

Spectrum and Spectrum Spatial Python Packages

This notebook describes the `spectrumpy` and `spectrumspatialpy` python libraries through examples.

Copyright 2019 Pitney Bowes Inc.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

About the Spectrum and Spectrum Spatial packages

`spectrumpy` is a Python package that connects to a Spectrum server. The servers and credentials available can be defined in a configuration file located on the Jupyter notebook server. This is to avoid the need to include Spectrum URLs and credentials in notebooks in plain text.

`spectrumpy` will dynamically detect web services exposed from data flows created in Spectrum and make them available as Python functions. This package, as well as `spectrumspatialpy` can be used in any Python environment including Jupyter notebooks such as these.

The `spectrumpy` package can be used without the `spectrumspatialpy` package.

The **`spectrumspatialpy`** package provides Python integration for the Spectrum Spatial services such as the Feature Service for querying spatial data. This package requires `spectrumpy` as a prerequisite, along with other requirements listed below.

Setup and Prerequisites

Installing Jupyter

Download and install Jupyter using the Anaconda Python 3.6 or later distribution at <http://jupyter.org/install>. Note do not use Python v2.x. The Anaconda distribution makes it very easy to setup and get up and running quickly.

This notebook, the resources it embeds such as images, etc can be stored anywhere as long as the internal folder structure and file names are retained. Once Jupyter is installed and started, from the Jupyter home page navigate to the notebook and launch it.

Installing the spectrumpy package

```
In [1]: ▶ pip install --quiet git+https://github.com/spectrumpy/spectrumpy
```

Note: you may need to restart the kernel to use updated packages.

Prerequisite to installing the spectrumspatialpy package:

The **spectrumspatialpy** package provides Python integration for the Spectrum Spatial services such as the Feature Service for querying spatial data. The package depends on several other packages, many of which will be installed when installing the spectrumspatialpy package. However, there are a few exceptions. From the Anaconda Command Prompt (which can be found in the Windows menu under Anaconda3 (64-bit)) run these commands:

- conda install shapely
- conda install geopandas

```
In [2]: ▶ conda install -y shapely geopandas
```

```
Collecting package metadata (current_repodata.json): ...working... done  
Solving environment: ...working... done
```

```
# All requested packages already installed.
```

Note: you may need to restart the kernel to use updated packages.

Installing the spectrumspatialpy package

```
In [3]: ▶ pip install --quiet git+https://github.com/spectrumpy/spectrumspatialpy
```

Note: you may need to restart the kernel to use updated packages.

Installing and Enabling Leaflet

`spectrumsatialpy` does not use `ipyleaflet`, however, the sample notebook for it does. The `ipyleaflet` package enables leaflet maps to be generated and displayed directly within the notebook itself.

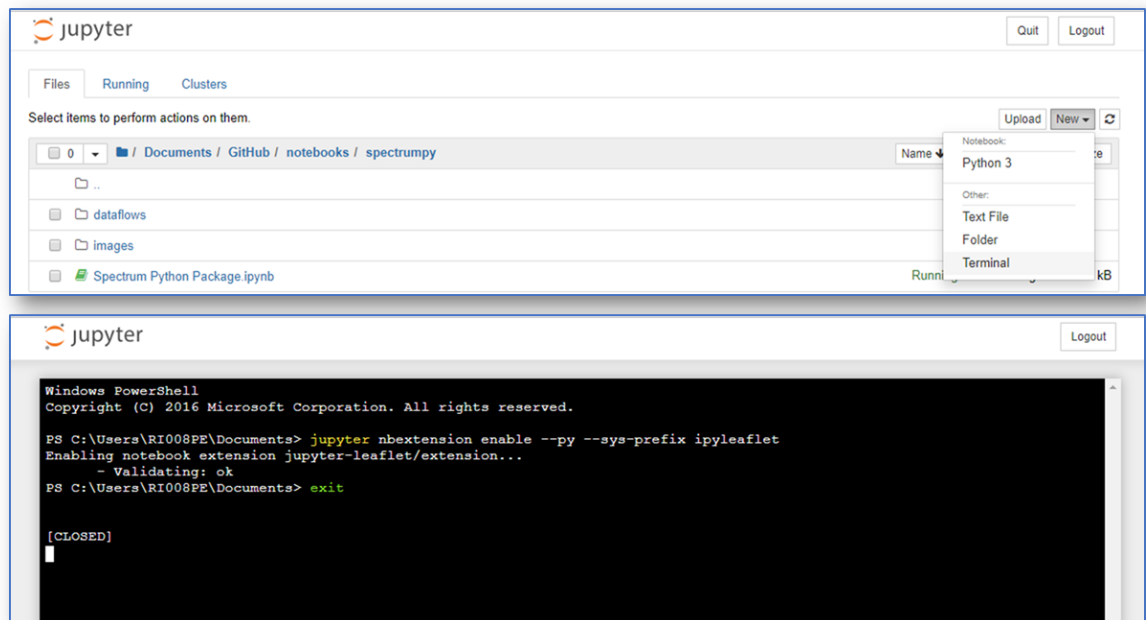
```
In [4]: ► pip install --quiet ipyleaflet
```

Note: you may need to restart the kernel to use updated packages.

