
OWSLib Documentation

Release 0.21.0

Tom Kralidis

2021-12-21

CONTENTS

1	Introduction	3
2	Features	5
2.1	Standards Support	5
2.2	OGC API Support	5
3	Installation	7
3.1	Requirements	7
3.2	Install	7
4	Usage	9
4.1	WMS	9
4.2	WFS	11
4.3	OGC API	12
4.4	WCS	14
4.5	CSW	15
4.6	WMC	17
4.7	WPS	17
4.8	SOS 1.0	21
4.9	SOS 2.0	32
4.10	SensorML	35
4.11	ISO	37
4.12	CRS Handling	38
4.13	Dublin Core	39
4.14	NASA DIF	39
4.15	FGDC	39
4.16	Swiss GM03	39
4.17	WMTS	40
4.18	WaterML	42
4.19	OGC OWS Context 1.0.0 Atom CML and GeoJSON Encoding (alpha/under-review)	46
5	Development	49
5.1	Testing	49
6	Support	51
6.1	Mailing Lists	51
6.2	Submitting Questions to Community	51
6.3	Searching the Archives	51
6.4	Metrics	51
6.5	IRC	51

7	Logging	53
8	License	55
9	Credits	57
10	Changes	59
10.1	0.21.0 (2020-12-09)	59
10.2	0.20.0 (2020-06-05)	59
10.3	0.19.2 (2020-03-13)	59
10.4	0.19.1 (2020-01-29)	60
10.5	0.19.0 (2019-11-14)	60
10.6	0.18.0 (2019-06-24)	61
10.7	0.17.1 (2019-01-12)	61
10.8	0.17.0 (2018-09-04)	61
10.9	0.16.0 (2017-12-21)	62
10.10	0.15.0 (2017-09-13)	62
10.11	0.14.0 (2017-01-12)	63
10.12	0.13.0 (2016-09-24)	63
10.13	0.12.0 (2016-09-12)	63
10.14	0.11.0 (2016-04-01)	64
10.15	0.10.0 (2015-11-11)	64
10.16	0.9.2 (2015-09-23)	64
10.17	0.9.1 (2015-09-03)	64
10.18	0.9.0 (2015-06-12)	65
10.19	0.8.13 (2015-02-12)	65
10.20	0.8.11 (2014-12-17)	65
10.21	0.8.10 (2014-10-13)	65
10.22	0.8.9 (2014-09-24)	66
10.23	0.8.8 (2014-07-05)	66
10.24	0.8.7 (2014-05-02)	66
10.25	0.8.3 (2014-01-10)	67
10.26	0.8.0 (2013-09-11)	67
10.27	0.7 (2013-02-18)	67
10.28	0.6 (2012-12-22)	68
10.29	0.5 (2012-06-15)	68
10.30	0.4 (2011-10-02)	68
10.31	0.3 (2008-05-08)	69
10.32	0.2.1 (2007-08-06)	69
10.33	0.2.0 (2007-02-01)	69
10.34	0.1.0 (2006-10-19)	69
10.35	0.0.1 (2006-07-30)	69

Author Tom Kralidis

Contact tomkralidis at gmail.com

Release 0.21.0

Date 2021-12-21

INTRODUCTION

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 was buried down inside PCL (Python Cartography Library), but has been brought out as a separate project.

FEATURES

2.1 Standards Support

Standard	Version(s)
OGC WMS	1.1.1, 1.3.0
OGC WFS	1.0.0, 1.1.0, 2.0.0, 3.0
OGC WCS	1.0.0, 1.1.0, 2.0, 2.0.1
OGC WMC	1.1.0
OGC SOS	1.0.0, 2.0.0
OGC SensorML	1.0.1
OGC CSW	2.0.2
OGC WPS	1.0.0
OGC Filter	1.1.0
OGC OWS Common	1.0.0, 1.1.0, 2.0
OGC OWS Context	1.0.0 (alpha/under-review)
NASA DIF	9.7
FGDC CSDGM	1998
ISO 19139	2007
Dublin Core	1.1
Swiss GM03	2.3
WMTS	1.0.0
WaterML	1.0, 1.1, 2.0

2.2 OGC API Support

Standard	Version(s)
OGC API - Features - Part 1: Core	1.0
OGC API - Records - Part 1: Core	draft

INSTALLATION

3.1 Requirements

OWSLib requires a Python interpreter, as well as [ElementTree](#) or [lxml](#) for XML parsing.

3.2 Install

PyPI:

```
$ easy_install OWSLib
# pip works too
$ pip install OWSLib
```

Git:

```
$ git clone git://github.com/geopython/OWSLib.git
```

Anaconda:

Note: The OWSLib conda packages are provided by the community, not OSGEO, and therefore there may be multiple packages available. To search all conda channels: <http://anaconda.org/search?q=type%3Aconda+owslib> However usually conda-forge will be the most up-to-date.

```
$ conda install -c conda-forge owslib
```

openSUSE:

```
$ zypper ar http://download.opensuse.org/repositories/Application:/Geo/openSUSE_12.1/ GEO
$ zypper refresh
$ zypper install owslib
```

CentOS:

```
$ wget -O /etc/yum.repos.d/CentOS-CBS.repo http://download.opensuse.org/repositories/
↪Application:/Geo/CentOS_6/Application:Geo.repo
$ yum install owslib
```

RedHat Enterprise Linux

```
$ wget -O /etc/yum.repos.d/RHEL-CBS.repo http://download.opensuse.org/repositories/  
↪Application:/Geo/RHEL_6/Application:Geo.repo  
$ yum install owslib
```

Fedora:

Note: As of Fedora 20, OWSLib is part of the Fedora core package collection

```
$ yum install OWSLib
```

4.1 WMS

Find out what a WMS has to offer. Service metadata:

```
>>> from owslib.wms import WebMapService
>>> wms = WebMapService('http://wms.jpl.nasa.gov/wms.cgi', version='1.1.1')
>>> wms.identification.type
'OGC:WMS'
>>> wms.identification.version
'1.1.1'
>>> wms.identification.title
'JPL Global Imagery Service'
>>> wms.identification.abstract
'WMS Server maintained by JPL, worldwide satellite imagery.'
```

Available layers:

```
>>> list(wms.contents)
['global_mosaic', 'global_mosaic_base', 'us_landsat_wgs84', 'srtm_mag', 'daily_terra_721
↳ ', 'daily_aqua_721', 'daily_terra_ndvi', 'daily_aqua_ndvi', 'daily_terra', 'daily_aqua
↳ ', 'BMNG', 'modis', 'huemapped_srtm', 'srtmplus', 'worldwind_dem', 'us_ned', 'us_
↳ elevation', 'us_colordem']
```

Details of a layer:

```
>>> wms['global_mosaic'].title
'WMS Global Mosaic, pan sharpened'
>>> wms['global_mosaic'].queryable
0
>>> wms['global_mosaic'].opaque
0
>>> wms['global_mosaic'].boundingBox
>>> wms['global_mosaic'].boundingBoxWGS84
(-180.0, -60.0, 180.0, 84.0)
>>> wms['global_mosaic'].crsOptions
['EPSG:4326', 'AUTO:42003']
>>> wms['global_mosaic'].styles
{'pseudo_bright': {'title': 'Pseudo-color image (Uses IR and Visual bands, 542 mapping),
↳ gamma 1.5'}, 'pseudo': {'title': '(default) Pseudo-color image, pan sharpened (Uses IR
↳ and Visual bands, 542 mapping), gamma 1.5'}, 'visual': {'title': 'Real-color image,
↳ pan sharpened (Uses the visual bands, 321 mapping), gamma 1.5'}, 'pseudo_low': {'title':
↳ (continues on next page)
↳ ': 'Pseudo-color image, pan sharpened (Uses IR and Visual bands, 542 mapping)'},
↳ 'visual_low': {'title': 'Real-color image, pan sharpened (Uses the visual bands, 321
↳ mapping)'}, 'visual_bright': {'title': 'Real-color image (Uses the visual bands, 321
↳ mapping), gamma 1.5'}}
9
```

(continued from previous page)

Available methods, their URLs, and available formats:

```
>>> [op.name for op in wms.operations]
['GetCapabilities', 'GetMap']
>>> wms.getOperationByName('GetMap').methods
{'Get': {'url': 'http://wms.jpl.nasa.gov/wms.cgi?'}}
>>> wms.getOperationByName('GetMap').formatOptions
['image/jpeg', 'image/png', 'image/geotiff', 'image/tiff']
```

That's everything needed to make a request for imagery:

```
>>> img = wms.getmap( layers=['global_mosaic'],
...                  styles=['visual_bright'],
...                  srs='EPSG:4326',
...                  bbox=(-112, 36, -106, 41),
...                  size=(300, 250),
...                  format='image/jpeg',
...                  transparent=True
...                  )
>>> out = open('jpl_mosaic_visb.jpg', 'wb')
>>> out.write(img.read())
>>> out.close()
```

Result:



4.2 WFS

Connect to a WFS and inspect its capabilities.

```
>>> from owslib.wfs import WebFeatureService
>>> wfs11 = WebFeatureService(url='http://geoserv.weichand.de:8080/geoserver/wfs',
↳ version='1.1.0')
>>> wfs11.identification.title
'INSPIRE WFS 2.0 DemoServer Verwaltungsgrenzen Bayern'

>>> [operation.name for operation in wfs11.operations]
['GetCapabilities', 'DescribeFeatureType', 'GetFeature', 'GetGmlObject']
```

List FeatureTypes

```
>>> list(wfs11.contents)
['bvv:vg_ex', 'bvv:bayern_ex', 'bvv:lkr_ex', 'bvv:regbez_ex', 'bvv:gmd_ex']
```

Download GML using typename, bbox and srsname.

```
>>> # OWSLib will switch the axis order from EN to NE automatically if designated by
↳ EPSG-Registry
>>> response = wfs11.getfeature(typename='bvv:gmd_ex', bbox=(4500000,5500000,4500500,
↳ 5500500), srsname='urn:x-ogc:def:crs:EPSG:31468')
```

Return a FeatureType's schema via DescribeFeatureType. The dictionary returned is compatible with a Fiona schema object.

```
>>> wfs11.get_schema('bvv:vg_ex')
>>> {'properties': {'land': 'string', 'modellart': 'string', 'objart': 'string', 'objart_
↳ txt': 'string', 'objid': 'string', 'hdu_x': 'short', 'beginn': 'string', 'ende':
↳ 'string', 'adm': 'string', 'avg': 'string', 'bez_gem': 'string', 'bez_krs': 'string',
↳ 'bez_lan': 'string', 'bez_rbz': 'string', 'sch': 'string'}, 'geometry': '3D
↳ MultiPolygon', 'geometry_column': 'geom'}
```

Download GML using typename and filter. OWSLib currently only support filter building for WFS 1.1 (FE.1.1).

```
>>> from owslib.fes import *
>>> from owslib.etree import etree
>>> from owslib.wfs import WebFeatureService
>>> wfs11 = WebFeatureService(url='http://geoserv.weichand.de:8080/geoserver/wfs',
↳ version='1.1.0')

>>> filter = PropertyIsLike(propertyname='bez_gem', literal='Ingolstadt', wildCard='*')
>>> filterxml = etree.tostring(filter.toXML()).decode("utf-8")
>>> response = wfs11.getfeature(typename='bvv:gmd_ex', filter=filterxml)
```

Save response to a file.

```
>>> out = open('/tmp/data.gml', 'wb')
>>> out.write(bytes(response.read(), 'UTF-8'))
>>> out.close()
```

Download GML using StoredQueries(only available for WFS 2.0 services)

```
>>> from owslib.wfs import WebFeatureService
>>> wfs20 = WebFeatureService(url='http://geoserv.weichand.de:8080/geoserver/wfs',
↳ version='2.0.0')

>>> # List StoredQueries
>>> [storedquery.id for storedquery in wfs20.storedqueries]
['bboxQuery', 'urn:ogc:def:query:OGC-WFS::GetFeatureById',
↳ 'GemeindeByGemeindeschluesselEpsg31468', 'DWithinQuery']

>>> # List Parameters for StoredQuery[1]
>>> parameter.name for parameter in wfs20.storedqueries[1].parameters]
['ID']

>>> response = wfs20.getfeature(storedQueryID='urn:ogc:def:query:OGC-WFS::GetFeatureById
↳ ', storedQueryParams={'ID': 'gmd_ex.1'})
```

4.3 OGC API

The **OGC API** standards are a clean break from the traditional OGC service architecture using current design patterns (RESTful, JSON, OpenAPI). As such, OWSLib the code follows the same pattern.

