spatialist Documentation

Release 0.2.8

John Truckenbrodt, Felix Cremer, Ismail Baris

Jun 24, 2019

Contents

1 Installation **1**

1.1 Installation of dependencies 1

1.2 Installation of spatialist 2

2 API Documentation **5**

2.1 Raster Class 5

2.2 Raster Tools 5

2.3 Vector Class 5

2.4 Vector Tools 5

2.5 General Spatial Tools 5

2.6 Database Tools 5

2.7 Ancillary Functions 5

2.8 ENVI HDR file manipulation 5

2.9 Data Exploration 5

3 Some general examples **7**

3.1 in-memory vector object rasterization 7

4 Indices and tables **9**

CHAPTER 1

Installation

1.1 Installation of dependencies

If you are using Windows, the easiest way to work with spatialist and Python in general is by using Anaconda. It comes with all basic requirements of spatialist. The more specific instructions below are intended for Linux users.

1.1.1 GDAL

spatialist requires GDAL version >=2.1 built with GEOS and PROJ4 as dependency as well as the GDAL Python binding. Alternatively, one can use pygdal, a virtualenv and setuptools friendly version of standard GDAL python bindings.

Ubuntu

Starting with release Yakkety (16.10), Ubuntu comes with GDAL >2.1. See here. You can install it like this:

```
sudo apt-get install python-gdal python3-gdal gdal-bin
```

For older Ubuntu releases you can add the ubuntugis repository to apt prior to installation to install version >2.1:

```
sudo add-apt-repository ppa:ubuntugis/ppa
sudo apt-get update
```

This way the required dependencies (GEOS and PROJ4 in particular) are also installed. You can check the version by typing:

```
gdalinfo --version
```

Debian

Starting with Debian 9 (Stretch) GDAL is available in version >2.1 in the official repository.

Building from source

Alternatively, you can build GDAL and the dependencies from source. The script *spatialist/install/install_deps.sh* gives specific instructions on how to do it. It is not yet intended to run this script via shell, but rather to follow the instructions step by step.

1.1.2 SQLite + SpatiaLite

Windows

While sqlite3 and its Python binding are usually already installed, the spatialite extension needs to be added. Two packages exist, libspatialite and mod_spatialite. Both can be used by spatialist. It is strongly recommended to use Ubuntu >= 16.04 (Xenial) or Debian >=9 (Stretch), which offer the package *libsqlite3-mod-spatialite*. This package is specifically intended to only serve as an extension to *sqlite3* and can be installed like this:

```
sudo apt-get install libsqlite3-mod-spatialite
```

After installation, the following can be run in Python to test the needed functionality:

```python
import sqlite3
# setup an in-memory database
con = sqlite3.connect(':memory:')
# enable loading extensions and load spatialite
con.enable_load_extension(True)
try:
    con.load_extension('mod_spatialite.so')
except sqlite3.OperationalError:
    con.load_extension('libspatialite.so')
```

In case loading extensions is not permitted you might need to install the package *pysqlite2*. See the script *spatialist/install/install_deps.sh* for instructions. There you can also find instructions on how to install spatialite from source. To test *pysqlite2* you can import it as follows and then run the test above:

```python
from pysqlite2 import dbapi2 as sqlite3
```

Installing this package is likely to cause problems with the sqlite3 library installed on the system. Thus, it is safer to build a static sqlite3 library for it (see installation script).

1.2 Installation of spatialist

For the installation we need the Python tool pip and the version control system git. On Windows, pip is installed together with Anaconda. Git can be installed like this:

```
conda install git
```

On Linux:

```
sudo apt-get install python-pip git
```

Once everything is set up, spatialist is ready to be installed. You can install stable releases like this:

```
python -m pip install spatialist
```

or the latest developer version like this:

```
sudo python -m pip install git+https://github.com/johntruckenbrodt/spatialist.git
```

On Windows you need to use the Anaconda Prompt and leave out `sudo` in the above command.

CHAPTER 2

API Documentation

2.1 Raster Class

2.2 Raster Tools

2.3 Vector Class

2.4 Vector Tools

2.5 General Spatial Tools

2.6 Database Tools

2.7 Ancillary Functions

2.8 ENVI HDR file manipulation

2.9 Data Exploration

CHAPTER 3

Some general examples

3.1 in-memory vector object rasterization

Here we create a new raster data set with the same geo-information and extent as a reference data set and burn the geometries from a shapefile into it.

In this example, the shapefile contains an attribute `Site_name` and one of the geometries in the shapefile has a value of `my_testsite` for this attribute.

We use the `expressions` parameter to subset the shapefile and burn a value of 1 in the raster at all locations where the geometry selection overlaps. Multiple expressions can be defined together with multiple burn values.

Also, burn values can be appended to an already existing raster data set. In this case, the rasterization is performed in-memory to further use it for e.g. plotting. Alternatively, an `outname` can be defined to directly write the result to disk as a GeoTiff.

See `spatialist.raster.rasterize()` for further reference.

```
>>> from spatialist import Vector, Raster
>>> from spatialist.raster import rasterize
>>> import matplotlib.pyplot as plt
>>>
>>> shapefile = 'testsites.shp'
>>> rasterfile = 'extent.tif'
>>>
>>> with Raster(rasterfile) as ras:
>>>     with Vector(shapefile) as vec:
>>>         mask = rasterize(vec, reference=ras, burn_values=1, expressions=["Site_
↪Name='my testsite'"])
>>>         plt.imshow(mask.matrix())
>>>         plt.show()
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search