GDAL

From Python GIS

GDAL/OGR - Geospatial Data Abstraction Library GDAL (http://www.gdal.org/) is a powerful C/C++ library for handling various raster (GDAL) and vector (OGR) formats

GDAL Python bindings are allowing you to access the full functional range of GDAL/OGR in Python.

Contents

Installation sources

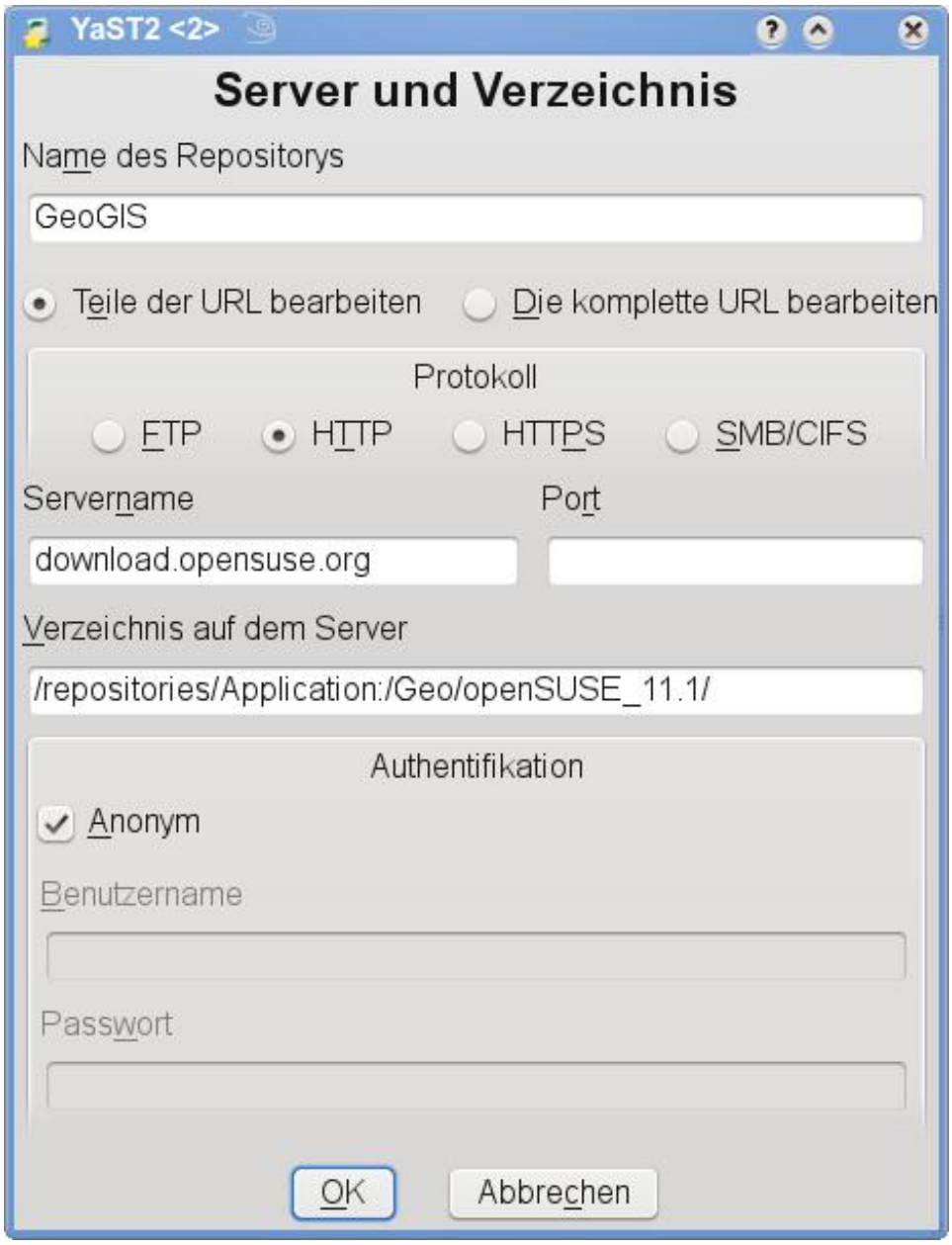
The primary site for Python GDAL is http://trac.osgeo.org/gdal/wiki/GdalOgrInPython.