4.3.1 OGC API - Features 1.0

```
>>> from owslib.ogcapi.features import Features
>>> w = Features('https://demo.pygeoapi.io/master')
>>> w.url
'https://demo.pygeoapi.io/master'
>>> conformance = w.conformance()
{'u'conformsTo': [u'http://www.opengis.net/spec/ogcapi-features-1/1.0/conf/core', u'http://
↳ www.opengis.net/spec/ogcapi-features-1/1.0/conf/oas30', u'http://www.opengis.net/spec/
↳ ogcapi-features-1/1.0/conf/html', u'http://www.opengis.net/spec/ogcapi-features-1/1.0/
↳ conf/geojson']}
>>> api = w.api() # OpenAPI document/
>>> collections = w.collections()
>>> len(collections['collections'])
13
>>> feature_collections = w.feature_collections()
>>> len(feature_collections)
13
>>> lakes = w.collection('lakes')
>>> lakes['id']
'lakes'
>>> lakes['title']
'Large Lakes'
>>> lakes['description']
'lakes of the world, public domain'
>>> lakes_queryables = w.collection_queryables('lakes')
>>> len(lakes_queryables['queryables'])
```

(continues on next page)

(continued from previous page)

```

6
>>> lakes_query = w.collection_items('lakes')
>>> lakes_query['features'][0]['properties']
{'u'scalerank': 0, u'name_alt': None, u'admin': None, u'featureclass': u'Lake', u'id': 0,
↳ u'name': u'Lake Baikal'}

```

4.3.2 OGC API - Coverages 1.0

```

>>> from owslib.ogcapi.coverages import Coverages
>>> w = Coverages('https://dev.api.weather.gc.ca/coverages-demo')
>>> w.url
'https://dev.api.weather.gc.ca/coverages-demo/'
>>> api = w.api() # OpenAPI document
>>> collections = w.collections()
>>> len(collections['collections'])
3
>>> coverages = w.coverages()
>>> len(coverages)
1
>>> gdps = w.collection('gdps-temperature')
>>> gdps['id']
'gdps-temperature'
>>> gdps['title']
'Global Deterministic Prediction System sample'
>>> gdps['description']
'Global Deterministic Prediction System sample'
>>> domainset = w.coverage_domainset('gdps-temperature')
>>> domainset['generalGrid']['axisLabels']
['x', 'y']
>>> domainset['generalGrid']['gridLimits']['axisLabels']
['i', 'j']
>>> rangetype = w.coverage_rangetype('gdps-temperature')
>>> len(rangetype['field'])
1
>>> rangetype['field'][0]['definition']
'float64'
>> gdps_coverage_query = w.coverage('gdps-temperature', range_subset=[1])

```

4.3.3 OGC API - Records 1.0

```

>>> from owslib.ogcapi.records import Records
>>> w = Records('https://example.org/records-api')
>>> w.url
'https://example.org/records-api'
>>> conformance = w.conformance()
{'conformsTo': [u'http://www.opengis.net/spec/ogcapi-features-1/1.0/conf/core', u'http://
↳ www.opengis.net/spec/ogcapi-features-1/1.0/conf/oas30', u'http://www.opengis.net/spec/
↳ ogcapi-features-1/1.0/conf/html', u'http://www.opengis.net/spec/ogcapi-features-1/1.0/
↳ conf/gejson', u'http://www.opengis.net/spec/ogcapi-records-1/1.0/req/core', u'http://
↳ www.opengis.net/spec/ogcapi-records-1/1.0/req/oas30', u'http://www.opengis.net/spec/
↳ ogcapi-records-1/1.0/req/json', u'http://www.opengis.net/spec/ogcapi-records-1/1.0/req/
↳ html']}

```

(continued from previous page)

```

>>> api = w.api() # OpenAPI document
>>> collections = w.collections()
>>> len(collections)
1
>>> records = w.records()
>>> len(records)
1
>>> my_catalogue = w.collection('my-catalogue')
>>> my_catalogue['id']
'my-catalogue'
>>> my_catalogue['title']
'My catalogue'
>>> my_catalogue['description']
'My catalogue'
>>> my_catalogue_queryables = w.collection_queryables('my-catalogue')
>>> len(my_catalogue_queryables['queryables'])
8
>>> my_catalogue_query = w.collection_items('my-catalogue')
>>> my_catalogue_query['features'][0]['properties'].keys()
[u'title', u'abstract', u'keywords']
>>> my_catalogue_query['features'][0]['properties']['title']
u'Roadrunner ambush locations'
>>> my_catalogue_query2 = w.collection_items('my-catalogue', q='birds')
>>> msc_wis_dcpc_query2['numberMatched']
2
>>> msc_wis_dcpc_query2['numberReturned']
2

```

4.4 WCS

```

>>> # Import OWSLib in Python once installed
... from owslib.wcs import WebCoverageService

>>> # Create coverage object
... my_wcs = WebCoverageService('http://ows.rasdaman.org/rasdaman/ows',
...                               version='2.0.1')

>>> # Get list of coverages
... print my_wcs.contents.keys()
['RadianceColor', 'test_irr_cube_2', 'test_mean_summer_airtemp', 'test_double_1d',
→ 'INSPIRE_EL', 'AverageChlorophyllScaled', 'INSPIRE_OI_RGB', 'Temperature4D', 'INSPIRE_
→ OI_IR', 'visible_human', 'INSPIRE_WS_LC', 'meris_lai', 'climate_earth', 'mean_summer_
→ airtemp', 'multiband', 'ls8_coastal_aerosol', 'NN3_3', 'NN3_2', 'NN3_1', 'NN3_4',
→ 'AvgTemperatureColorScaled', 'AverageChloroColorScaled', 'lena', 'Germany_DTM',
→ 'climate_cloud', 'FiLCCoverageBit', 'AverageChloroColor', 'LandsatMultiBand',
→ 'RadianceColorScaled', 'AvgLandTemp', 'NIR', 'BlueMarbleCov']

>>> # Get geo-bounding boxes and native CRS
... my_wcs.contents['AverageChlorophyllScaled'].boundingboxes
[{'nativeSrs': 'http://ows.rasdaman.org/def/crs-compound?1=http://ows.rasdaman.org/def/
→ crs/EPSG/0/4326&2=http://ows.rasdaman.org/def/crs/OGC/0/UnixTime', 'bbox': ('180.0, 90.0, 180.0)}]

```

(continues on next page)

(continued from previous page)

```

>>> # Get axis labels
... my_wcs.contents['AverageChlorophyllScaled'].grid.axislabels
['Lat', 'Long', 'unix']

>>> # Get dimension
... my_wcs.contents['AverageChlorophyllScaled'].grid.dimension
3

>>> # Get grid lower and upper bounds
... my_wcs.contents['AverageChlorophyllScaled'].grid.lowlimits
['0', '0', '0']

>>> my_wcs.contents['AverageChlorophyllScaled'].grid.hightlimits
['119', '239', '5']

>>> # Get offset vectors for geo axes
... my_wcs.contents['AverageChlorophyllScaled'].grid.offsetvectors
[['-1.5', '0', '0'], ['0', '1.5', '0'], ['0', '0', '1']]

>>> # For coverage with time axis get the date time values
... my_wcs.contents['AverageChlorophyllScaled'].timepositions
[datetime.datetime(2015, 1, 1, 0, 0), datetime.datetime(2015, 2, 1, 0, 0), datetime.
↪ datetime(2015, 3, 1, 0, 0), datetime.datetime(2015, 4, 1, 0, 0), datetime.
↪ datetime(2015, 5, 1, 0, 0), datetime.datetime(2015, 7, 1, 0, 0)]

```

4.5 CSW

Connect to a CSW, and inspect its properties:

```

>>> from owslib.csw import CatalogueServiceWeb
>>> csw = CatalogueServiceWeb('http://geodiscover.cgdi.ca/wes/serviceManagerCSW/csw')
>>> 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': 'csw:DomainValuesType'}
>>>

```

Search for bird data:

```

>>> from owslib.fes import PropertyIsEqualTo, PropertyIsLike, BBox
>>> birds_query = PropertyIsEqualTo('csw:AnyText', 'birds')
>>> csw.getrecords2(constraints=[birds_query], maxrecords=20)

```

(continues on next page)

(continued from previous page)

```
>>> csw.results
{'matches': 101, 'nextrecord': 21, 'returned': 20}
>>> for rec in csw.records:
...     print(csw.records[rec].title)
...
ALLSPECIES
NatureServe Canada References
Bird Studies Canada - BirdMap WMS
Parks Canada Geomatics Metadata Repository
Bird Studies Canada - BirdMap WFS
eBird Canada - Survey Locations
WHC CitizenScience WMS
Project FeederWatch - Survey Locations
North American Bird Banding and Encounter Database
Wildlife Habitat Canada CitizenScience WFS
Parks Canada Geomatics Metadata Repository
Parks Canada Geomatics Metadata Repository
Wildlife Habitat Canada CitizenScience WMS
Canadian IBA Polygon layer
Land
Wildlife Habitat Canada CitizenScience WMS
WATER
Parks Canada Geomatics Metadata Repository
Breeding Bird Survey
SCALE
>>>
```

Search for bird data in Canada:

```
>>> bbox_query = BBox([-141,42,-52,84])
>>> csw.getrecords2(constraints=[birds_query, bbox_query])
>>> csw.results
{'matches': 3, 'nextrecord': 0, 'returned': 3}
>>>
```

Search for keywords like 'birds' or 'fowl'

```
>>> birds_query_like = PropertyIsLike('dc:subject', '%birds%')
>>> fowl_query_like = PropertyIsLike('dc:subject', '%fowl%')
>>> csw.getrecords2(constraints=[birds_query_like, fowl_query_like])
>>> csw.results
{'matches': 107, 'nextrecord': 11, 'returned': 10}
>>>
```

Search for a specific record:

```
>>> csw.getrecordbyid(id=['9250AA67-F3AC-6C12-0CB9-0662231AA181'])
>>> c.records['9250AA67-F3AC-6C12-0CB9-0662231AA181'].title
'ALLSPECIES'
```

Search with a CQL query

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

Transaction: insert

```
>>> csw.transaction(ttype='insert', typename='gmd:MD_Metadata', record=open("file.xml").
↳ read())
```

Transaction: update

```
>>> # update ALL records
>>> csw.transaction(ttype='update', typename='csw:Record', propertyname='dc:title',
↳ propertyvalue='New Title')
>>> # update records satisfying keywords filter
>>> csw.transaction(ttype='update', typename='csw:Record', propertyname='dc:title',
↳ propertyvalue='New Title', keywords=['birds','fowl'])
>>> # update records satisfying BBOX filter
>>> csw.transaction(ttype='update', typename='csw:Record', propertyname='dc:title',
↳ propertyvalue='New Title', bbox=[-141,42,-52,84])
```

Transaction: delete

```
>>> # delete ALL records
>>> csw.transaction(ttype='delete', typename='gmd:MD_Metadata')
>>> # delete records satisfying keywords filter
>>> csw.transaction(ttype='delete', typename='gmd:MD_Metadata', keywords=['birds','fowl
↳ '])
>>> # delete records satisfying BBOX filter
>>> csw.transaction(ttype='delete', typename='gmd:MD_Metadata', bbox=[-141,42,-52,84])
```

Harvest a resource

```
>>> csw.harvest('http://host/url.xml', 'http://www.isotc211.org/2005/gmd')
```

4.6 WMC

4.7 WPS

Inspect a remote WPS and retrieve the supported processes:

```
>>> from owslib.wps import WebProcessingService
>>> wps = WebProcessingService('http://cida.usgs.gov/climate/gdp/process/
↳ WebProcessingService', verbose=False, skip_caps=True)
>>> wps.getcapabilities()
>>> wps.identification.type
'WPS'
>>> wps.identification.title
'Geo Data Portal WPS Processing'
>>> wps.identification.abstract
'Geo Data Portal WPS Processing'
>>> for operation in wps.operations:
...     operation.name
```

(continues on next page)

(continued from previous page)

```

...
'GetCapabilities'
'DescribeProcess'
'Execute'
>>> for process in wps.processes:
...     process.identifier, process.title
...
('gov.usgs.cida.gdp.wps.algorithm.FeatureCoverageIntersectionAlgorithm', 'Feature_
↳ Coverage WCS Intersection')
('gov.usgs.cida.gdp.wps.algorithm.FeatureCoverageOPeNDAPIntersectionAlgorithm', 'Feature_
↳ Coverage OPeNDAP Intersection')
('gov.usgs.cida.gdp.wps.algorithm.FeatureCategoricalGridCoverageAlgorithm', 'Feature_
↳ Categorical Grid Coverage')
('gov.usgs.cida.gdp.wps.algorithm.FeatureWeightedGridStatisticsAlgorithm', 'Feature_
↳ Weighted Grid Statistics')
('gov.usgs.cida.gdp.wps.algorithm.FeatureGridStatisticsAlgorithm', 'Feature Grid_
↳ Statistics')
('gov.usgs.cida.gdp.wps.algorithm.PRMSParameterGeneratorAlgorithm', 'PRMS Parameter_
↳ Generator')
>>>

