:BriefRecord>
        <dc:identifier>urn:uuid:8ed68189-c33e-43c8-9ac0-59ade5cbf682-cdp</dc:identifier>
        <dc:title>1 Million Scale - 113th Congressional Districts by Party</dc:title>
      </csw:BriefRecord>
      <csw:BriefRecord>
        <dc:identifier>urn:uuid:8ed68189-c33e-43c8-9ac0-59ade5cbf682-amtrak1m</dc:identifier
        <dc:title>1 Million Scale - Railroad and Bus Passenger Stations</dc:title>
      </csw:BriefRecord>
      <csw:BriefRecord>
        <dc:identifier>urn:uuid:8ed68189-c33e-43c8-9ac0-59ade5cbf682-airports1m</dc:identifi
```

```
        <dc:title>1 Million Scale - Airports</dc:title>
      </csw:BriefRecord>
      <csw:BriefRecord>
        <dc:identifier>urn:uuid:8ed68189-c33e-43c8-9ac0-59ade5cbf682-one_million</dc:identif
        <dc:title>1 Million Scale WMS Layers from the National Atlas of the United States</d
      </csw:BriefRecord>
      <csw:BriefRecord>
        <dc:identifier>urn:uuid:8ed68189-c33e-43c8-9ac0-59ade5cbf682-states1m</dc:identifier
        <dc:title>1 Million Scale - States</dc:title>
      </csw:BriefRecord>
      <csw:BriefRecord>
        <dc:identifier>urn:uuid:8ed68189-c33e-43c8-9ac0-59ade5cbf682-treecanopy</dc:identifi
        <dc:title>1 Million Scale - Tree Canopy 100 Meter Resolution</dc:title>
      </csw:BriefRecord>
    </csw:InsertResult>
  </csw:TransactionResponse>
</csw:HarvestResponse>
```

pycsw and Open Data

- pycsw is embedded in various Open Data portal software
- metadata editing / management done with portal
- portal exposes CSW service automagically
- easy integration into existing apps/workflows

Contents:

GeoNode

- Open Source Geospatial Content Management System
  - geospatial data / metadata management
  - interactive mapping
  - collaboration
- pycsw enabled out of the box
  - embedded CSW

Open Data Catalog

- Code for America app
- open data publishing
- pycsw enabled out of the box
  - CSW embdedded

pycsw Future Development

1.6.0 (June 2013)

- extended harvesting (WAF, SOS, RDF)
- ISO 19115-2 (gmi) support

- spatial relevance ranking
- enhanced OGC filter support
- flexible administation enhancements

1.8 / 2.0

- CSW 3.0
- OPeNDAP integration via pydap
- THREDDS catalog harvesting
- CKAN integration
- native spatial databases
- backends (GeoCouch)
- PostgreSQL full text search (FTS)
- enhanced harvesting / additional formats / APIs
- search engine libraries
- Open Data metadata JSON summary format