Spectrum Python Package

This notebook describes the **`spectrumpy`** python library through examples.

Setup and Prerequisites ¶

Installing Jupyter

Download and install Jupyter using the Anaconda Python 3.6 distribution at http://jupyter.org/install (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 and the packages developed as part of this work are all expected to be stored in a root folder named **`jupyter`** off your home directory (e.g. `C:\Users\yourUserID\jupyter`). Within this folder should be a folder named **`notebooks`** where this notebook will reside. These are not requirements of Jupyter, rather they are how this notebook and code have been developed and this documentation assumes.

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

About the Spectrum package

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 credntials in notebooks in plain text.

Installing the spectrumpy package

In [1]:
```
pip install --quiet git+https://github.com/carypeebles/spectrumpy
```
```
Note: you may need to restart the kernel to use updated packages.
```

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

In [2]:
```
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 [3]:
```
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 [4]:
```
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 [5]:
```
for api in myServer.SpectrumServices().Apis:
    print(api)
```

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

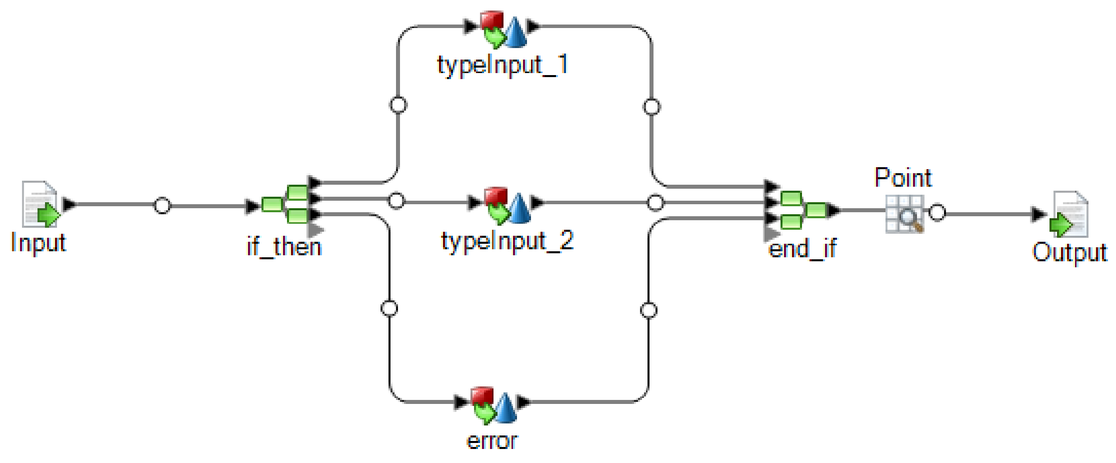
There should be in the list above "GeocodeUSAddress". Since most Spectrums will have some US geocoding installed, we will use that as an example of how to dynamically call this service. The actual service typically exposes two resources - results.json and results.xml. The JSON endpoint is used. Data 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 [6]:
```
s = myServer.SpectrumServices().GeocodeUSAddress(Data_AddressLine1="one globa
                                                 Option_Dataset="us",
                                                 Option_OutputRecordType="Au
print (s)
```

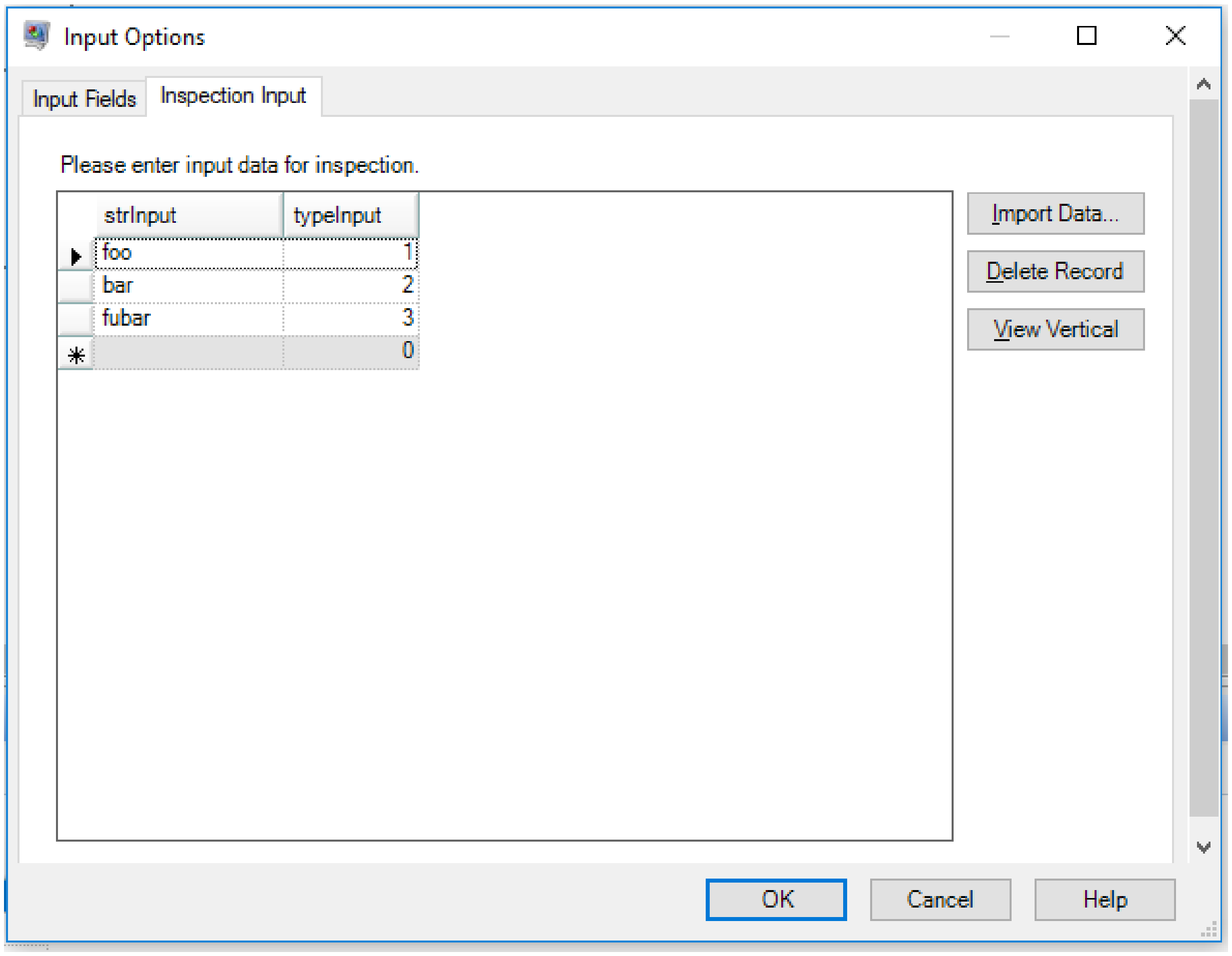
```
{
  "output_port" : [ {
    "BlockSuffix" : "",
    "CBSADivisionCode" : "",
    "CBSAMetro" : "Y",
    "CBSACode" : "10580",
    "CensusBlockID" : "360830523011008",
    "USFIPSCountyNumber" : "083",
    "CSACode" : "104",
    "CensusTract" : "052301",
    "USFIPSStateCode" : "36",
    "USFIPSStateCountyCode" : "36083",
    "Latitude" : "42.682259",
    "Longitude" : "-73.704710",
    "StreetSide" : "L",
    "FirmName" : "",
    "AddressLine1" : "1 Global Vw",
    "AddressLine2" : "",
    "LastLine" : "Troy, NY  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:



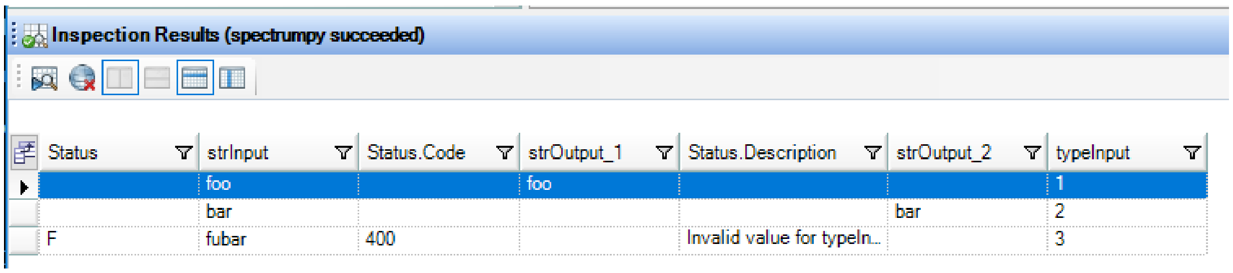
Given the following sample input



Please enter input data for inspection.

strInput	typeInput
foo	1
bar	2
fubar	3
	0

It produces the following output



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 [7]:

```
s = myServer.SpectrumServices().spectrumpy(Data_strInput="foo",Data_typeInput
print(s)
s = myServer.SpectrumServices().spectrumpy(Data_strInput="bar",Data_typeInput
print(s)
s = myServer.SpectrumServices().spectrumpy(Data_strInput="fubar",Data_typeInp
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" : [ ]
  } ]
}
```