```

Determine how a specific process needs to be invoked - i.e. what are its input parameters, and output result:

```

>>> from owslib.wps import printInputOutput
>>> process = wps.describeprocess('gov.usgs.cida.gdp.wps.algorithm.
↳ FeatureWeightedGridStatisticsAlgorithm')
>>> process.identifier
'gov.usgs.cida.gdp.wps.algorithm.FeatureWeightedGridStatisticsAlgorithm'
>>> process.title
'Feature Weighted Grid Statistics'
>>> process.abstract
'This algorithm generates area weighted statistics of a gridded dataset for a set of_
↳ vector polygon features. Using the bounding-box that encloses ...
>>> for input in process.dataInputs:
...     printInputOutput(input)
...
identifier=FEATURE_COLLECTION, title=Feature Collection, abstract=A feature collection_
↳ encoded as a WFS request or one of the supported GML profiles.,...
Supported Value: mimeType=text/xml, encoding=UTF-8, schema=http://schemas.opengis.net/
↳ gml/2.0.0/feature.xsd
Supported Value: mimeType=text/xml, encoding=UTF-8, schema=http://schemas.opengis.net/
↳ gml/2.1.1/feature.xsd
Supported Value: mimeType=text/xml, encoding=UTF-8, schema=http://schemas.opengis.net/
↳ gml/2.1.2/feature.xsd
Supported Value: mimeType=text/xml, encoding=UTF-8, schema=http://schemas.opengis.net/
↳ gml/2.1.2.1/feature.xsd
Supported Value: mimeType=text/xml, encoding=UTF-8, schema=http://schemas.opengis.net/
↳ gml/3.0.0/base/feature.xsd
Supported Value: mimeType=text/xml, encoding=UTF-8, schema=http://schemas.opengis.net/
↳ gml/3.0.1/base/feature.xsd
Supported Value: mimeType=text/xml, encoding=UTF-8, schema=http://schemas.opengis.net/
↳ gml/3.1.0/base/feature.xsd

```

(continues on next page)

(continued from previous page)

Supported Value: mimeType=text/xml, encoding=UTF-8, schema=http://schemas.opengis.net/gml/3.1.1/base/feature.xsd
 ↳gml/3.1.1/base/feature.xsd
 Supported Value: mimeType=text/xml, encoding=UTF-8, schema=http://schemas.opengis.net/gml/3.2.1/base/feature.xsd
 ↳gml/3.2.1/base/feature.xsd
 Default Value: mimeType=text/xml, encoding=UTF-8, schema=http://schemas.opengis.net/gml/2.0.0/feature.xsd
 ↳2.0.0/feature.xsd
 minOccurs=1, maxOccurs=1
 identifier=DATASET_URI, title=Dataset URI, abstract=The base data web service URI for the dataset of interest., data type=anyURI
 ↳the dataset of interest., data type=anyURI
 Allowed Value: AnyValue
 Default Value: None
 minOccurs=1, maxOccurs=1
 identifier=DATASET_ID, title=Dataset Identifier, abstract=The unique identifier for the data type or variable of interest., data type=string
 ↳data type or variable of interest., data type=string
 Allowed Value: AnyValue
 Default Value: None
 minOccurs=1, maxOccurs=2147483647
 identifier=REQUIRE_FULL_COVERAGE, title=Require Full Coverage, abstract=If turned on, the service will require that the dataset of interest
 ↳the service will require that the dataset of interest
 Allowed Value: True
 Default Value: True
 minOccurs=1, maxOccurs=1
 identifier=TIME_START, title=Time Start, abstract=The date to begin analysis., data type=dateTime
 ↳type=dateTime
 Allowed Value: AnyValue
 Default Value: None
 minOccurs=0, maxOccurs=1
 identifier=TIME_END, title=Time End, abstract=The date to end analysis., data type=dateTime
 ↳type=dateTime
 Allowed Value: AnyValue
 Default Value: None
 minOccurs=0, maxOccurs=1
 identifier=FEATURE_ATTRIBUTE_NAME, title=Feature Attribute Name, abstract=The attribute that will be used to label column headers in processing output., ...
 ↳that will be used to label column headers in processing output., ...
 Allowed Value: AnyValue
 Default Value: None
 minOccurs=1, maxOccurs=1
 identifier=DELIMITER, title=Delimiter, abstract=The delimiter that will be used to separate columns in the processing output., data type=string
 ↳separate columns in the processing output., data type=string
 Allowed Value: COMMA
 Allowed Value: TAB
 Allowed Value: SPACE
 Default Value: COMMA
 minOccurs=1, maxOccurs=1
 identifier=STATISTICS, title=Statistics, abstract=Statistics that will be returned for each feature in the processing output., data type=string
 ↳each feature in the processing output., data type=string
 Allowed Value: MEAN
 Allowed Value: MINIMUM
 Allowed Value: MAXIMUM
 Allowed Value: VARIANCE
 Allowed Value: STD_DEV
 Allowed Value: SUM

(continues on next page)

(continued from previous page)

```

Allowed Value: COUNT
Default Value: None
minOccurs=1, maxOccurs=7
identifier=GROUP_BY, title=Group By, abstract=If multiple features and statistics are
↳selected, this will change whether the processing output ...
Allowed Value: STATISTIC
Allowed Value: FEATURE_ATTRIBUTE
Default Value: None
minOccurs=1, maxOccurs=1
identifier=SUMMARIZE_Timestep, title=Summarize Timestep, abstract=If selected,
↳processing output will include columns with summarized statistics ...
Allowed Value: True
Default Value: True
minOccurs=0, maxOccurs=1
identifier=SUMMARIZE_FEATURE_ATTRIBUTE, title=Summarize Feature Attribute, abstract=If
↳selected, processing output will include a final row of ...
Allowed Value: True
Default Value: True
minOccurs=0, maxOccurs=1
>>> for output in process.processOutputs:
...     printInputOutput(output)
...
    identifier=OUTPUT, title=Output File, abstract=A delimited text file containing
↳requested process output., data type=ComplexData
Supported Value: mimeType=text/csv, encoding=UTF-8, schema=None
Default Value: mimeType=text/csv, encoding=UTF-8, schema=None
reference=None, mimeType=None
>>>

```

Submit a processing request (extraction of a climate index variable over a specific GML polygon, for a given period of time), monitor the execution until complete:

```

>>> from owslib.wps import GMLMultiPolygonFeatureCollection
>>> polygon = [(-102.8184, 39.5273), (-102.8184, 37.418), (-101.2363, 37.418), (-101.
↳2363, 39.5273), (-102.8184, 39.5273)]
>>> featureCollection = GMLMultiPolygonFeatureCollection( [polygon] )
>>> processid = 'gov.usgs.cida.gdp.wps.algorithm.FeatureWeightedGridStatisticsAlgorithm'
>>> inputs = [ ("FEATURE_ATTRIBUTE_NAME", "the_geom"),
...            ("DATASET_URI", "dods://cida.usgs.gov/qa/thredds/dodsC/derivatives/
↳derivative-days_above_threshold.pr.ncml"),
...            ("DATASET_ID", "ensemble_b1_pr-days_above_threshold"),
...            ("TIME_START", "2010-01-01T00:00:00.000Z"),
...            ("TIME_END", "2011-01-01T00:00:00.000Z"),
...            ("REQUIRE_FULL_COVERAGE", "false"),
...            ("DELIMITER", "COMMA"),
...            ("STATISTICS", "MEAN"),
...            ("GROUP_BY", "STATISTIC"),
...            ("SUMMARIZE_Timestep", "false"),
...            ("SUMMARIZE_FEATURE_ATTRIBUTE", "false"),
...            ("FEATURE_COLLECTION", featureCollection)
...            ]
>>> output = "OUTPUT"

```

(continues on next page)

(continued from previous page)

```
>>> execution = wps.execute(processid, inputs, output = "OUTPUT")
Executing WPS request...
Execution status=ProcessStarted
>>> from owslib.wps import monitorExecution
>>> monitorExecution(execution)
Checking execution status... (location=http://cida.usgs.gov/climate/gdp/process/
↳RetrieveResultServlet?id=6809217153012787208)
Execution status=ProcessSucceeded
Execution status: ProcessSucceeded
Output URL=http://cida.usgs.gov/climate/gdp/process/RetrieveResultServlet?
↳id=6809217153012787208OUTPUT.3cbcd666-a912-456f-84a3-6ede450aca95
```

Alternatively, define the feature through an embedded query to a WFS server:

```
>>> from owslib.wps import WFSQuery, WFSFeatureCollection
>>> wfsUrl = "http://cida.usgs.gov/climate/gdp/proxy/http://igsarm-cida-gdp2.er.usgs.
↳gov:8082/geoserver/wfs"
>>> query = WFSQuery("sample:CONUS_States", propertyNames=['the_geom',"STATE"], filters=[
↳"CONUS_States.508","CONUS_States.469"])
>>> featureCollection = WFSFeatureCollection(wfsUrl, query)
>>> # same process submission as above
...

```

You can also submit a pre-made request encoded as WPS XML:

```
>>> request = open('/Users/cinquini/Documents/workspace-cog/wps/tests/resources/wps_
↳USGSExecuteRequest1.xml','rb').read()
>>> execution = wps.execute(None, [], request=request)
Executing WPS request...
Execution status=ProcessStarted
>>> monitorExecution(execution)
Checking execution status... (location=http://cida.usgs.gov/climate/gdp/process/
↳RetrieveResultServlet?id=5103866488472745994)
Execution status=ProcessSucceeded
Execution status: ProcessSucceeded
Output URL=http://cida.usgs.gov/climate/gdp/process/RetrieveResultServlet?
↳id=5103866488472745994OUTPUT.f80e2a78-96a9-4343-9777-be60fac5b256
```

4.8 SOS 1.0

GetCapabilities

Imports

```
>>> from tests.utils import cast_tuple_int_list, resource_file
>>> from owslib.sos import SensorObservationService
>>> from owslib.fes import FilterCapabilities
>>> from owslib.ows import OperationsMetadata
>>> from owslib.crs import Crs
>>> from datetime import datetime
>>> from operator import itemgetter
```

Initialize ncSOS

```
>>> xml = open(resource_file('sos_ncSOS_getcapabilities.xml'), 'rb').read()
>>> ncsos = SensorObservationService(None, xml=xml)
```

Initialize 52N

```
>>> xml = open(resource_file('sos_52n_getcapabilities.xml'), 'rb').read()
>>> f2n = SensorObservationService(None, xml=xml)
```

Initialize NDBC

```
>>> xml = open(resource_file('sos_ndbc_getcapabilities.xml'), 'rb').read()
>>> ndbc = SensorObservationService(None, xml=xml)
```

ServiceIdentification

```
>>> id = ndbc.identification
```

```
>>> id.service
'OGC:SOS'
```

```
>>> id.version
'1.0.0'
```

```
>>> id.title
'National Data Buoy Center SOS'
```

```
>>> id.abstract
'National Data Buoy Center SOS'
```

```
>>> id.keywords
['Weather', 'Ocean Currents', 'Air Temperature', 'Water Temperature', 'Conductivity',
↳ 'Salinity', 'Barometric Pressure', 'Water Level', 'Waves', 'Winds', 'NDBC']
```

```
>>> id.fees
'NONE'
```

```
>>> id.accessconstraints
'NONE'
```

ServiceProvider

```
>>> p = ndbc.provider
```

```
>>> p.name
'National Data Buoy Center'
```

```
>>> p.url
'http://sdf.ndbc.noaa.gov/'
```

ServiceContact

```
>>> sc = p.contact
```

Unused fields should return nothing

```
>>> sc.role
>>> sc.position
>>> sc.instructions
>>> sc.organization
>>> sc.fax
>>> sc.hours
```

```
>>> sc.name
'Webmaster'
```

```
>>> sc.phone
'228-688-2805'
```

```
>>> sc.address
'Bldg. 3205'
```

```
>>> sc.city
'Stennis Space Center'
```

```
>>> sc.region
'MS'
```

```
>>> sc.postcode
'39529'
```

```
>>> sc.country
'USA'
```

```
>>> sc.email
'webmaster.ndbc@noaa.gov'
```

OperationsMetadata

```
>>> o = ndbc.operations
>>> len(o)
3
```

Name

```
>>> len(o)
3
```

Get by name

```
>>> getcap = ndbc.get_operation_by_name('GetCapabilities')
>>> isinstance(getcap, OperationsMetadata)
True
```

Get by name (case insensitive)

```
>>> getcap = ndbc.get_operation_by_name('getcapabilities')
>>> isinstance(getcap, OperationsMetadata)
True
```

GetCapabilities

```
>>> getcap.constraints
[]
```

```
>>> x = getcap.parameters
>>> x == {'Sections': {'values': ['ServiceIdentification', 'ServiceProvider',
↳ 'OperationsMetadata', 'Contents', 'All']}}
True
```

```
>>> x = sorted(getcap.methods, key=itemgetter('type'))
>>> x == [{'type': 'Get', 'url': 'http://sdf.ndbc.noaa.gov/sos/server.php',
↳ 'constraints': []}, {'type': 'Post', 'url': 'http://sdf.ndbc.noaa.gov/sos/
↳ server.php', 'constraints': []}]
True
```

