mothergeo Documentation

Release 0.0.1

Pat Daburu

Feb 27, 2020

CONTENTS

Word to your mother.

CHAPTER

ONE

DEVELOPMENT

mothergeo is under active development.

1.1 Python Module Dependencies

The `requirements.txt` file contains this project's module dependencies. You can install these dependencies using `pip`.

```
pip install -r requirements.txt
```

1.1.1 requirements.txt

```
alabaster==0.7.12
Babel==2.8.0
certifi==2019.11.28
chardet==3.0.4
docutils==0.16
idna==2.8
imagesize==1.2.0
Jinja2==2.10.3
MarkupSafe==1.1.1
packaging==20.1
Pygments==2.5.2
pyparsing==2.4.6
pytz==2019.3
requests==2.22.0
six==1.14.0
snowballstemmer==2.0.0
Sphinx==2.3.1
sphinx-rtd-theme==0.4.3
sphinxcontrib-applehelp==1.0.1
sphinxcontrib-devhelp==1.0.1
sphinxcontrib-htmlhelp==1.0.2
sphinxcontrib-jsmath==1.0.1
sphinxcontrib-qthelp==1.0.2
sphinxcontrib-serializinghtml==1.1.3
urllib3==1.25.8
```

1.2 Design Considerations

Coming soon.

1.3 Source Control

Coming soon.

1.4 How To...

1.4.1 How to Use an Alternate PyPI (Package Index)

The Python Package Index (PyPI) works great, but there may be times when, for whatever reason, you need to host packages elsewhere. You can set up your own index, or you can use a hosted service like Gemfury.

This article describes, in general terms, some of the things you'll need to do in your development environment to make your use of an alternate package index a little smoother.

Installing Modules with `pip`

One of the tricks to installing packages from your alternate repository is telling `pip` about it.

`pip.ini`

While you can use command line parameters with `pip` to indicate the location of your package index server, you can also modify (or create) a special `pip` configuration file called `pip.ini` that will allow you install packages from the command line just as you would if you were installing them from the public repositories.

Windows

On Windows, you can place a `pip.ini` file at `%APPDATA%\pip\pip.ini`. Use the `extra-index-url` option to tell `pip` where your alternate package index lives. If your package index doesn't support SSL, you can supress warnings by identifying it as a `trusted-host`. The example below assumes the name of your server is **pypi.mydomain.com** and you're running on non-standard port **8080**.

```
[global]
extra-index-url = http://pypi.mydomain.com:8080

[install]
trusted-host = pypi.mydomain.com
```

Linux

Coming soon.

Note: If you are using SSL with a verified certificate, you won't need the `trusted-host` directive.

Publishing Modules

This article doesn't go into much detail on the general process of publishing modules, but we'll assume that you're using setuptools to publish.

.pypirc

You can automate the process of publishing your package with distutils by modifying the `.pypirc` file in your home directory. This file typically contains the common public indexes, but you can also add your alternate index. The example below assumes you're using Gemfury, but the format will be fundamentally similar regardless of where you're hosting your repository.

```
[distutils]
index-servers =
  pypi
  fury

[pypi]
username=mypypiuser
password=$ecret-Pa$$w0rd

[fury]
repository: https://pypi.fury.io/myfuryusername/
username: $ecret-K3y!
password:
```

With that in place, you can build and upload your package by identifying the configured index server name.

```
python setup.py sdist upload -r fury
```

Note: Rembemer that the keys and passwords in your *.pypirc* are secrets, and should be kept away from prying eyes.

1.4.2 How To Document Your Code

Coming Soon.

1.4.3 How To Set Up Your Development Environment

This article describes the steps you can follow to get the mothergeo-py project set up for development.

Get the Source

This project is managed under source control in GitHub, so you'll need to install `git`. Once you have that going, getting the latest version of the code is just a matter of cloning the repository into your development directory.

```
git clone https://github.com/patdaburu/mothergeo-py.git
```

Get the Requirements

The project uses a number of modules that are available from the PyPI package repository. All of the required modules should be listed in the *requirements.txt* file in the root directory, and you can get them using `pip`.

```
pip install -r requirements.txt
```

1.4.4 How To Install GDAL/OGR Packages on Ubuntu

GDAL is a translator library for raster and vector geospatial data formats.