Windows users should search the Python Package Index (http://pypi.python.org/pypi) for **GDAL** to get Windows *.exe installers. The current version is GDAL 1.6.1 and installers are available for Python 2.4, 2.5 and 2.6.

Linux users may build the GDAL Python module by theirselves.

For some distributions you'll find precompiled packages as *.rpm or *.deb.

openSUSE users should add http://download.opensuse.org/repositories/Application:/Geo/openSUSE_11.1/i586/ to their YaST Software-Repositories.



Similar packages are also available for Ubuntu and Debian. You should search for "python-gdal".

Mac OS X users may consult William Kyngesburyes KyngChaos Wiki (http://www.kyngchaos.com/) or http://darwinports.com/.

OGR for vector data

The easiest way to describe what OGR is, is to quote the website OGR Simple Feature Library (http://www.gdal.org/ogr/index.html) :

```
"The OGR Simple Features Library is a C++ open source library (and commandline tools) providing rea
access to a variety of vector file formats including ESRI Shapefiles, S-57, SDTS, PostGIS, Oracle S

OGR is a part of the GDAL library."
```

A simple Shapefile writer

So let's write a simple ESRI Shapefile for a quick start:

```
# script 1
from osgeo import ogr as ogr #[1]
import datetime

# Some simple data as input
data = [[0,3333317.0,5684892.0,'Hans','true'],[1,3333417.0,5684892.0,'Fritz','true'],\
        [2,3333317.0,5684992.0,'Willi','false'],[3,3333417.0,5684992.0,'Walter','false']]

# Create Shapefile named "points.shp" for output
driver     = ogr.GetDriverByName('ESRI Shapefile') #[2]
output     = "points.shp" #[3]

datasource = driver.CreateDataSource(output) #[4]
layer      = datasource.CreateLayer(output, geom_type=ogr.wkbPoint) #[5]

# Define attribute table
field_ID    = ogr.FieldDefn()    #[6]
field_ID.SetName('ID')           #[7]
field_ID.SetType(ogr.OFTInteger) #[8]
field_ID.SetWidth(1)             #[9]
layer.CreateField(field_ID)      #[10]

field_TEXT  = ogr.FieldDefn('TEXT', ogr.OFTString) #[11]
field_TEXT.SetWidth(20)
layer.CreateField(field_TEXT)

field_FLOAT = ogr.FieldDefn('FLOAT', ogr.OFTReal)
field_FLOAT.SetWidth(10)
field_FLOAT.SetPrecision(2)
layer.CreateField(field_FLOAT)

field_DATE  = ogr.FieldDefn('DATE', ogr.OFTDate)
field_DATE.SetWidth(8)
layer.CreateField(field_DATE)

field_BOOL  = ogr.FieldDefn('BOOLEAN', ogr.OFTString)
field_BOOL.SetWidth(5)
layer.CreateField(field_BOOL)

feature    = ogr.Feature(layer.GetLayerDefn()) #[12]
# Iterate over input data
for ID, X, Y, TEXT, BOOLEAN in data:
    DATE    = datetime.date.today()

    wkt     = "POINT (" + str(X) + " " + str(Y) + ")" #[13]
    point   = ogr.CreateGeometryFromWkt(wkt)

    # Set geometry
    feature.SetGeometryDirectly(point) #[14]

    # Set attributes
    feature.SetField('ID',ID) #[15]
    feature.SetField('TEXT',TEXT)
    feature.SetField('FLOAT',X)
    feature.SetField('DATE',DATE)
    feature.SetField('BOOLEAN',BOOLEAN)

    layer.CreateFeature(feature) #[16]

# Clean up
feature.Destroy()    #[17]
datasource.Destroy() #[18]

print "Feature <" + str(output) + "> successfully created."
del output
```

This simple script will be the initial point for further explanations, but let's describe in detail how it works.