DescribeSensor

```
>>> descscen = ndbc.get_operation_by_name('describesensor')
```

```
>>> descscen.constraints
[]
```

```
>>> x = descscen.parameters
>>> x == {'outputFormat': {'values': ['text/xml;subtype="sensorML/1.0.1"']}}
True
```

```
>>> x = sorted(descscen.methods, key=itemgetter('type'))
>>> x == [{'type': 'Get', 'url': 'http://sdf.ndbc.noaa.gov/sos/server.php',
↳ 'constraints': []}, {'type': 'Post', 'url': 'http://sdf.ndbc.noaa.gov/sos/
↳ server.php', 'constraints': []}]
True
```

GetObservation

```
>>> getob = ndbc.get_operation_by_name('getobservation')
```

```
>>> getob.constraints
[]
```

```
>>> x = getob.parameters
>>> x == {'observedProperty': {'values': ['air_temperature', 'air_pressure_at_
↳ sea_level', 'sea_water_electrical_conductivity', 'currents', 'sea_water_
↳ salinity', 'sea_floor_depth_below_sea_surface', 'sea_water_temperature',
↳ 'waves', 'winds']}}
True
```

```
>>> x = sorted(getob.methods, key=itemgetter('type'))
>>> x == [{'type': 'Get', 'url': 'http://sdf.ndbc.noaa.gov/sos/server.php',
↳ 'constraints': []}, {'type': 'Post', 'url': 'http://sdf.ndbc.noaa.gov/sos/
↳ server.php', 'constraints': []}]
True
```

Filter_Capabilities

```
>>> filter = ndbc.filters
>>> isinstance(filter, FilterCapabilities)
False
```

Contents

```
>>> contents = ndbc.contents
>>> len(contents)
848
```

Dict access __getitem__

```
>>> ndbc['station-46084'].name
'urn:ioos:station:wmo:46084'
```

```
>>> ndbc['rubbishrubbish'].name
Traceback (most recent call last):
...
KeyError: 'No Observational Offering with id: rubbishrubbish'
```

Network

```
>>> network = contents['network-all']
>>> network.id
'network-all'
```

```
>>> network.name
'urn:ioos:network:noaa.nws.ndbc:all'
```

```
>>> network.description
'All stations on the NDBC SOS server'
```

```
>>> srs = network.srs
>>> isinstance(srs, Crs)
True
>>> srs.getcodeurn()
'urn:ogc:def:crs:EPSG::4326'
>>> srs.getcode()
'EPSG:4326'
```

(left, bottom, right, top)

```
>>> cast_tuple_int_list(network.bbox)
[-179, -77, 180, 80]
```

(continues on next page)

(continued from previous page)

```
>>> bbsrs = network.bbox_srs
>>> isinstance(bbsrs, Crs)
True
>>> bbsrs.getcodeurn()
'urn:ogc:def:crs:EPSG::4326'
>>> bbsrs.getcode()
'EPSG:4326'
```

```
>>> bp = network.begin_position
>>> isinstance(bp, datetime)
True
>>> ep = network.end_position
>>> isinstance(ep, datetime)
True
```

```
>>> network.result_model
'om:Observation'
```

```
>>> procs = network.procedures
>>> len(procs)
847
```

```
>>> network.observed_properties
['http://mmisw.org/ont/cf/parameter/air_temperature', 'http://mmisw.org/ont/cf/
↳parameter/air_pressure_at_sea_level', 'http://mmisw.org/ont/cf/parameter/sea_
↳water_electrical_conductivity', 'http://mmisw.org/ont/cf/parameter/currents',
↳'http://mmisw.org/ont/cf/parameter/sea_water_salinity', 'http://mmisw.org/ont/cf/
↳parameter/sea_floor_depth_below_sea_surface', 'http://mmisw.org/ont/cf/parameter/
↳sea_water_temperature', 'http://mmisw.org/ont/cf/parameter/waves', 'http://mmisw.
↳org/ont/cf/parameter/winds']
```

```
>>> foi = network.features_of_interest
>>> len(foi)
1082
```

```
>>> rfs = network.response_formats
>>> len(rfs)
5
>>> rfs[-1]
'application/vnd.google-earth.kml+xml'
```

```
>>> rms = network.response_modes
>>> len(rms)
1
>>> rms[0]
'inline'
```

Station

```
>>> station = contents['station-zbqn7']
>>> station.id
'station-zbqn7'
```

```
>>> station.name
'urn:ioos:station:wmo:zbqn7'
```

```
>>> station.description
'Zeke's Basin, North Carolina'
```

```
>>> srs = station.srs
>>> isinstance(srs, Crs)
True
>>> srs.getcodeurn()
'urn:ogc:def:crs:EPSG::4326'
>>> srs.getcode()
'EPSG:4326'
>>> cast_tuple_int_list(station.bbox)
[-77, 33, -77, 33]
```

```
>>> bbsrs = station.bbox_srs
>>> isinstance(bbsrs, Crs)
True
>>> bbsrs.getcodeurn()
'urn:ogc:def:crs:EPSG::4326'
>>> bbsrs.getcode()
'EPSG:4326'
```

```
>>> bp = station.begin_position
>>> isinstance(bp, datetime)
True
>>> ep = station.end_position
>>> isinstance(ep, datetime)
True
```

```
>>> station.result_model
'om:Observation'
```

```
>>> procs = station.procedures
>>> len(procs)
1
>>> procs[0]
'urn:ioos:station:wmo:zbqn7'
```

```
>>> ops = station.observed_properties
>>> len(ops)
3
>>> ops[0]
'http://mmisw.org/ont/cf/parameter/sea_water_electrical_conductivity'
```

```
>>> foi = station.features_of_interest
>>> len(foi)
1
>>> foi[0]
'urn:cgi:Feature:CGI:EarthOcean'
```

```
>>> rfs = station.response_formats
>>> len(rfs)
5
>>> rfs[0]
'text/xml;schema="ioos/0.6.1"'
```

```
>>> rm = station.response_modes
>>> len(rm)
1
>>> rm[0]
'inline'
```

GetObservation

Imports

```
>>> from tests.utils import resource_file
>>> from owslib.sos import SensorObservationService
>>> from owslib.fes import FilterCapabilities200
>>> from owslib.ows import OperationsMetadata
>>> from owslib.crs import Crs
>>> from datetime import datetime
>>> from operator import itemgetter
```

Initialize

```
>>> xml = open(resource_file('sos_ngwd.xml'),'rb').read()
>>> ngwd = SensorObservationService(None, xml=xml, version='2.0.0')
```

ServiceIdentification

```
>>> id = ngwd.identification
```

```
>>> id.service
'OGC:SOS'
```

```
>>> id.version
'2.0.0'
```

```
>>> len(id.profiles)
5
```

```
>>> id.title
'GIN SOS'
```



```
>>> id.abstract
'GIN SOS mediator'
```

```
>>> id.keywords
['water level', 'groundwater level', 'surface water flow']
```

```
>>> id.fees
'NONE'
```

```
>>> id.accessconstraints
'NONE'
```

ServiceProvider

```
>>> p = ngwd.provider
```

```
>>> p.name
'Geological Survey of Canada, Earth Sciences Sector, Natural Resources Canada,
↳ Government of Canada'
```

```
>>> p.url
'http://gw-info.net'
```

ServiceContact

```
>>> sc = p.contact
```

Unused fields should return nothing >>> sc.role >>> sc.position 'Research Scientist' >>> sc.instructions
>>> sc.organization >>> sc.fax '+1-613-995-9273' >>> sc.hours

```
>>> sc.name
'Boyan Brodaric'
```

```
>>> sc.phone
'+1-613-992-3562'
```

```
>>> sc.address
'615 Booth Street'
```

```
>>> sc.city
'Ottawa'
```

```
>>> sc.region
```

```
>>> sc.postcode
'K1A 0E9'
```

```
>>> sc.country
'Canada'
```

```
>>> sc.email
'brodaric at nrcan dot gc dot ca'
```

OperationsMetadata

```
>>> o = ngwd.operations
>>> len(o)
4
```

Get by name >>> getcap = ngwd.get_operation_by_name('GetCapabilities') >>> isinstance(getcap, OperationsMetadata) True

Get by name (case insensitive) >>> getcap = ngwd.get_operation_by_name('getcapabilities') >>> isinstance(getcap, OperationsMetadata) True

GetCapabilities >>> getcap.constraints []

```
>>> x = getcap.parameters
>>> x == {'AcceptVersions': {'values': ['2.0.0']}, 'updateSequence': {'values': []}, 'Sections': {'values': ['ServiceIdentification', 'ServiceProvider', 'OperationsMetadata', 'FilterCapabilities', 'Contents', 'All']}, 'AcceptFormats': {'values': ['text/xml', 'application/zip']}}
True
```

```
>>> x = sorted(getcap.methods, key=itemgetter('type'))
>>> x == [{'type': 'Get', 'url': 'http://ngwd-bdnes.cits.nrcan.gc.ca:8080/proxy/GinService/sos/gw?', 'constraints': []}, {'type': 'Post', 'url': 'http://ngwd-bdnes.cits.nrcan.gc.ca:8080/proxy/GinService/sos/gw', 'constraints': []}]
True
```

DescribeSensor >>> descscen = ngwd.get_operation_by_name('describesensor')

```
>>> descscen.constraints
[]
```

```
>>> x = descscen.parameters
>>> x == {'procedureDescriptionFormat': {'values': ['http://www.opengis.net/sensorML/1.0.1']}, 'procedure': {'values': ['urn:ogc:object:Sensor::GIN_GroundwaterLevelProcess']}}
True
```

```
>>> x = sorted(descscen.methods, key=itemgetter('type'))
>>> x == [{'type': 'Get', 'url': 'http://ngwd-bdnes.cits.nrcan.gc.ca:8080/proxy/GinService/sos/gw?', 'constraints': []}, {'type': 'Post', 'url': 'http://ngwd-bdnes.cits.nrcan.gc.ca:8080/proxy/GinService/sos/gw', 'constraints': []}]
True
```

GetObservation >>> getob = ngwd.get_operation_by_name('getobservation')

```
>>> getob.constraints
[]
```

```
>>> x = getob.parameters
>>> x == {'srsName': {'values': []}, 'temporalFilter': {'values': []},
↳ 'offering': {'values': ['GW_LEVEL']}, 'result': {'values': []},
↳ 'responseFormat': {'values': ['http://www.opengis.net/waterml/2.0',
↳ 'application/zip']}, 'observedProperty': {'values': [
↳ 'urn:ogc:def:phenomenon:OGC:1.0.30:groundwaterlevel']}, 'procedure': {'values
↳ ': ['urn:ogc:object:Sensor::GIN_GroundwaterLevelProcess']}]}}
True
```

```
>>> x = getob.methods
>>> x == [{'type': 'Get', 'url': 'http://ngwd-bdnes.cits.nrcan.gc.ca:8080/
↳ proxy/GinService/sos/gw?', 'constraints': []}, {'type': 'Post', 'url':
↳ 'http://ngwd-bdnes.cits.nrcan.gc.ca:8080/proxy/GinService/sos/gw',
↳ 'constraints': []}]
True
```

Filter_Capabilities

```
>>> filter = ngwd.filters
>>> isinstance(filter, FilterCapabilities200)
False
```

Contents

```
>>> contents = ngwd.contents
>>> len(contents)
1
```

```
Network >>> network = contents['GW_LEVEL'] >>> network.id 'GW_LEVEL'
```

```
>>> network.procedures
['urn:ogc:object:Sensor::GIN_GroundwaterLevelProcess']
```

```
>>> srs = network.bbox_srs
>>> isinstance(srs, Crs)
True
>>> srs.id
'http://www.opengis.net/def/crs/EPSG/0/4326'
```

```
>>> srs.getcodeurn()
'urn:ogc:def:crs:EPSG::4326'
```

```
>>> srs.getcode()
'EPSG:4326'
```

```
# (left, bottom, right, top) >>> network.bbox (-120.0, 41.0, -60.0, 60.0)
```

```
>>> bbsrs = network.bbox_srs
```

```
>>> isinstance(bbsrs, Crs)
True
```

```
>>> bbsrs.getcodeurn()
'urn:ogc:def:crs:EPSG::4326'
```

```
>>> bbsrs.getcode()
'EPSG:4326'
```

```
>>> bp = network.begin_position
>>> isinstance(bp, datetime)
True
>>> ep = network.end_position
>>> isinstance(ep, datetime)
True
```

```
>>> network.observed_properties
['urn:ogc:def:phenomenon:OGC:1.0.30:groundwaterlevel']
```

```
>>> network.response_formats
[]
```

```
>>> network.observation_models
[]
```