There are sometimes issues with the map displaying. To enable `ipyleaflet` to display properly, the following command must be run:

```
jupyter nbextension enable --py --sys-prefix ipyleaflet
```

To run this command, either launch an Anaconda prompt from the Windows menu or from the Jupyter main home page, launch a terminal session and enter the command there as shown in the following images.



Note: You should restart the kernel before opening and running the Spectrum Spatial notebook.

Using spectrumpy

Once the library is installed, you should be able to import it into this notebook by executing the import command as shown in the following cell. Note that using `spectrumsatialpy` is covered in a separate notebook with this notebook serving as a prerequisite.

```
In [5]: ► import spectrumpy
```

Spectrum Servers

The package was designed to not require username and passwords to be embedded within the notebook. The package looks for an INI file which will identify all "registered" or known Spectrum hosts and credentials. The default INI in the package looks like this:

```
[SERVERS]
1=localhost

[localhost]
url=http://127.0.0.1:8080/
user=admin
pwd=admin
```

This file identifies one known server named "localhost". The localhost section then stores the URL, username, and password. This file is read when the package is imported into the notebook. The localhost server is local to the Jupyter notebook (python engine). Additional initialization files can be specified in the user's home directory in a file named `.spectrum_servers.ini` or in this notebook's folder in a file named `.spectrum_servers.ini`.

The root class in the spectrum package is called `Servers` and provides a method called `getAvailableServers` to print out the names of the known servers. The next cell will list them.

```
In [6]: ▶ print (spectrumpy.Servers.getAvailableServers())

['localhost', 'CaryLaptop']
```

On my machine, the above cell lists two servers: 'localhost' and 'CaryLaptop'. Since the server.ini file is located within the package source, we don't want to require users to have to modify it in this location. This notebook includes a file named ".spectrum_servers.ini" in the notebook's root directory. This file on my machine adds another Spectrum server called 'CaryLaptop' that refers to my local Spectrum machine like this:

```
[SERVERS]
2=CaryLaptop

[CaryLaptop]
url=http://127.0.0.1:8080/
user=admin
pwd=admin
```

Notice that the `SERVERS` section uses a numeric key starting with 2. This is because the INI file found with the package has a key starting with 1. If this file started at 1, this would replace the 1 from the root INI file and effectively eliminate the localhost default setting. The definition of `CaryLaptop` happens to be the same as localhost, but is included for illustrative purposes.

To connect to a named Spectrum server, use the method `"getServer"` off the `Servers` object. The cell below connects to my Troy dev instance and returns a `Server` object which is assigned to a variable named `myServer`.

```
In [7]: ▶ myServer=spectrumpy.Servers.getServer('CaryLaptop')
```

The `Spectrum Server` object will connect to Spectrum, dynamically detect all of the exposed rest services through the `"/rest/"` endpoint and add methods for each under an object called `SpectrumServices`. The `Apis` member of this object provides an iterator through each of the services. The following cell will list all of the known services exposed at `"myServer"`.

```
In [8]: ▶ for api in myServer.SpectrumServices().Apis:  
        print(api)
```

```
GlobalSentry  
ValidateAddressAUS  
USDatabaseLookup  
RelationshipExtractor  
spectrumspatialpy_route  
GetCityStateProvinceLoqate  
ValidateAddressGlobal  
GetPostalCodes  
AddressParser  
GeocodeAddressWorld  
Centrus  
ReverseGeoTAXInfoLookup  
ReverseAPNLookup  
GlobalAddressValidation  
GlobalGeocode  
AssignGeoTAXInfo  
TextCategorizer  
GetPostalCodesLoqate  
AutoCompleteLoqate  
EnvinsaGeocode  
CalculateDistance  
ReverseGeocodeUSLocation  
GeocodeUSAddress  
ValidateAddress  
ReversePBKeyLookup  
spectrumpy  
GetCityStateProvince  
GetCandidateAddressesLoqate  
EnvinsaHealthCheck  
GlobalTypeAhead  
OpenNameParser  
EntityExtractor  
GlobalSentryBatch  
PlatformConfiguration  
Spatial  
ValidateAddressLoqate  
GetCandidateAddresses  
GetTravelBoundary
```

There should be in the list above "GeocodeUSAddress". Since most Spectrums will have some US geocoding installed, we will use that as an example of how to dynamically call this service. The actual service typically exposes two resources - results.json and results.xml. The JSON endpoint is used. Data and Option query parameters can be passed to the function **except** the periods (.) should be replaced with underscores ("_"). Thus the following cell will call the GeocodeUSAddress rest service using the Data.AddressLine1 and Option.Dataset query parameters as function arguments Data_AddressLine1 and Option_Dataset respectively.