OGR Simple Features Library is a C++ open source library (and commandline tools) providing read (and sometimes write) access to a variety of vector file formats including ESRI Shapefiles, S-57, SDTS, PostGIS, Oracle Spatial, and Mapinfo mid/mif and TAB formats.

OGR is a part of the GDAL library.

GDAL/OGR are used in numerous GIS software projects and, lucky for us, there are bindings for python. In fact, you may want to check out the Python GDAL/OGR Cookbook.

This article describes a process you can follow to install GDAL/OGR on Ubuntu.

Before You Begin: Python 3.6

If you are installing the GDAL/OGR packages into a virtual environment based on Python 3.6, you may need to install the python3.6-dev package.

```
sudo apt-get install python3.6-dev
```

For more information about creating virtual environments on Ubuntu 16.04 LTS, see *A Note About Python 3.6 and Ubuntu 16.04 LTS*.

Install GDAL/OGR

Much of this section is taken from a really helpful blog post by Sara Safavi. Follow these steps to get GDAL/OGR installed.

To get the latest GDAL/OGR version, add the PPA to your sources, then install the gdal-bin package (this should automatically grab any necessary dependencies, including at least the relevant libgdal version).

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

Once you add the repository, go ahead and update your source packages.

```
sudo apt-get update
```

Now you should be able to install the GDAL/OGR package.

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

To verify the installation, you can run `ogrinfo --version`.

```
ogrinfo --version
```

You will need the GDAL version to install the correct python bindings.

Install GDAL for Python

Before installing the GDAL Python libraries, you'll need to install the GDAL development libraries.

```
sudo apt-get install libgdal-dev
```

You'll also need to `export` a couple of environment variables for the compiler.

```
export CPLUS_INCLUDE_PATH=/usr/include/gdal
export C_INCLUDE_PATH=/usr/include/gdal
```

Now you can use `pip` to install the Python GDAL bindings.

```
pip install GDAL==<GDAL VERSION FROM OGRINFO>
```

Putting It All Together

If you want to run the whole process at once, we've collected all the commands above in the script below.

```
#!/usr/bin/env bash

sudo add-apt-repository ppa:ubuntugis/ppa && sudo apt-get update
sudo apt-get update
sudo apt-get install gdal-bin
sudo apt-get install libgdal-dev
export CPLUS_INCLUDE_PATH=/usr/include/gdal
export C_INCLUDE_PATH=/usr/include/gdal
pip install GDAL
```

Try It Out

Now that GDAL/OGR is installed, and you can program against it in Python, why not try it out? The code block below is a sample from the Python OGR/GDAL Cookbook that gets all the layers in an Esri file geodatabase.

```
# standard imports
import sys

# import OGR
from osgeo import ogr

# use OGR specific exceptions
ogr.UseExceptions()

# get the driver
driver = ogr.GetDriverByName("OpenFileGDB")

# opening the FileGDB
try:
    gdb = driver.Open(gdb_path, 0)
except Exception, e:
    print e
    sys.exit()

# list to store layers'names
featsClassList = []

# parsing layers by index
for featsClass_idx in range(gdb.GetLayerCount()):
    featsClass = gdb.GetLayerByIndex(featsClass_idx)
    featsClassList.append(featsClass.GetName())

# sorting
featsClassList.sort()

# printing
for featsClass in featsClassList:
    print featsClass

# clean close
del gdb
```

Acknowledgements

Thanks to Sara Safavi and Paul Whipp for contributing some of the leg work on this.

1.4.5 How To Use Mother's `logging` Module

Mother has her own `logging` module which you are *strongly* encouraged to use within the project.

Using the `loggable_class` Decorator

When you create a new class, you can provide it with a Python logger just by decorating it with the `@loggable_class` decorator (abbreviated to `@loggable` in the example). The code sample below will provide a `logger` property to the decorated class.

```
import logging
import sys
# (Let's alias the decorator's name for brevity's sake.)
from mothergeo.logging import loggable_class as loggable

# We'll just create a simple test configuration so we can see logging occur.
logging.basicConfig(
    stream=sys.stdout,
    level=logging.DEBUG
)

@loggable
class MyUsefulClass(object):

    def my_useful_method(self):
        self.logger.debug("I'm using the logger that was provided by the decorator.")
        print("You should see some logging output.")

if __name__ == "__main__":
    my_useful_object = MyUsefulClass()
    my_useful_method()
```