4.9 SOS 2.0

4.9.1 Examples of service metadata and GetObservation

Tests using the 52North demo service

```
>>> from owslib.sos import SensorObservationService
>>> service = SensorObservationService('http://sensorweb.demo.52north.org/52n-sos-webapp/
↳ sos/kvp', version='2.0.0')
>>> for content in sorted(service.contents):
...     print(content)
...
http://www.52north.org/test/offering/1
http://www.52north.org/test/offering/2
http://www.52north.org/test/offering/3
http://www.52north.org/test/offering/4
http://www.52north.org/test/offering/5
http://www.52north.org/test/offering/6
http://www.52north.org/test/offering/7
http://www.52north.org/test/offering/8
http://www.52north.org/test/offering/developer
```

```
>>> id = service.identification
```

Check basic service metadata >>> id.service 'OGC:SOS'

```
>>> id.title
'52N SOS'
```

```
>>> provider=service.provider
>>> provider.name
'52North'
```

```
>>> len(service.operations)
16
```

Check allowed params for get FOI

```
>>> get_foi=service.get_operation_by_name('GetFeatureOfInterest')
>>> try:
...     x = unicode('test')
...     for x in sorted(get_foi.parameters['featureOfInterest']['values']):
...         print(x.encode('utf8'))
... except:
...     for x in sorted(get_foi.parameters['featureOfInterest']['values']):
...         print(x)
http://www.52north.org/test/featureOfInterest/1
http://www.52north.org/test/featureOfInterest/2
http://www.52north.org/test/featureOfInterest/3
http://www.52north.org/test/featureOfInterest/4
http://www.52north.org/test/featureOfInterest/5
http://www.52north.org/test/featureOfInterest/6
http://www.52north.org/test/featureOfInterest/7
http://www.52north.org/test/featureOfInterest/8
http://www.52north.org/test/featureOfInterest/Heiden
http://www.52north.org/test/featureOfInterest/Münster/FE101
http://www.52north.org/test/featureOfInterest/Portland
http://www.52north.org/test/featureOfInterest/TODO
http://www.52north.org/test/featureOfInterest/world
```

Check allowed params for get observation

```
>>> get_obs=service.get_operation_by_name('GetObservation')
get_obs.parameters['responseFormat']['values']
['http://www.opengis.net/om/2.0']
```

Get observation call

Get latest value using O&M response format

```
#' <?xml version="1.0" encoding="UTF-8"?>n<sos:GetObservationResponse xmlns:sos="http://www.opengis.net/sos/2.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"          xmlns:om="http://www.opengis.net/om/2.0"
xmlns:gml="http://www.opengis.net/gml/3.2"                  xmlns:xlink="http://www.w3.org/1999/xlink"
xsi:schemaLocation="http://www.opengis.net/sos/2.0          http://schemas.opengis.net/sos/2.0/sosGetObservation.xsd
http://www.opengis.net/gml/3.2          http://schemas.opengis.net/gml/3.2.1/gml.xsd          http://www.opengis.net/om/2.0
http://schemas.opengis.net/om/2.0/observation.xsd">n          <sos:observationData>n          <om:OM_Observation
gml:id="o_729058FD6AAFC4701C83D125175F889F51FD4798">n <om:type xlink:href="http://www.opengis.net/def/observationTyp
OM/2.0/OM_Measurement"/>n <om:phenomenonTime>n <gml:TimeInstant gml:id="phenomenonTime_2118549">n
<gml:timePosition>2014-07-10T04:31:58.000Z</gml:timePosition>n          </gml:TimeInstant>n
</om:phenomenonTime>n          <om:resultTime          xlink:href="#phenomenonTime_2118549"/>n
```

```

<om:procedure                                xlink:href="http://geoviqua.dev.52north.org/procedures/WXT520"/>n
<om:observedProperty                         xlink:href="http://geoviqua.dev.52north.org/properties/AirTemperature"/>n
<om:featureOfInterest                       xlink:href="http://geoviqua.dev.52north.org/stations/Vaisala-WXT520"
xlink:title="Vaisala_WXT520"/>n              <om:result                                xmlns:ns="http://www.opengis.net/gml/3.2"
uom="degC"                                xsi:type="ns:MeasureType">17.2</om:result>n              </om:OM_Observation>n
</sos:observationData>n</sos:GetObservationResponse>'    >>>    latest_response    =    ser-
vice.get_observation('http://www.opengis.net/om/2.0',      ['http://www.52north.org/test/offering/1'],
['http://www.52north.org/test/observableProperty/1'], eventTime='om:phenomenonTime,latest', timeout=120)

```

4.9.2 Using the GetObservation response decoder for O&M and WaterML2.0 results

SOS version 2.0 tests using the 52North installation for IOOS: <http://ioossos.axiomalaska.com/>

Imports

```

>>> from owslib.sos import SensorObservationService
>>> from owslib.swe.observation.sos200 import SOSGetObservationResponse
>>> from owslib.etree import etree

```

Setup

```
#>>> service = SensorObservationService('http://geoviqua.dev.52north.org/SOS-Q/sos/kvp',version='2.0.0')
```

```

>>> service = SensorObservationService('http://ioossos.axiomalaska.com/52n-sos-ioos-
↳stable/sos/kvp',version='2.0.0')

```

```

#          http://ioossos.axiomalaska.com/52n-sos-ioos-stable/sos/kvp?service=SOS&request=GetObservation&
namespaces=xmlns(om%2Chttp%3A%2F%2Fwww.opengis.net%2Fom%2F2.0)&temporalFilter=om%
3AphenomenonTime%2Clatest&version=2.0.0

```

Check allowed params for get observation

```
>>> get_obs=service.get_operation_by_name('GetObservation')
```

```

>>> response = service.get_observation(responseFormat='http://www.opengis.net/om/2.0',↳
↳offerings=['urn:ioos:station:test:8'], observedProperties=['http://mmisw.org/ont/cf/
↳parameter/sea_water_temperature'], timeout=60)

```

```
>>> xml_tree = etree.fromstring(response)
```

```
>>> parsed_response = SOSGetObservationResponse(xml_tree)
```

```

>>> type(parsed_response)
<class 'owslib.swe.observation.sos200.SOSGetObservationResponse'>

```

```
>>> o=parsed_response.observations[0]
```

Value changes each call so can't be tested

```
## >>> o.get_result().value
```

```

>>> o.get_result().uom
'urn:ogc:def:uom:udunits:2:Cel'

```

This O&M structure of the results splits each point into an O&M object, resulting in 400 results

```
>>> len(parsed_response.observations)
400
>>> type(parsed_response.observations[0])
<class 'owslib.swe.observation.om.MeasurementObservation'>
```

Get observation for a specific offering (in this case corresponds to a station) and observed property (sea water temperature)

```
>>> response = service.get_observation(responseFormat='http://www.opengis.net/waterml/2.0
↳', offerings=['urn:ioos:station:test:8'], observedProperties=['http://mmisw.org/ont/cf/
↳parameter/sea_water_temperature'], timeout=60)
>>> xml_tree = etree.fromstring(response)
>>> parsed_response = SOSGetObservationResponse(xml_tree)
>>> type(parsed_response)
<class 'owslib.swe.observation.sos200.SOSGetObservationResponse'>
```

```
>>> len(parsed_response.observations)
20
>>> type(parsed_response.observations[0])
<class 'owslib.swe.observation.waterml2.MeasurementTimeseriesObservation'>
>>> type(parsed_response.observations[0].get_result())
<class 'owslib.swe.observation.waterml2.MeasurementTimeseries'>
```

```
>>> measurement_timeseries = parsed_response.observations[0].get_result()
>>> len(measurement_timeseries)
20
```

4.10 SensorML

Imports

```
>>> from tests.utils import resource_file
>>> from owslib.swe.sensor.sml import SensorML
>>> from dateutil import parser
>>> import pytz
```

Initialize

```
>>> xml = open(resource_file('sml_ndbc_station.xml'), 'rb').read()
>>> root = SensorML(xml)
>>> system = root.members[0]
```

```
>>> system.description
'Station metadata for 41012 - 40NM ENE of St Augustine, FL'
```

Contacts

```
>>> sorted(system.contacts.keys())
['urn:ogc:def:classifiers:OGC:contactType:operator',
↳ 'urn:ogc:def:classifiers:OGC:contactType:publisher']
```

```
>>> operators = system.get_contacts_by_role(
↳ 'urn:ogc:def:classifiers:OGC:contactType:operator')
>>> operators[0].role
'urn:ogc:def:classifiers:OGC:contactType:operator'
>>> operators[0].organization
'National Data Buoy Center'
>>> operators[0].country
'US'
```

```
>>> publishers = system.get_contacts_by_role(
↳ 'urn:ogc:def:classifiers:OGC:contactType:publisher')
>>> publishers[0].role
'urn:ogc:def:classifiers:OGC:contactType:publisher'
>>> publishers[0].organization
'National Data Buoy Center'
>>> publishers[0].country
'USA'
>>> publishers[0].phone
'228-688-2805'
>>> publishers[0].address
'Bldg. 3205'
>>> publishers[0].city
'Stennis Space Center'
>>> publishers[0].postcode
'39529'
>>> publishers[0].email
'webmaster.ndbc@noaa.gov'
>>> publishers[0].region
'MS'
```

Identification

```
>>> sorted(system.identifiers.keys())
['Long Name', 'Short Name', 'StationId']
```

```
>>> sid = system.get_identifiers_by_name('StationId')
>>> sid[0].name
'StationId'
>>> sid[0].definition
'urn:ioos:def:identifier:NOAA:stationID'
>>> sid[0].codeSpace
'http://sdf.ndbc.noaa.gov'
>>> sid[0].value
'urn:ioos:station:wmo:41012'
```

Classifiers

```
>>> sorted(system.classifiers.keys())
['Platform Type']
>>> classi = system.get_classifiers_by_name('Platform type')
>>> classi[0].name
'Platform Type'
>>> classi[0].definition
```

(continues on next page)

(continued from previous page)

```
'urn:ioos:def:classifier:NOAA:platformType'
>>> classi[0].codeSpace
'http://sdf.ndbc.noaa.gov'
>>> classi[0].value
'MOORED BUOY'
```

Documents

```
>>> system.documentation
[<owslib.swe.sensor.sml.Documentation ...>]
```

```
>>> doc = system.documentation[0].documents[0]
>>> doc.description
'Handbook of Automated Data Quality Control Checks and Procedures, National Data Buoy
↳ Center, August 2009'
>>> doc.format
'pdf'
>>> doc.url
'http://www.ndbc.noaa.gov/NDBCHandbookofAutomatedDataQualityControl2009.pdf'
```

History

```
>>> sorted(system.history.keys())
['deployment_start', 'deployment_stop']
```

```
>>> his = system.get_history_by_name('deployment_start')
>>> his
[<owslib.swe.sensor.sml.Event ...>]
>>> len(his)
2
```

```
>>> event = his[0]
>>> parser.parse(event.date).replace(tzinfo=pytz.utc).isoformat()
'2010-01-12T00:00:00+00:00'
>>> event.description
'Deployment start event'
>>> event.documentation[0].url
'http://sdfest.ndbc.noaa.gov/sos/server.php?service=SOS&request=DescribeSensor&
↳ version=1.0.0&outputformat=text/xml;subtype="sensorML/1.0.1"&
↳ procedure=urn:ioos:station:wmo:41012:20100112'
```

4.11 ISO

```
>>> from owslib.iso import *
>>> m=MD_Metadata(etree.parse('tests/resources/9250AA67-F3AC-6C12-0CB9-0662231AA181_iso.
↳ xml'))
>>> m.identification.topiccategory
'farming'
>>>
```

ISO Codelists:

Imports

```
>>> from tests.utils import resource_file
>>> from owslib.etree import etree
>>> from owslib.iso import CodelistCatalogue
```

Print testing the code lists

```
>>> e=etree.parse(resource_file('gmxCodelists.xml'))
>>> c=CodelistCatalogue(e)
>>> sorted(c.getcodelistdictionaries())
['CI_DateTypeCode', 'CI_OnLineFunctionCode', 'CI_PresentationFormCode', 'CI_RoleCode',
↪ 'DQ_EvaluationMethodTypeCode', 'DS_AssociationTypeCode', 'DS_InitiativeTypeCode', 'MD_
↪ CellGeometryCode', 'MD_CharacterSetCode', 'MD_ClassificationCode', 'MD_
↪ CoverageContentTypeCode', 'MD_DatatypeCode', 'MD_DimensionNameTypeCode', 'MD_
↪ GeometricObjectTypeCode', 'MD_ImagingConditionCode', 'MD_KeywordTypeCode', 'MD_
↪ MaintenanceFrequencyCode', 'MD_MediumFormatCode', 'MD_MediumNameCode', 'MD_
↪ ObligationCode', 'MD_PixelOrientationCode', 'MD_ProgressCode', 'MD_RestrictionCode',
↪ 'MD_ScopeCode', 'MD_SpatialRepresentationTypeCode', 'MD_TopicCategoryCode', 'MD_
↪ TopologyLevelCode', 'MX_ScopeCode']
```

```
>>> sorted(c.getcodedefinitionidentifiers('CI_RoleCode'))
['author', 'custodian', 'distributor', 'originator', 'owner', 'pointOfContact',
↪ 'principalInvestigator', 'processor', 'publisher', 'resourceProvider', 'user']
```