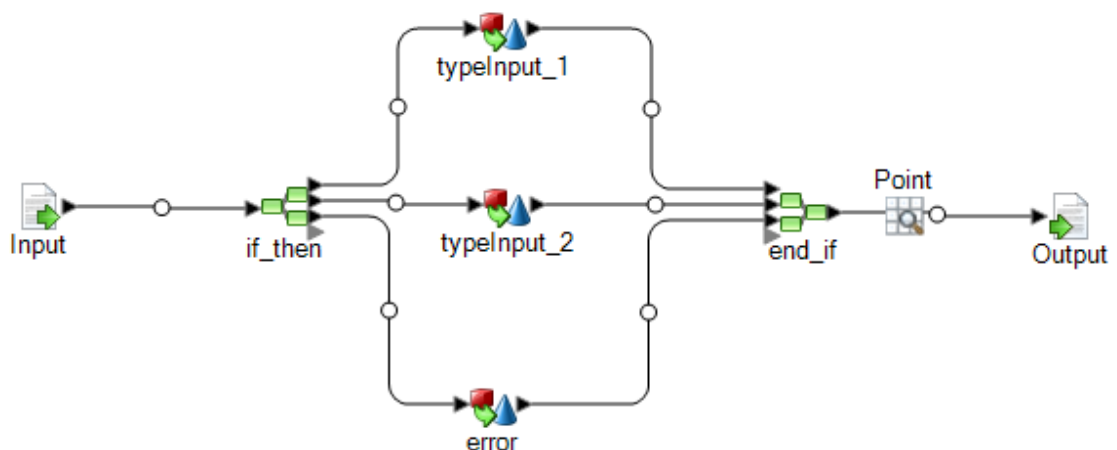
```
In [9]: s = myServer.SpectrumServices().GeocodeUSAddress(Data_AddressLine1="one global",
Option_Dataset="us",
Option_OutputRecordType="Auxiliary")

print (s)
```

```
{
  "output_port" : [ {
    "BlockSuffix" : "",
    "CBSADivisionCode" : "",
    "CBSAMetro" : "Y",
    "CBSACode" : "10580",
    "CensusBlockID" : "360830523011008",
    "USFIPSCountyNumber" : "083",
    "CSACode" : "104",
    "CensusTract" : "052301",
    "USFIPSSStateCode" : "36",
    "USFIPSSStateCountyCode" : "36083",
    "Latitude" : "42.682259",
    "Longitude" : "-73.704710",
    "StreetSide" : "L",
    "FirmName" : "",
    "AddressLine1" : "1 Global Vw",
    "AddressLine2" : "",
    "LastLine" : "Troy, NY 12180-8371",
    "City" : "Troy",
    "State" : "NY",
    "Zip" : "12180-8371"
  } ]
}
```

Calling a DataFlow service

Dataflows exposed as web services will be dynamically exposed on the spectrumpy server as functions that can be invoked as well. This notebook includes a sample service named `spectrumpy`. The service does nothing very interesting and makes no assumptions about installed modules. The dataflow is included with this notebook under the `dataflows` folder and can be imported into your Spectrum. The dataflow is defined as follows:



Given the following sample input

The 'Input Options' dialog box has two tabs: 'Input Fields' and 'Inspection Input'. The 'Inspection Input' tab is active, displaying the instruction 'Please enter input data for inspection.' Below this is a table with two columns: 'strInput' and 'typeInput'. The table contains four rows: 'foo' with '1', 'bar' with '2', 'fubar' with '3', and a wildcard '*' with '0'. To the right of the table are three buttons: 'Import Data...', 'Delete Record', and 'View Vertical'. At the bottom of the dialog are 'OK', 'Cancel', and 'Help' buttons.

strInput	typeInput
foo	1
bar	2
fubar	3
*	0

It produces the following output

The 'Inspection Results (spectrumpy succeeded)' window displays a table with the following data:

Status	strInput	Status.Code	strOutput_1	Status.Description	strOutput_2	typeInput
	foo		foo			1
	bar				bar	2
F	fubar	400		Invalid value for typeIn...		3

Here is how to call the web service from within the notebook:


```
In [10]: ► s = myServer.SpectrumServices().spectrumpy(Data_strInput="foo",Data_typeInput=1)
print(s)
s = myServer.SpectrumServices().spectrumpy(Data_strInput="bar",Data_typeInput=2)
print(s)
s = myServer.SpectrumServices().spectrumpy(Data_strInput="fubar",Data_typeInput=3)
print(s)
```

```
{
  "Output" : [ {
    "strOutput_1" : "foo",
    "strInput" : "foo",
    "typeInput" : 1,
    "user_fields" : [ ]
  } ]
}
{
  "Output" : [ {
    "strInput" : "bar",
    "typeInput" : 2,
    "strOutput_2" : "bar",
    "user_fields" : [ ]
  } ]
}
{
  "Output" : [ {
    "Status" : "F",
    "Status.Code" : "400",
    "Status.Description" : "Invalid value for typeInput",
    "strInput" : "fubar",
    "typeInput" : 3,
    "user_fields" : [ ]
  } ]
}
```

```
In [ ]: ►
```