Overriding the Default Logger Name

By default, the name of the Python logger a class decorated with the `@logger_class` decorator creates a logger based on the module in which the class is found, and the name of the class. You can override this behavior by providing a `logger_name` property on the decorated class, as in the example below.

```
import logging
import sys
# (Let's alias the decorator's name for brevity's sake.)
from mothergeo.logging import loggable_class as loggable

# We'll just create a simple test configuration so we can see logging occur.
logging.basicConfig(
    stream=sys.stdout,
    level=logging.DEBUG
)

@loggable
class MyUsefulClass(object):

    logger_name = 'alterate.logger.name'  # Override the default logger name formula.

    def my_useful_method(self):
```

(continues on next page)

(continued from previous page)

```
        self.logger.debug("The logger's name should reflect the 'logger_name'␣
↪property.")
        print("You should see some logging output.")

if __name__ == "__main__":
    my_useful_object = MyUsefulClass()
    my_useful_method()
```

See also:

If you're interested in reading more about logging in Python, have a look at *From the Hitchhicker's Guide: Logging in Python Libraries*.

1.4.6 How To Run Your Own PyPI Server

This article includes some notes we hope will be helpful in setting up your own PyPI server for those times when you need to share modules, but you're not ready to publish them to the rest of the World.

The Server Side

There are a few different ways to host your repository. This article focuses on pypi-server which you can get from the public package index.

The Client Side

Installing Modules with `pip`

One of the tricks to installing packages from your private repository is telling `pip` about it.

On Windows, you can place a `pip.ini` file at `%APPDATA%\pip\pip.ini`

```
[global]
extra-index-url = http://<host>:<port>/

[install]
trusted-host = <host>
```

Note: If you are using SSL with a verified certificate, you won't need the `trusted-host` directive.

Publishing Updates

To keep life a little simpler, you probably want to modify your `.pypirc` file to include information about your new repository server. You can do this by adding an alias for the server in your list of *index-servers*. When you're finished, your `.pypirc` file might look something like the one below assuming your give your new repository **myownpypi** as an alias.

```
[distutils]
index-servers =
  pypi
```

(continues on next page)

(continued from previous page)

```
  pypitest
  myownpypi

[pypi]
username=<pypi_user>
password=<pypi_password>

[pypitest]
username=<pypitest_user>
password=<pypitest_password>

[myownpypi]
repository: http://<host>:<port>
username: <myownpypi_user>
password: <myownpypi_password>
```

Note: There are refinements to this process and we'll update this document as we go along.

1.4.7 How to Exclude Folders in PyCharm

If you're using PyCharm to develop, you may have noticed that it has some pretty righteous searching and refactoring capabilities; *however,* there are likely to be some folders in your project's directory tree that contain files you don't want PyCharm to look at when it comes time to search or perform automatic refactoring. Examples of these directories include:

- the Python virtual environment (because you *definitely* don't want to modify the stuff in there);
- the `docs` directory (because you don't really *need* to refactor the stuff in there); and
- the `lib` directory (because nothing in there should depend on the code you're writing, *right?*).

There may be others as well.

PyCharm allows you to *exclude* directories from consideration when searching and refactoring. You can exclude a directory by right-clicking on it and selecting **Mark Directory as → Excluded**.

See also:

JetBrains' website has an article called Configuring Folders Within a Content Root which has additional insights on how and why you might want to configure the folders in the project.

1.4.8 How To Set Up a Virtual Python Environment (Linux)

`virtualenv` is a tool to create isolated Python environments. You can read more about it in the Virtualenv documentation. This article provides a quick summary to help you set up and use a virtual environment.

A Note About Python 3.6 and Ubuntu 16.04 LTS

If you're running Ubuntu 16.04 LTS (or and earlier version), Python 3.5 is likely installed by default. *Don't remove it!* To get Python 3.6, follow the instructions in this section.

Add the PPA

Run the following command to add the Python 3.6 PPA.

```
sudo add-apt-repository ppa:jonathonf/python-3.6
```

Check for Updates and Install

Check for updates and install Python 3.6 via the following commands.

```
sudo apt-get update
sudo apt-get install python3.6
```

Now you have three Python version, use `python` to run version 2.7, `python3` for version 3.5, and `python3.6` for version 3.6.

For more information on this subject, check out Ji m's article How to Install Python 3.6.1 in Ubuntu 16.04 LTS.