4.12 CRS Handling

```
>>> from owslib import crs
>>> c=crs.Crs('EPSG:4326')
>>> c.code
4326
>>> c.getcodeurn()
'urn:ogc:def:crs:EPSG::4326'
>>> c.getcodeuri1()
'http://www.opengis.net/def/crs/EPSG/0/4326'
>>> c.getcodeuri2()
'http://www.opengis.net/gml/srs/epsg.xml#4326'
>>> c=crs.Crs('urn:ogc:def:crs:EPSG::4326')
>>> c.authority
'EPSG'
>>> c.axisorder
'yx'
>>> c=crs.Crs('http://www.opengis.net/gml/epsg.xml#4326')
>>> c.code
4326
>>> c.axisorder
'yx'
>>> c=crs.Crs('urn:x-ogc:def:crs:EPSG:6.11:2192')
>>> c.axisorder
```

(continues on next page)

(continued from previous page)

```
'xy'
>>> c.code
2192
>>> c.version
'6.11'
>>> c=crs.Crs('http://www.opengis.net/def/crs/EPSG/0/4326')
>>> c.authority
'EPSG'
>>> c.code
4326
>>> c.axisorder
'yx'
>>> c=crs.Crs('PROJ4:+proj=lcc +lat_1=46.8 +lat_0=46.8 +lon_0=0 +k_0=0.99987742 +x_
↪_0=6000000 +y_0=22000000')
>>> c.authority
'PROJ4'
>>> c.code
'+proj=lcc +lat_1=46.8 +lat_0=46.8 +lon_0=0 +k_0=0.99987742 +x_0=6000000 +y_0=22000000'
```

4.13 Dublin Core

4.14 NASA DIF

4.15 FGDC

4.16 Swiss GM03

Imports

```
>>> from tests.utils import resource_file
>>> from owslib.etree import etree
>>> from owslib.gm03 import GM03
```

Print testing some metadata elements

```
>>> e = etree.parse(resource_file('gm03_example1.xml'))
>>> gm03 = GM03(e)
>>> gm03.header.version
'2.3'
>>> gm03.header.sender
'geocat.ch'
>>> hasattr(gm03.data, 'core')
False
>>> hasattr(gm03.data, 'comprehensive')
True
>>> len(gm03.data.comprehensive.elements)
13
>>> sorted(list(gm03.data.comprehensive.elements.keys()))
```

(continues on next page)

(continued from previous page)

```

['address', 'citation', 'contact', 'data_identification', 'date', 'extent', 'extent_
↳ geographic_element', 'geographic_bounding_box', 'identification_point_of_contact',
↳ 'keywords', 'metadata', 'metadata_point_of_contact', 'responsible_party']
>>> isinstance(gm03.data.comprehensive.date, list)
True
>>> len(gm03.data.comprehensive.date)
1
>>> gm03.data.comprehensive.metadata.file_identifier
'41ac321f632e55ceb0508a2cea5d9023fd12d9ad46edd679f2c275127c88623fb9c9d29726bef7c'
>>> gm03.data.comprehensive.metadata.date_stamp
'1999-12-31T12:00:00'
>>> gm03.data.comprehensive.metadata.language
'de'

```

Test TID searching

```

>>> gm03.data.comprehensive.metadata.tid
'xN6509077498146737843'
>>> search_tid = gm03.data.comprehensive.metadata.tid
>>> gm03.data.comprehensive.get_element_by_tid('404') is None
True
>>> gm03.data.comprehensive.get_element_by_tid(search_tid) is None
False
>>> search_tid2 = gm03.data.comprehensive.extent.data_identification.ref
>>> search_tid2
'xN8036063300808707346'
>>> gm03.data.comprehensive.get_element_by_tid(search_tid2) is None
False
>>> e = etree.parse(resource_file('gm03_example2.xml'))
>>> gm03 = GM03(e)
>>> gm03.data.comprehensive.geographic_bounding_box.extent_type_code
'false'
>>> gm03.data.comprehensive.geographic_bounding_box.north_bound_latitude
'47.1865387201702'
>>> gm03.data.comprehensive.geographic_bounding_box.south_bound_latitude
'47.1234508676764'
>>> gm03.data.comprehensive.geographic_bounding_box.east_bound_longitude
'9.10597474389878'
>>> gm03.data.comprehensive.geographic_bounding_box.west_bound_longitude
'9.23798212070671'

```

4.17 WMTS

Imports

```
>>> from tests.utils import scratch_file
```

Find out what a WMTS has to offer. Service metadata:

```
>>> from owslib.wmts import WebMapTileService
>>> wmts = WebMapTileService("http://map1c.vis.earthdata.nasa.gov/wmts-geo/wmts.cgi")
>>> wmts.identification.type
'OGC WMTS'
>>> wmts.identification.version
'1.0.0'
>>> wmts.identification.title
'NASA Global Imagery Browse Services for EOSDIS'
>>> bytearray(wmts.identification.abstract, 'utf-8')
bytearray(b'Near real time imagery from multiple NASA instruments')
>>> wmts.identification.keywords
['World', 'Global']
```

Service Provider:

```
>>> wmts.provider.name
'National Aeronautics and Space Administration'
```

```
>>> wmts.provider.url
'https://earthdata.nasa.gov/'
```

Available Layers:

```
>>> len(wmts.contents.keys()) > 0
True
>>> sorted(list(wmts.contents))[0]
'AIRS_CO_Total_Column_Day'
```

Fetch a tile (using some defaults):

```
>>> tile = wmts.gettile(layer='MODIS_Terra_CorrectedReflectance_TrueColor',
↳ tilematrixset='EPSG4326_250m', tilematrix='0', row=0, column=0, format="image/jpeg")
>>> out = open(scratch_file('nasa_modis_terra_truecolour.jpg'), 'wb')
>>> bytes_written = out.write(tile.read())
>>> out.close()
```

Test styles for several layers

```
>>> wmts.contents['MLS_S02_147hPa_Night'].styles
{'default': {'isDefault': True, 'title': 'default'}}
>>> wmts.contents['MLS_S02_147hPa_Night'].styles
{'default': {'isDefault': True, 'title': 'default'}}
```

Test a WMTS without a ServiceProvider tag in Capabilities XML

```
>>> wmts = WebMapTileService('http://data.geus.dk/arcgis/rest/services/
↳ OneGeologyGlobal/S071_G2500_OneGeology/MapServer/WMTS/1.0.0/WMTSCapabilities.xml')
```

Result:



4.18 WaterML

Using WaterML 1.1 for examples

Imports

```
>>> from tests.utils import resource_file
>>> from owslib.waterml.wml11 import WaterML_1_1 as wml
```

An example GetSites response (from cuahsi)

```
>>> f = open(resource_file('cuahsi_example_all_sites.xml'), 'rb').read()
>>> sites = wml(f).response
```

Can view the queryInfo structure for information about the query

```
>>> sites.query_info.creation_time
datetime.datetime(2009, 6, 12, 10, 47, 54, 531250, tzinfo=tzoffset(None, -25200))
>>> sites.query_info.notes
```

(continues on next page)

(continued from previous page)

```
['ALL Sites(empty request)']
>>> sites.query_info.criteria.method_called
'GetSites'
```

Get a list of codes for the sites returned

```
>>> codes = sites.site_codes
>>> sorted(codes)
[['10105900'], ['USU-LBR-Confluence'], ['USU-LBR-EFLower'], ['USU-LBR-EFRepeater'],
↳ ['USU-LBR-EFWeather'], ['USU-LBR-ExpFarm'], ['USU-LBR-Mendon'], ['USU-LBR-Paradise
↳'], ['USU-LBR-ParadiseRepeater'], ['USU-LBR-SFLower'], ['USU-LBR-SFUpper'], ['USU-
↳ LBR-Wellsville']]
```

Get the names of the sites

```
>>> sorted(sites.site_names)
['East Fork Little Bear River Radio Repeater near Avon, Utah', 'East Fork Little
↳ Bear River at Paradise Canal Diversion near Avon, Utah', 'Little Bear River Upper
↳ Weather Station near Avon, Utah', 'Little Bear River at McMurdy Hollow near
↳ Paradise, Utah', 'Little Bear River at Mendon Road near Mendon, Utah', 'Little
↳ Bear River at Paradise, Utah', 'Little Bear River below Confluence of South and
↳ East Forks near Avon, Utah', 'Little Bear River near Wellsville, Utah', 'Radio
↳ Repeater near Paradise, Utah', 'South Fork Little Bear River above Davenport
↳ Creek near Avon, Utah', 'South Fork Little Bear River below Davenport Creek near
↳ Avon, Utah', 'Utah State University Experimental Farm near Wellsville, Utah']
```

Get a site to view it in more detail

```
>>> site = sites[codes[0][0]]
>>> site.geo_coords
[(-111.946402, '41.718473')]
>>> site.latitudes
['41.718473']
>>> site.longitudes
['-111.946402']
>>> info = site.site_info
>>> info.notes
[]
>>> x = info.site_properties
>>> x == {'County': 'Cache', 'PosAccuracy_m': '1', 'State': 'Utah', 'Site Comments
↳': 'Located below county road bridge at Mendon Road crossing'}
True
>>> info.altname
>>> info.elevation
'1345'
```

An example GetSiteInfo response

```
>>> f = open(resource_file('cuahsi_example_siteinfo_multiple.xml')).read()
>>> sites = wml(f).response
```

```
>>> sites.query_info.criteria.method_called
'GetSiteInfo'
```

List codes and names of the sites

```
>>> codes = sites.site_codes
>>> sorted(codes)
[['USU-LBR-Mendon'], ['USU-LBR-Wellsville']]
>>> sorted(sites.site_names)
['Little Bear River at Mendon Road near Mendon, Utah', 'Little Bear River near_
↪Wellsville, Utah']
```

Get a site for a closer look

```
>>> site = sites[codes[1][0]]
>>> site.geo_coords
[(-111.917649, '41.643457')]
```

Get the (first) catalog of series for the site

```
>>> catalog = site[0]
```

Get a series from the catalog for a closer look

```
>>> series = catalog[3]
>>> series.properties
{}
>>> series.begin_date_time
datetime.datetime(2007, 11, 5, 14, 30)
>>> series.end_date_time
datetime.datetime(2008, 4, 5, 20, 30)
>>> series.method_id
'23'
>>> series.source_id
'2'
>>> series.value_count
'7309'
>>> series.value_type
'Field Observation'
>>> series.name
'Turbidity'
>>> series.organization
'Utah State University Utah Water Research Laboratory'
```

List variable names and codes

```
>>> sorted(site.variable_names)
['Battery voltage', 'Gage height', 'Oxygen, dissolved', 'Oxygen, dissolved percent_
↪of saturation', 'Phosphorus, total as P', 'Phosphorus, total as P, filtered',
↪'Solids, total Suspended', 'Specific conductance, unfiltered', 'Temperature',
↪'Turbidity', 'pH, unfiltered']
>>> sorted(site.variable_codes)
['USU10', 'USU13', 'USU3', 'USU32', 'USU33', 'USU34', 'USU35', 'USU36', 'USU39',
↪'USU4', 'USU40', 'USU41', 'USU5', 'USU6', 'USU7', 'USU8', 'USU9']
```

Get a variable by its code


```
>>> variable = site['USU7']
>>> variable.variable_name
'Turbidity'
>>> variable.properties
{}
>>> variable.speciation
'Not Applicable'
>>> variable.unit.name
'nephelometric turbidity units'
>>> variable.time_scale.unit.name
'second'
```

Example GetValues response

```
>>> f = open(resource_file('cuahsi_example_get_values.xml')).read()
>>> series = wml(f).response
```

```
>>> series.query_info.criteria.method_called
'GetValuesForASite'
```

List the names of the series returned (usually None)

```
>>> series.series_names
[None, None, None, None, None, None, None, None, None, None, None]
```

List the variables and their codes

```
>>> sorted(series.variable_names)
['Battery Voltage', 'Battery voltage', 'Gage height', 'Temperature', 'Turbidity']
>>> codes = series.variable_codes
>>> sorted(codes)
['SDSC45', 'USU10', 'USU11', 'USU12', 'USU13', 'USU3', 'USU4', 'USU5', 'USU6', 'USU7',
 'USU8', 'USU9']
```

Get variables by code

```
>>> var = series.get_series_by_variable(var_code='USU4')
```

Get the first values set from the first variable retrieved from the code

```
>>> vals = var[0].values[0]
```

