# Spectrum and Spectrum Spatial Python Packages

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

# 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 credntials 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

Prerequisites are desicribed in the `Spectrum Python Setup` 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 spectrumspatialpy is covered in a separate notebook with this notebook serving as a prerequisite.

In [1]:

```
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 [2]:

```
print (spectrumpy.Servers.getAvailableServers())
```

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

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 [3]:

```
mySerer=spectrumpy.Servers.getServer('localhost')
```

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 [4]:

```
for api in myServer.SpectrumServices().Apis:
    print(api)
```

```
RelationshipExtractor
USDatabaseLookup
spectrumspatialpy_route
GetCityStateProvinceLoqate
ValidateAddressGlobal
GetPostalCodes
AddressParser
GeocodeAddressWorld
Centrus
ReverseGeoTAXInfoLookup
ReverseAPNLookup
GlobalAddressValidation
GlobalGeocode
AssignGeoTAXInfo
GetPostalCodesLoqate
TextCategorizer
AutoCompleteLoqate
DataHub
CalculateDistance
ReverseGeocodeUSLocation
GeocodeUSAddress
passthru
ReversePBKeyLookup
ValidateAddress
spectrumpy
GetCandidateAddressesLoqate
GetCityStateProvince
GlobalTypeAhead
OpenNameParser
EntityExtractor
Spatial
GetCandidateAddresses
ValidateAddressLoqate
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 by this library. 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 [5]:

```
s = myServer.SpectrumServices().GeocodeUSAddress(Data_AddressLine1="one glo
                                                 Option_Dataset="egm",
                                                 Option_OutputRecordType="A
print (s)
```

```
    "USBCCheckDigit" : "8",
    "PostalBarCode" : "837101",
    "DeliveryPointCode" : "01",
    "GovernmentBuilding" : "",

    "USLOTCode" : "0053A",
    "USCarrierRouteSort" : "D",
    "USCityDelivery" : "Y",
    "PostalCodeClass" : "",
    "PostalFacility" : "P",
    "PostalCodeUnique" : "",
    "CityStateRecordName" : "Troy",
    "CityPreferredName" : "Troy",
    "CityShortName" : "Troy",
    "Alternate" : "B",
    "HouseNumberHigh" : "1",
    "HouseNumberLow" : "1",
    "HouseNumberParity" : "O",
    "UnitNumberHigh" : "",
    "UnitNumberLow" : "",
```

The services object also exposes a Help function that will print out the detailed list of available function parameters. The following example illustrates the help for the GeocodeUSAddress function.

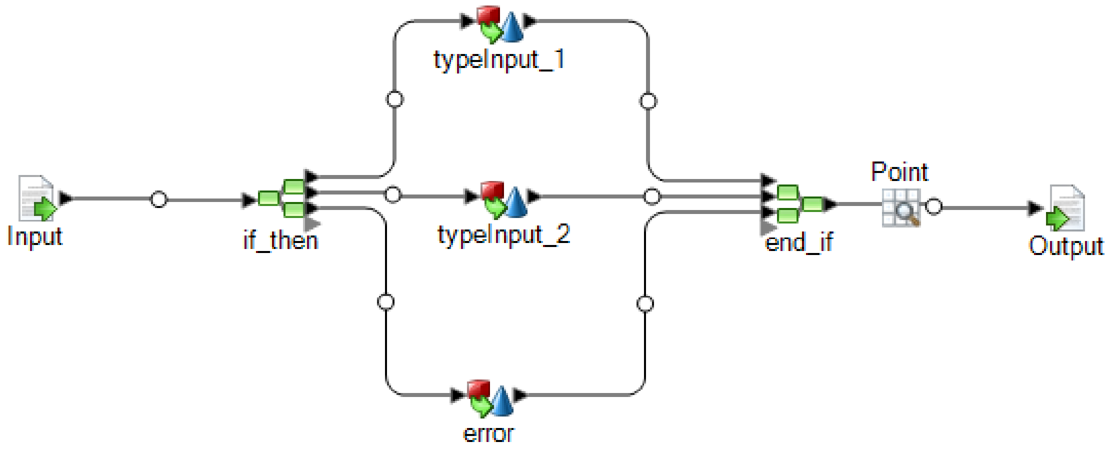
In [6]:

```
myServer.SpectrumServices().Help('GeocodeUSAddress')
```