[1] First of all import the ogr module. Depending on your installation, you'll find the ogr module normally under

```
- Python[XX]\Lib\site-packages\osgeo\ogr.py  --> from osgeo import ogr as ogr
or
- Python[XX]\Lib\site-packages\ogr.py        --> import ogr
```

where [XX] stands for your Python version, e.g. *Python25*.

[2] Next a specific driver is necessary. In the following examples I'll use the 'ESRI Shapefile' driver. As Python GDAL is a wrapper for the gdal.dll, you can only use those drivers, which are supported by the library you installed. Go to your GDAL installation directory, e.g. c:\gdalwin32-1.6\bin and type in **ogr2ogr**. The supported drivers will be listed, e.g.:

```
"ESRI Shapefile"
"MapInfo File"
"TIGER"
"S57"
"DGN"
"Memory"
"BNA"
"CSV"
"GML"
"GPX"
"KML"
"GeoJSON"
"GMT"
"ODBC"
"Geoconcept"
```

It is essential that the driver name is used exactly, regarding upper and lower cases!

[3] After specifying the driver, define a name for your output dataset, e.g. "points.shp". Please note, that some formats need a file extension like *.kml and it's dispensable for other formats. Just try what happens, if you vary your script the following way:

```
output = "points.shp"
output = "points"
```

[4] Now, when a driver is loaded and a name for the output dataset is defined, create a datasource.

[5] As the datasource is only an empty container, it's necessary to create some content in the form of a layer by geometry type ogr.wkbPoint. Please explore your ogr module which geometry types are also supported by typing

```
>>>dir(ogr)
```

You'll get a list of geometry types similar to this:

```
'wkb25Bit', 'wkbGeometryCollection', 'wkbGeometryCollection25D', 'wkbLineString', 'wkbLineString25D',
'wkbMultiLineString25D', 'wkbMultiPoint', 'wkbMultiPoint25D', 'wkbMultiPolygon', 'wkbMultiPolygon25D',
'wkbPoint25D', 'wkbPolygon', 'wkbPolygon25D', 'wkbUnknown', 'wkbXDR'
```

[6] After determining the geometry type, define some attributes as instances of ogr.FieldDefn().

[7] An attribute allways has a name, set by .SetName('AttributeName'). Note that some formats have restrictions regarding naming conventions. In case of a ESRI Shapefile, which stores the attributes in a dBase file (*.dbf), the attribute name must not have more than 11 characters. Special characters and space characters are also not allowed.

[8] Next configure the data type by .SetType(). To get a list of supported data types, just type in:

```
>>>dir(ogr)
```

and search the output for "OFT*", which means OGR Field Type. You'll get a list like this:

```
'OFTBinary', 'OFTDate', 'OFTDateTime', 'OFTInteger', 'OFTIntegerList', 'OFTReal', 'OFTRealList',
'OFTString', 'OFTStringList', 'OFTTime', 'OFTWideString', 'OFTWideStringList'
```

[9] Having determined the data type, now set the field length using .SetWidth(). If decimal places are requiered, e.g. for floating-point numbers ('OFTReal'), additionally set the positions after decimal point using .SetPrecision().

[10] Finally add the new field definition to the layer.

[11] To shorten field definition (steps 6 to 8), just define attribute name and data type when instantiating a new field.

[12] Having done all predefinitions, it's time to create features. For this purpose instantiate a new feature.

[13] There are 3 ways for creating geometries:

```
1. CreateGeometryFromWkt  --> define geometries using Well-Known Text (WKT) representations
2. CreateGeometryFromGML  --> define geometries as OpenGIS Geography Markup Language (GML)
3. CreateGeometryFromWkb  --> define geometries using Well-Known Binary (WKB) constructs
```

Creating geometries from GML and WKB will be discussed later on.