List (in tuple) the dates and their corresponding measurements

```
>>> sorted(vals.get_date_values())
[(datetime.datetime(2005, 8, 5, 0, 0), '34.53'), (datetime.datetime(2005, 8, 5, 0, 30), '37.12'), (datetime.datetime(2005, 8, 5, 1, 0), '35.97'), (datetime.datetime(2005, 8, 5, 1, 30), '35.78'), (datetime.datetime(2005, 8, 5, 2, 0), '35.68'), (datetime.datetime(2005, 8, 5, 2, 30), '36.08'), (datetime.datetime(2005, 8, 5, 3, 0), '37.8'), (datetime.datetime(2005, 8, 5, 3, 30), '37.93'), (datetime.datetime(2005, 8, 5, 4, 0), '38.88'), (datetime.datetime(2005, 8, 5, 4, 30), '37.34'), (datetime.datetime(2005, 8, 5, 5, 0), '35.15'), (datetime.datetime(2005, 8, 5, 5, 30), '35.96'), (datetime.datetime(2005, 8, 5, 6, 0), '35.62'), (datetime.datetime(2005, 8, 5, 6, 30), '34.72'), (datetime.datetime(2005, 8, 5, 7, 0), '34.7'), (datetime.datetime(2005, 8, 5, 7, 30), '33.54'), (datetime.datetime(2005, 8, 5, 8, 0), '34.98'), (datetime.datetime(2005, 8, 5, 8, 30), '31.65'), (datetime.datetime(2005, 8, 5, 9, 0), '32.49'), (datetime.datetime(2005, 8, 5, 9, 30), '32.78'), (datetime.datetime(2005, 8, 5, 10, 0), '30.58'), (datetime.datetime(2005, 8, 5, 10, 30), '32.8'), (datetime.datetime(2005, 8, 5, 11, 0), '31.83'), (datetime.datetime(2005, 8, 5, 11, 30), '30.71'), (datetime.datetime(2005, 8, 5, 12, 0), '30.82'), (datetime.datetime(2005, 8, 5, 12, 30), '29.72'), (datetime.datetime(2005, 8, 5, 13, 0), '29.72')]
```

(continued from previous page)

Example GetVariables response

```
>>> f = open(resource_file('cuahsi_example_get_variables.xml')).read()
>>> varis = wml(f).response
```

Just a list of variables

```
>>> codes = varis.variable_codes
>>> sorted(codes)
['SDSC45', 'USU10', 'USU11', 'USU12', 'USU13', 'USU14', 'USU15', 'USU16', 'USU17',
↪ 'USU18', 'USU19', 'USU20', 'USU21', 'USU22', 'USU23', 'USU24', 'USU25', 'USU26',
↪ 'USU27', 'USU28', 'USU29', 'USU3', 'USU30', 'USU31', 'USU32', 'USU33', 'USU34',
↪ 'USU35', 'USU36', 'USU37', 'USU38', 'USU39', 'USU4', 'USU40', 'USU41', 'USU42',
↪ 'USU43', 'USU5', 'USU6', 'USU7', 'USU8', 'USU9']
>>> sorted(varis.variable_names)
['Barometric pressure', 'Battery Voltage', 'Battery voltage', 'Discharge', 'Gage_
↪ height', 'Oxygen, dissolved', 'Oxygen, dissolved percent of saturation',
↪ 'Phosphorus, total as P', 'Phosphorus, total as P, filtered', 'Precipitation',
↪ 'Radiation, incoming shortwave', 'Relative humidity', 'Solids, total Suspended',
↪ 'Specific conductance, unfiltered', 'Temperature', 'Turbidity', 'Wind direction',
↪ 'Wind speed', 'pH, unfiltered']
>>> var = varis[codes[10]]
>>> var.variable_name
'Gage height'
>>> var.no_data_value
'-9999'
>>> var.properties
{}
>>> var.unit.name
'international foot'
```

4.19 OGC OWS Context 1.0.0 Atom CML and GeoJSON Encoding (alpha/under-review)

The OGC OWS Context implementation in OWSlib is currently in alpha and under review, and will still be improved. Please get in touch if you (want to) use it and provide feedback on how more comfortable it should be (especially handling geometries and dates in different encodings) and if it doesn't treat your "standards-compliant" OWS Context document right. Greatly appreciated :-)

Basic reading/parsing of OGC Web Services Context Documents (OWS Context) in OWC Atom 1.0.0 Encoding and OWC GeoJSON 1.0.0 Encoding Standards:

Imports

```
>>> import os
>>> from owslib.owscontext.core import OwContext
>>> from tests.utils import resource_file
```

Test OWC Atom XML Read

```

>>> f = open(resource_file(os.path.join('owc_atom_examples', 'wms_meris.xml')), 'rb').
↳ read()
>>> owc = OwContext.from_atomxml(f)
>>> owc.title
'Satellite Image and Countries borders Over France and England'
>>> res = owc.resources[0]
>>> res.id
'http://geoserver.sourceforge.net/html/index.php/'
>>> res.title
'MER_RR__2PRLRA20120406_102429_000026213113_00238_52838_0211.N1.tif'
>>> off = res.offerings[0]
>>> off.offering_code
'http://www.opengis.net/spec/owc-atom/1.0/req/wms'
>>> op = off.operations[0]
>>> op.operations_code
'GetCapabilities'
>>> op.request_url
'http://ows.genesi-dec.eu/geoserver/385d7d71-650a-414b-b8c7-739e2c0b5e76/wms?SERVICE=WMS&
↳ VERSION=1.3.0&REQUEST=GetCapabilities'

```

Test OWC GeoJSON Read

```

>>> f = open(resource_file(os.path.join('owc_geojson_examples', 'owc1.geojson')), 'rb').
↳ read().decode('utf-8')
>>> owc = OwContext.from_json(f)
>>> # because json works on pure strings, the doctest unicode results for Python 2.7
↳ would cause errors, I add print()
>>> print(owc.title)
WPS 52North example
>>> res = owc.resources[0]
>>> print(res.id)
http://geoprocessing.demo.52north.org:8080/wps/WebProcessingService/process1/
>>> print(res.title)
WPS 52 north
>>> off = res.offerings[0]
>>> print(off.offering_code)
http://www.opengis.net/spec/owc-geojson/1.0/req/wps
>>> op = off.operations[0]
>>> print(op.operations_code)
GetCapabilities
>>> print(op.request_url)
http://geoprocessing.demo.52north.org:8080/wps/WebProcessingService?SERVICE=WPS&
↳ VERSION=1.0.0&REQUEST=GetCapabilities

```

additionally, possibility to construct OWS Context documents from scratch, and then write/serialise into OWC Atom 1.0.0 Encoding or OWC GeoJSON 1.0.0 Encoding Standards:

```

>>> from owslib.owscontext.core import OwResource, OwContext
>>> myContext=OwContext(id='http://my.url.com/context/id/1',
    update_date='2017-11-02T15:24:24.446+12:00',
    title='Awesome new Context doc')
>>> myContext.rights='Creative Commons 4.0 BY'
>>> myEntry=OwResource(id='http://my.url.com/resource/demo-feature-1',

```

(continues on next page)

(continued from previous page)

```
        update_date='2017-11-02T15:24:24.446+12:00',
        title='This is a feature')
>>> contributor={'name': 'Alex K',
                 'email': None,
                 'uri': 'https://allixender.blogspot.com'}
>>> myEntry.authors.append(contributor)

>>> # ... here also continue to build your OGC data offerings, e.g. WMS GetMap etc.

>>> myContext.resources.append(myEntry)
>>> myContext.to_json()
>>> myContext.to_atomxml()
```

DEVELOPMENT

The OWSLib wiki is located at <https://github.com/geopython/OWSLib/wiki>

The OWSLib source code is available at <https://github.com/geopython/OWSLib>

You can find out about software metrics at the OWSLib OpenHub page at <https://www.openhub.net/p/OWSLib>.

5.1 Testing

```
$ python setup.py test
```

Or...

```
# install requirements
$ pip install -r requirements.txt
$ pip install -r requirements-dev.txt # needed for tests only

# run tests
python -m pytest

# additional pep8 tests
pep8 owslib/wmts.py
```


6.1 Mailing Lists

OWSLib provides users and developers mailing lists. Subscription options and archives are available at <https://lists.osgeo.org/mailman/listinfo/owslib-users> and <https://lists.osgeo.org/mailman/listinfo/owslib-devel>.

6.2 Submitting Questions to Community

To submit questions to a mailing list, first join the list by following the subscription procedure above. Then post questions to the list by sending an email message to either owslib-users@lists.osgeo.org or owslib-devel@lists.osgeo.org.

6.3 Searching the Archives

All Community archives are located in <https://lists.osgeo.org/pipermail/owslib-users/> <https://lists.osgeo.org/pipermail/owslib-devel/>

6.4 Metrics

You can find out about software metrics at the OWSLib [OpenHub](#) page.

6.5 IRC

As well, visit OWSLib on IRC on [#geopython](#) at [freenode](#) for realtime discussion.

LOGGING

OWSLib logs messages to the ‘owslib’ named python logger. You may configure your application to use the log messages like so:

```
import logging
owslib_log = logging.getLogger('owslib')
# Add formatting and handlers as needed
owslib_log.setLevel(logging.DEBUG)
```


LICENSE

Copyright (c) 2006, Ancient World Mapping Center All rights reserved.

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

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

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

CREDITS

- Sean Gillies <sgillies@frie.com>
- Julien Anguenot <ja@nuxeo.com>
- Kai Lautaportti <kai.lautaportti@hexagonit.fi>
- Dominic Lowe <D.Lowe@rl.ac.uk>
- Jo Walsh <jo@frot.org>
- Tom Kralidis <tomkralidis@gmail.com>
- Jachym Cepicky <jachym.cepicky@gmail.com>
- Luca Cinquini <luca.cinquini@jpl.nasa.gov>
- Brad Hards <bradh@frogmouth.net>
- Christian Ledermann <christian.ledermann@gmail.com>
- Sean Cowan <scowan@asascience.com>
- Kyle Wilcox <wilcox.kyle@gmail.com>
- Angelos Tzotsos <gcpp.kalxas@gmail.com>

CHANGES

10.1 0.21.0 (2020-12-09)

A full list of commits for 0.21.0 can be found at:

<https://github.com/geopython/OWSLib/commits/0.21.0>

- OGC API: Added support for Coverages (#699)
- WFS: Added POST support to WFS GetFeature (#706)
- WCS: Allow user to specify timeout on WCS GetCoverage calls (#714)
- WPS: fixed bounding-box (#719)
- Various fixes: #695, #707, #702, #716, #718, #722, #691, #720

10.2 0.20.0 (2020-06-05)

This release provides initial support for the draft OGC API - Records standard.

A full list of commits for 0.20.0 can be found at:

<https://github.com/geopython/OWSLib/commits/0.20.0>

- WFS: make wfs:FeatureTypeList optional for 1.1 and 2.0 (#673)
- OGC API - Records: initial draft implementation (#679, #693)
- WPS: add support for retrieving data from local filesystem (huard, #681)
- WMTS: add support for boundingboxes (kordian-kowalski, #687)
- Authentication: Enable switching off SSL verification (Samweli, #685)

10.3 0.19.2 (2020-03-13)

This release is an update. It drops Python 3.5 support and adds language support to the WPS module.

A full list of commits for 0.19.2 can be found at:

<https://github.com/geopython/OWSLib/commits/0.19.2>

- Dropped Python 3.5 support (#659).
- Fix pyproj deprecation (only use pyproj>=2) (#661).

- WPS: Added support for multiple languages (#654, #655).
- OGC API: fix api media type handling (#653).
- WMTS: fix cartopy examples (#656).
- Tests: fix wms tests (#657).
- WCS: added WCS code example do documentation (#658).
- WCS: fix params list (#663).

10.4 0.19.1 (2020-01-29)

This release is an update with a fix for Python 3.8 installation.

A full list of commits for 0.19.1 can be found at:

<https://github.com/geopython/OWSLib/commits/0.19.1>

- Fixed Python 3.8 installation (#635, ocefpaf)
- Refactored OGC API (#641, tomkralidis)
- Add *python_requires* to prevent pip installing this on Python 2.x (#634, craigds)
- Tests: use HEAD instead of GET in service_ok (#651, roel)
- Tests: replaced service URLs with working versions (#648, roel)
- WFS: Fix WFS get_schema when typename doubles as attributename (#647, roel)
- WCS: Removed redundant check on logging level (#646, johanvdw)
- WFS3: renamed time parameter to datetime (#639, mattfung)
- WFS: Added required fields to wfs schema (#628, Alexandre27)
- WCS: added resolution and size params to WCS GetCoverage requests (#633, dukestep)
- DOCS: updated WMS docs (#649, pathmapper)

10.5 0.19.0 (2019-11-14)

Python 2 support has been removed. Users are strongly encouraged to upgrade to the latest version of OWSLib and Python 3.