Create a Virtual Python Environment

`cd` to your project directory and run `virtualenv` to create the new virtual environment.

The following commands will create a new virtual environment under my-project/my-venv.

```
cd my-project
virtualenv --python python3.6 venv
```

Activate the Environment

Now that we have a virtual environment, we need to activate it.

```
source venv/bin/activate
```

After you activate the environment, your command prompt will be modified to reflect the change.

Add Libraries and Create a *requirements.txt* File

After you activate the virtual environment, you can add packages to it using `pip`. You can also create a description of your dependencies using `pip`.

The following command creates a file called `requirements.txt` that enumerates the installed packages.

```
pip freeze > requirements.txt
```

This file can then be used by collaborators to update virtual environments using the following command.

```
pip install -r requirements.txt
```

Deactivate the Environment

To return to normal system settings, use the `deactivate` command.

```
deactivate
```

After you issue this command, you'll notice that the command prompt returns to normal.

Acknowledgments

Much of this article is taken from The Hitchhiker's Guide to Python. Go buy a copy right now.

1.4.9 How To Set Up a Virtual Python Environment (Windows)

`virtualenv` is a tool to create isolated Python environments. You can read more about it in the Virtualenv documentation. This article provides a quick summary to help you set up and use a virtual environment.

Where's My Python?

Sometimes the trickiest part of setting up a virtual environment on Windows is finding your python distribution. If the installer didn't add it to your `PATH` variable, you may have to go looking. If you downloaded and installed python from python.org and accepted all the defaults during installation, `python.exe` may be found in one of the following locations:

64-bit (Preferred)

```
C:\Users\%username%\AppData\Local\Programs\Python\Python36\python.exe
```

32-bit

```
C:\Users\%username%\AppData\Local\Programs\Python\Python36-32\python.exe
```

Install `virtualenv`

If you try to run `virtualenv` and find it isn't present, you can install it using pip.

```
pip install virtualenv
```

`virtualenv.exe` will likely now be found in your python installation directory under the `Scripts` subdirectory.

Create a Virtual Python Environment

`cd` to your project directory and run `virtualenv` to create the new virtual environment.

The following commands will create a new virtual environment under `my-project/my-venv`.

```
cd my-project
virtualenv --python C:\Path\To\Python\python.exe venv
```

Note: If Windows cannot find `virtualenv.exe`, see *Install virtualenv*. You can either add the executable's home directory to your `PATH` variable, or just include the full path in your command line. If you aren't sure where `python.exe` is installed, see *Where's My Python?*.

Activate the Environment

Now that we have a virtual environment, we need to activate it.

```
.\venv\Scripts\activate
```

After you activate the environment, your command prompt will be modified to reflect the change.

Add Libraries and Create a *requirements.txt* File

After you activate the virtual environment, you can add packages to it using `pip`. You can also create a description of your dependencies using `pip`.

The following command creates a file called `requirements.txt` that enumerates the installed packages.

```
pip freeze > requirements.txt
```

This file can then be used by collaborators to update virtual environments using the following command.

```
pip install -r requirements.txt
```

Deactivate the Environment

To return to normal system settings, use the `deactivate` command.

```
deactivate
```

After you issue this command, you'll notice that the command prompt returns to normal.

Acknowledgments

Much of this article is taken from The Hitchhiker's Guide to Python. Go buy a copy right now.

1.5 Miscellany

1.5.1 From the Hitchhicker's Guide: Logging in Python Libraries

Note: If you're wondering how Mother likes to handle logging, take a look at *How To Use Mother's logging Module*.

It is, of course, desirable for library modules to perform logging. However, we generally want to maintain consistency and allow the consuming application to perform logging configuration. This article details a strategy for achieving those goals.

A Bit of Loging Boilerplate

From The Hitchhiker's Guide to Python, Chapter 4:

> Do not add any handlers other than `NullHandler` to your library's loggers. Place the following code in your project's top-level `__init__.py`.

```
# Set default logging handler to avoid "No handler found" warnings.
import logging
try: # Python 2.7+
    from logging import NullHandler
except ImportError:
    class NullHandler(logging.Handler):
        def emit(self, record):
            pass

logging.getLogger(__name__).addHandler(NullHandler())
```

Acknowledgments

Much of this article is taken from The Hitchhiker's Guide to Python. Go buy a copy right now.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search