Using Well-Known Text representations for creating a point geometry is realy simple:

```
1. define a string objekt like "POINT (3333317 5684892)" and
2. create the geometry importing the WKT sting
```

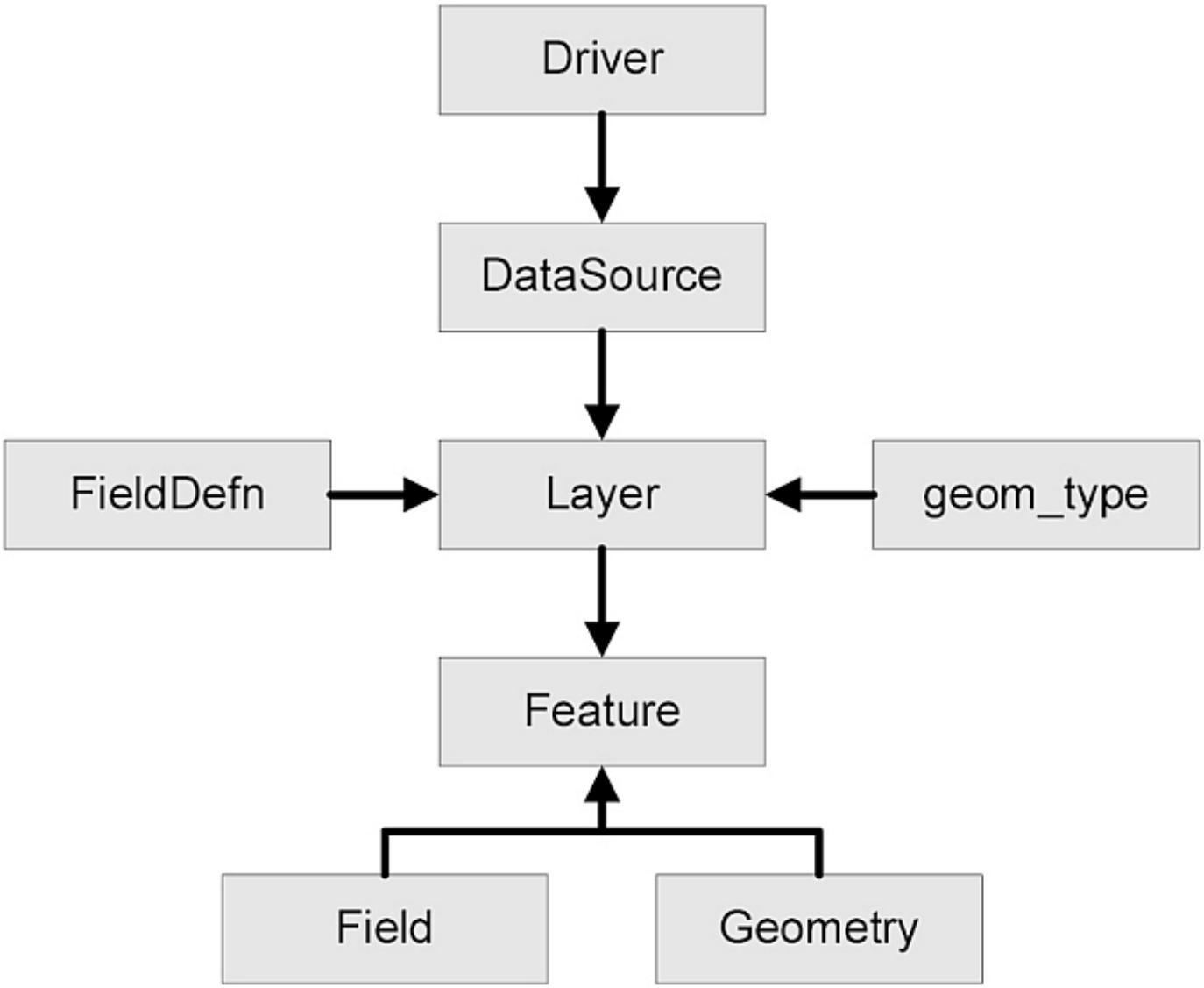
[14] Finally assign the new geometry to the (still empty) feature.

[15] Next set feature attributes by passing attribute name and attribute value to .SetField().

[16] Now the new feature has a geometry and attribute values, so that it seems to be complete and can be added to the layer.

[17|18] As a matter of form, clean up some Python objects. Don't worry, the new features will not be deleted! Only the Python references are purged.

At this point you should have an idea, how to create a simple point Shapefile. To memorize the last steps, the following map should be helpful:



Overwriting existing data

Other formats

Lines

Polygons

Retrieved from "http://www.pygis.de/index.php/GDAL"