A full list of commits for 0.19.0 can be found at:

<https://github.com/geopython/OWSLib/commits/0.19.0>

- ALL: 2 support removed (cehbrecht et. al.)
- OWS
- safeguard `ProviderSite/@href` (jannefleischer)
- safeguard bbox parsing (walkermatt)
- WMS: support ScaleDenominator
- WMTS: add timeout (jachym)
- OGC API

- Features updates following specification
- TMS: fix broken constructor (justb4)
- ALL: pass HTTP headers for WMS/WFS/WMTS/TMS (justb4)
- ALL: test fixes/updates

10.6 0.18.0 (2019-06-24)

This release includes initial support for the evolving OGC API - Features standard (aka WFS 3), which represents a clean break from the traditional design patterns of OGC interface specifications. This release also includes a refactoring of authentication support into a common approach for all parts of the codebase to use. Thanks to Eric Spitler for this enhanced functionality!

Users are strongly encouraged to use OWSLib via Python 3 as Python 2 support (Travis, python-six) will be removed in the future.

A full list of commits for 0.18.0 can be found at:

<https://github.com/geopython/OWSLib/commits/0.18.0>

- NEW: WFS 3 initial implementation
- NEW: add Authentication framework (eric-spitler)
- WPS:
 - add process properties, percentCompleted (cehbrecht)
 - add reference attributes (enolfc)
 - add support for multi process processes (huard)
- OWS: add support for crs and dimension (saimeCS)

10.7 0.17.1 (2019-01-12)

Bugfix release for issues in WPS and WMS.

A full list of commits for 0.17.1 can be found at:

<https://github.com/geopython/OWSLib/commits/0.17.1>

10.8 0.17.0 (2018-09-04)

This release provides numerous fixes, enhancements and re-engineering of OWSLib's test framework.

A full list of commits for 0.17.0 can be found at:

<https://github.com/geopython/OWSLib/commits/0.17.0>

- NEW: OWS Context implementation (#483 allixender)
- ISO:
 - Added MD_LegalConstraints to uselimination xpath (m431m)
 - Fix ISO metadata parsing for empty gmd:featureCatalogueCitation. (Roel)

- **OWS:**
 - Improve remote metadata parsing (Roel)
 - Allow the lack of optional ows:ServiceProvider (mhugo)
- **WPS:**
 - add headers, cert options (cehbrecht)
 - add lineage to execute (cehbrecht)
- **WMTS/TMS:** replaced ServiceMetadata (cehbrecht)
- **SOS:** fix encoding error (cehbrecht)
- **tests:** move away from doctests (#339 cehbrecht)
- **overall codebase:** move from pep8 to flake8
- **Support for WCS 2.0.0 and 2.0.1** (#430, thanks @doclements)

10.9 0.16.0 (2017-12-21)

- **drop Python 2.6 support**
- **WFS:** get schema auth params (karakostis)
- **WFS:** add sortBy to GetFeature requests (drnextgis)
- **CSW:** add ows namespace to bounding box queries
- **CSW:** add feature catalogue support parsing
- **CRS:** support proj.4 CRS definitions (orhygine)
- **fix namespaces** (jsanchezfr)
- **ISO GM03:** fix bounding box handling

10.10 0.15.0 (2017-09-13)

- **WFS:**
 - add doseq to WFS request query urlencode
 - handle non-existing bounding boxes in feature types
- **SOS:**
 - add support for authentication
- **WMTS:**
 - add support for styles
- **ISO:**
 - add support for gmd:locale
- **GM03:**
 - add support for GM03 ISO metadata profile
- **CRS:**

- catch invalid CRS codes
- **WMS:**
 - fix time dimension handling in Capabilities
- **SWE:**
 - various bug fixes
- **WPS:**
 - fix WPS DescribeProcess issue on DataType
 - fixed bbox lower/upper_corner conversion
 - added a test for wps BoundingBoxDataInput
 - added BoundingBoxDataInput and fix boundingbox parsing
- **Misc:**
 - fix double && in URL requests
 - add util.clean_ows_url function to remove basic service parameters from OWS base URLs

10.11 0.14.0 (2017-01-12)

- WFS: add authentication (@pmauduit)
- WFS: fix parameter names for WFS2
- OWS: implement updateSequence support
- CSW: fix ref bug in CSW-T workflows
- WCS: fix 1.0.0 Capabilities OWS namespace handling

10.12 0.13.0 (2016-09-24)

- general: Handle servers that give 500 errors better (@davidread)
- WMS: 1.3.0 support @roomthily / @b-cube
- WMS: add WMS request property to cache request URL, add service parameter
- OWS: add ows.ServiceIdentification.versions, fix ref in ows.ServiceIdentification.profiles

10.13 0.12.0 (2016-09-12)

- OWS: Support OWS Constraints and Parameters
- SOS/WaterML: handle WaterML 2.0 updates and SOS decoder
- Add username and password arguments to WFS class constructors

10.14 0.11.0 (2016-04-01)

- CSW: fix outputschema setting when raw XML is specified
- ISO:
 - parsing anchor for abstract and lineage fields added (madi)
 - added support for spatialRepresentationType (pmdias)
 - add MD_Keywords class (pmdias)
 - fix md.languagecode to come from the codeListValue attribute (pmdias)
- WFS: add get_schema method for DescribeFeatureType parsing (jachym)
- WMS: do not assume parent layers should be queryable if 1..n of their children is
- WMTS: fix parsing when ServiceProvider does not exist
- WPS:
 - fix bbox type, parsing bbox output (cehbrecht)
 - add support for bbox data and more robust literal data parsing (jachym)

10.15 0.10.0 (2015-11-11)

- GM03: add support for GM03 metadata
- WPS: add fix for optional Abstract

10.16 0.9.2 (2015-09-23)

- etree: add convenience function to report which etree is used
- WMS: add GetFeatureInfo support (JuergenWeichand)
- WMS: add a children attribute to ContentMetadata to handle WMS nested layers (Jenselme)
- WMTS: add support for restful only WMTS (JuergenWeichand)
- pass headers to requests (ayan-usgs)

10.17 0.9.1 (2015-09-03)

- etree: Fix incorrect lxml ParseError import (daf)
- CRS: make crs hashable (QuLogic)
- WPS:
 - statuslocation bugfix (dblodgett-usgs)
 - various bugfixes, tests and examples (cehbrecht)
- WFS:
 - fix WFS 2.0 stored queries bugfix (JuergenWeichand)

- add docs for WFS 1.1/2.0 (JuergenWeichand)
- ISO: ignored empty gmd:identificationInfo elements (menegon)

10.18 0.9.0 (2015-06-12)

- Python 3 compatibility (numerous contributors!)
- CSW:
 - fix Capabilities parsing when ows:ServiceProvider is empty
 - fix GetRecordById URL
- WCS: add support for 1.1.1 (ldesousa)
- ISO:
 - add support for gmd:MD_ReferenceSystem (kalxas)
 - safeguard vars (dblodgett-usgs)
- SOS: add sa namespace, add procedure as optional parameter (ict4eo)

10.19 0.8.13 (2015-02-12)

- SOS: fix var reference blocker (ocefpaf)
- various Python 3 enhancements

10.20 0.8.11 (2014-12-17)

- WMTS: add/fix vendor kwarg handling (rhattersley)
- WMS: add ScaleHint support (SiggyF)
- FES: add srsName support for gml:Envelope, add filter to string support
- WFS: add timeout support (selimnairb), add support for startindex
- fix/cleanup tests

10.21 0.8.10 (2014-10-13)

- CSW: fix bad URL being sent to GetRecords
- SOS: add timeout support (lukecampbell)
- WPS: add logging (dblodgett-usgs)
- WFS: ignore comments when parsing (Samuli Vuorinen)
- tests: add support for logging
- LICENSE: update reference (johanvdw)

10.22 0.8.9 (2014-09-24)

- CSW: support `gmi:MI_Metadata` as `gmd:MD_Metadata` when parsing results (@FuhuXia)
- CSW: add support for basic authentication
- ISO: add support for instantiation of `MD_Metadata` objects (@kalxas)
- ISO: add support for `CI_ResponsibleParty` as a responsible role attribute (@milokmet)
- ISO: add title support for `SV_ServiceIdentification` (@dblodgett-usgs)
- SOS: add 'om' back to namespace list (@ict4eo)
- util: add support for race conditions for WPS (@TobiasKipp)

10.23 0.8.8 (2014-07-05)

- CSW: use URLs as defined in GetCaps for CSW operations (@kwilcox)
- CSW: fix `GetRecordById` (@kwilcox)
- CSW: use default CSW URL when initialized with `skip_caps=True`
- WMTS: Allow vendor-specific args in WMTS tile requests (@rhattersley)
- ISO: allow for `MD_Metadata` to be initialized as empty, supporting export to XML functionality (@kalxas)
- ISO: add support for `gmd:RS_Identifier` needed by INSPIRE (@kalxas)
- numerous unit test / build fixes and cleanups

10.24 0.8.7 (2014-05-02)

- WPS: add method to write output to disk (@ldesousa)
- CSW: add method to get operations by name
- CSW: responses now maintain order using `OrderedDict`
- CSW: ensure namespace is declared for `GetRecords` `typeName` values in request (@kwilcox)
- SOS: fix error detections (@daf)
- ISO: fix xpath for selecting `gmd:thesaurusName` (@menegon)
- add timeouts to HTTP functions (@iguest)
- FES: add `matchCase` to `ogc:PropertyIsLike`
- logging: add Null handler to not write files randomly (@kwilcox)
- WFS: add `GetFeature` `outputformat` support (@kwilcox, @rsignell-usgs)
- ISO: support GML 3.2 extent handling
- numerous unit test / build fixes and cleanups

10.25 0.8.3 (2014-01-10)

- allow CSW URLs to be requested as unicode or string (@rclark)
- support multiple gmd:extent elements (@severo)
- support WMS default time position (@vicb)
- fix SOS GetCapabilities support (@kwilcox)
- support missing CSW nextRecord (@davidread)
- use child layers for WMS duplicates
- numerous unit test fixes and cleanups

10.26 0.8.0 (2013-09-11)

- Support for WaterML 1.0 and 1.1 (thanks @kwilcox and @CowanSM)
- drastically improved CSW getrecords support (owslib.csw.CatalogueServiceWeb.getrecords2, which will eventually replace owslib.csw.CatalogueServiceWeb.getrecords, which is now deprecated) (thanks @kwilcox and @rsignell-usgs input)
- fix owslib.csw.CatalogueServiceWeb to use HTTP GET for GetCapabilities and GetRecordById (thanks @rsignell-usgs for input)
- numerous test fixes
- support owslib.iso.MD_Metadata scanning of multiple extents (thanks @severo)
- add WMS elevation support in Capabilities (thanks @mhermida)
- travis-ci setup (thanks @brianmckenna)
- Support for TMS (thanks @cleder)
- updated build packages (thanks @kalxas)
- numerous bug fixes

10.27 0.7 (2013-02-18)

- Support for SOS 1.0.0, SOS 2.0.0, SensorML (thanks @kwilcox)
- Support for TMS (thanks @cleder)
- numerous bug fixes

10.28 0.6 (2012-12-22)

- Support for WMTS (thanks @bradh)
- **packaging support (thanks @kalxas) for:**
 - openSUSE
 - Debian
- addition of `owslib.__version__`
- **ISO support:**
 - multiple `gmd:identificationInfo` elements
 - `gmd:distributorInfo` elements
- **WMS**
 - read additional Layer attributes (thanks @elemoine)
- numerous bug fixes

10.29 0.5 (2012-06-15)

- **Support for the following parsers:**
 - WPS 1.0.0
 - WFS 1.1.0
 - CRS handling: URNs, URIs, EPSG:xxxx style
- `etree.py` looks for `lxml.etree` first now
- catch WMS service exceptions on `GetCapabilities`
- CSW exceptions are now Pythonic

10.30 0.4 (2011-10-02)

- **Support for the following parsers:**
 - CSW 2.0.2
 - OWS Common 1.0.0, 1.1.0, 2.0.0
 - Filter Encoding 1.1.0
 - ISO 19115:2003
 - FGDC CSDGM
 - NASA DIF
 - Dublin Core
 - WFS 2.0
 - WCS 1.1
- New SCM/bug/mailling list infrastructure

- Sphinx documentation

10.31 0.3 (2008-05-08)

- WCS support.
- Support for basic authorization in WMS requests (#107).

10.32 0.2.1 (2007-08-06)

- Added support for Python 2.5.
- Fixed ticket #105: Don't depend on Content-length in the http headers for getfeature.

10.33 0.2.0 (2007-02-01)

- Change license to BSD.
- Added service contact metadata.

10.34 0.1.0 (2006-10-19)

- New and improved metadata API.
- Wrappers for GetCapabilities, WMS GetMap, and WFS GetFeature requests.
- Doctests.

10.35 0.0.1 (2006-07-30)

- Brought OWSLib up out of the PCL trunk into its own space.
- Updated the testing frameworm.
- Initial test coverage:

Table 1: Test Coverage

Name	Stmts	Exec	Cover	Missing
wms	105	68	64%	36, 41-48, 61-63, 114-118, 125-155, 172, 203-205
wfs	74	69	93%	146, 166, 199-201
wmc	111	0	0%	33-220
TOTAL	290	137	47%	