pyodv

Python toolbox for handling ODV ascii format from OceanDataView adopted by SeaDataNet.
Please also see our Matlabtoolbox odv for ODV files.

Getting the toolbox

The free and open source python ODV toolbox can be downloaded from `https://svn.oss.deltares.nl/repos/openearthtools/trunk/python/OpenEarthTools/openearthtools/io/pyodv/` when you Join OpenEarth, download the tools with SubVersion and add them to your python path.

ODV reading

Loading, merging and parsing ODV sparse file

For setting up WPS for remote processing of collections of ODV files a python io library `pyodv` has been created. Within this library (as a folder), the file pyodv.py contains the ODV class, which contains useful methods for ODV collections loading, merging and parsing. Please also see our MATLAB toolbox odv for ODV files.

NB. If you are working within `pyodv` folder, `pyodv.*` is not needed. This is valid here and after along the following tutorial.

```
ODV      = pyodv.ODV.fromfile('test01.txt')
```

An overview of the contents (spatial extent, temporal range, parameters) of the `Odv` object can be displayed with

```
print(ODV)
```

due to the `str` and `html` function inside the `Odv` object, it will display something like this

```
filename           | myfile.txt
data_type          | profile
timeCoverage_min   | 2002-03-23T06:10:00.000
timeCoverage_max   | 2002-03-23T06:10:00.000
geospatial_lon_min | -6.997
geospatial_lon_max | -6.997
geospatial_lat_min | 46.434
geospatial_lat_max | 46.434
rows               | 9524
cols               | 22 with names:

  # |           SDN name |      SDN units |                        local name
| local units
--- + ------------------ + -------------- + ------------------------------
+ --->
000 |                    |                |                            Cruise
|
001 |                    |                |                           Station
```

```
|
002 |                     |                |                          Type
|
003 |                     |                |       yyyy-mm-ddThh:mm:ss.sss
|
004 |                     |                |                     Longitude
| degrees_east
005 |                     |                |                      Latitude
| degrees_north
006 |                     |                |                  LOCAL_CDI_ID
|
007 |                     |                |                     EDMO_code
|
008 |                     |                |                    Bot. Depth
| m
009 | SDN:P011::PRESPS01 | SDN:P061::UPDB |                      PRESSURE
| dbar
010 |                     |                |                  QV:SEADATANET
|
011 | SDN:P011::TEMPS901 | SDN:P061::UPAA |                          T90
| degC
012 |                     |                |                  QV:SEADATANET
|
013 | SDN:P011::PSALPR02 | SDN:P061::UUUU |                     Salinity
| PSU
014 |                     |                |                  QV:SEADATANET
|
015 | SDN:P011::CPHLPM01 | SDN:P061::UGPL |                   fluorescence
| ugr/l
016 |                     |                |                  QV:SEADATANET
|
017 | SDN:P011::POPTDR01 | SDN:P061::UPCT |                Trans_red_25cm
| %
018 |                     |                |                  QV:SEADATANET
|
019 | SDN:P011::POTMCV01 | SDN:P061::UPAA |                          POTM
| degC
020 |                     |                |                  QV:SEADATANET
|
```

```
021 | SDN:P011::SIGTPR01 | SDN:P061::UKMC |                          Density
| kg/m3
```

Alternatively, for a single parameter, functions like `odvspar2*()` can help you to handle your odv parameter object as a pandas Dataframe or json.

Loading, merging and parsing ODV collections from a Spatial Database

Lately, Spatial DB's like PostGIS on PostgreSQL have been increasingly used to handle large collections of ODV files. A mixed approach to store and parse data in and from the Spatial DB is the use of:

1. python psycopg2 package (PostGreSQL + Python)
2. SQLAlchemy's Object Relational Mapper (ORM)

OpenEarth general-scope function sqlfunctions.py makes extensive use of the first package. On the other hand, OpenEarth odv2orm*.py function s have strong dependencies with the second model. The advantage of the first method is to run a SQL query string as inputs. The second methods provides user-defined python classes to represent database tables and python class objects to represent rows in those tables, which means working with Python objects instead of SQL queries. The latter method is based on odv2orm_model, odv2orm_initialize, odv2orm_populate, odv2orm_query, to respectively define an object-relational model, initiate the database, populate the database tables with rows, query the database. As with the odv sparse file, an object from pyodv.Odv class is returned using the class methods.

ODV plotting

The `ODV` object (retried from DB or odv sparse file) can be viewed as html `ODV.html()` or as formatted command line text `print(ODV)`. Several plotting functions are available. In general, functions look like `pyodv.odv2*(file_name,ODV_object,parameter_name,z_name)`, where `(file_name,ODV_object,parameter_name,z_name)`.These four parameters represent the minimum number of input arguments. file_name is the output file name; ODV_object is the fore-mentioned pyodv.ODV class object; parameter_name and `z_name` are respectively the `P01` name of the chemical/physical parameter and depth definition, both according to BODC vocabulary. Last obligatory input is made necessary by the presence of several depth definitions based on Length/Pressure. Optional arguments are: parameter limits, z limits, time interval, Matplotlib colormap string, parameter in a log10 scale, uniform color string, marker type, markersize, alpha value. Further information is contained in python function headers and `args` handling.

Trajectory data can be plotted as a map in .png and .kmz images (circles or columns), with x: Lon, y: Lat, cbar: Parameter. Ldb info is provided via a .nc on the `static` folder.

```
pyodv.odv2map           ('test01.png',ODV,cname,zname)
pyodv.odv2mapkmz        ('test01.kmz',ODV,cname,zname)
pyodv.odv2mapcolumnskmz ('test02.kmz',ODV,cname,zname)
```

Plotting a vertical profile, with x: Parameter, y: Depth, cbar: Time.

```
pyodv.odv2profile('test01.png',ODV,cname,zname)
```

Plotting a timeseries, with x: Time, y: Parameter, cbar: Depth.

```
pyodv.odv2timeseries('test01.png',ODV,cname,zname)
```

Plotting a timeseries of vertical profiles, with x: Time, y: Depth, cbar: Parameter.

```
pyodv.odv2timeprofile('test01.png',ODV,cname,zname)
```

pydov as WPS

On our test server we host pyodv with some server-side data files as WPS. On the server the pyodv WPS wrappers are hosted here

```
/var/lib/wsgi/wps/processes
```

whereas a copy of the pyodv toolbox is here

```
/var/lib/wsgi/wps/processes/pyodv
```

It is deployed with NGINX as web server, and uwsgi as application server, see Setting up pyWPS in combination with uwsgi and nginx. As local server on a Windows environment, a WPS server can be installed following this tutorial.

The WPS can be accessed on a test server via:

```
http://dtvirt5.deltares.nl/wps?Request=GetCapabilities&Service=WPS
```

We designed WPS for pyodv similar to the WxS services, please see pyWPSodv.

Acknowlegdements

This toolbox is being developed as part of EMODnet chemistry 2.