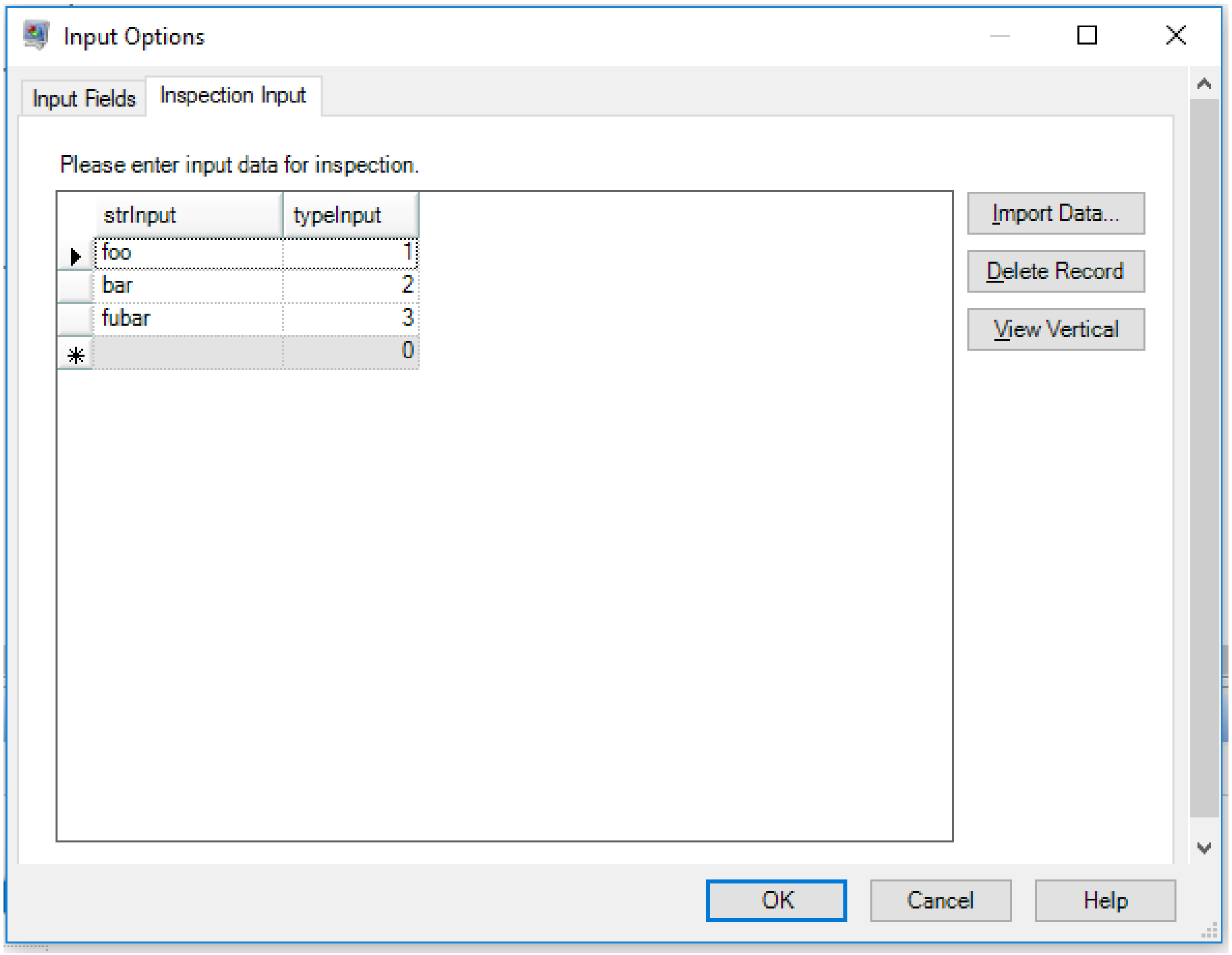
```
RequestType: GET
ContentType: application/json
Method: results.json_GET
URL: http://localhost:8080/rest/GeocodeUSAddress/results.json (http://
localhost:8080/rest/GeocodeUSAddress/results.json)
Arguments:
   Data_FirmName : xs:string => Data.FirmName
   Data_AddressLine1 : xs:string => Data.AddressLine1
   Data_AddressLine2 : xs:string => Data.AddressLine2
   Data_AddressLine3 : xs:string => Data.AddressLine3
   Data_AddressLine4 : xs:string => Data.AddressLine4
   Data_AddressLine5 : xs:string => Data.AddressLine5
   Data_AddressLine6 : xs:string => Data.AddressLine6
   Data_LastLine : xs:string => Data.LastLine
   Data_City : xs:string => Data.City
   Data_StateProvince : xs:string => Data.StateProvince
   Data_PostalCode : xs:string => Data.PostalCode
   Data_Longitude : xs:string => Data.Longitude
   Data_Latitude : xs:string => Data.Latitude
   Option_Dataset : xs:string => Option.Dataset
   Option_AlwaysFindCandidates : xs:string => Option.AlwaysFindCandida
tes
   Option_CenterlineOffset : xs:string => Option.CenterlineOffset
   Option_Offset : xs:string => Option.Offset
   Option_Squeeze : xs:string => Option.Squeeze
   Option_Datum : xs:string => Option.Datum
   Option_CentroidPreference : xs:string => Option.CentroidPreference
   Option_LatLonFormat : xs:string => Option.LatLonFormat
   Option_RetrieveAPN : xs:string => Option.RetrieveAPN
   Option_RetrieveElevation : xs:string => Option.RetrieveElevation
   Option_FallbackToStreet : xs:string => Option.FallbackToStreet
   Option_FallbackToGeographic : xs:string => Option.FallbackToGeograp
hic
   Option_AddressPointInterpolation : xs:string => Option.AddressPoint
Interpolation
   Option_MatchMode : xs:string => Option.MatchMode
   Option_ExtendedMatchCode : xs:string => Option.ExtendedMatchCode
   Option_MustMatchHouseNumber : xs:string => Option.MustMatchHouseNum
ber
   Option_MustMatchStreet : xs:string => Option.MustMatchStreet
   Option_MustMatchCity : xs:string => Option.MustMatchCity
   Option_MustMatchStateProvince : xs:string => Option.MustMatchStateP
rovince
   Option_MustMatchPostalCode : xs:string => Option.MustMatchPostalCod
e
   Option_AddressPreference : xs:string => Option.AddressPreference
   Option_PerformDPV : xs:string => Option.PerformDPV
   Option_PerformLACSLink : xs:string => Option.PerformLACSLink
   Option_PreferZipCodeOverCity : xs:string => Option.PreferZipCodeOve
rCity
   Option_KeepMultimatch : xs:string => Option.KeepMultimatch
   Option_FirmNameSearch : xs:string => Option.FirmNameSearch
   Option_BuildingSearch : xs:string => Option.BuildingSearch
   Option_FirstLetterSearch : xs:string => Option.FirstLetterSearch
   Option_PredictiveLastLine : xs:string => Option.PredictiveLastLine
   Option_KeepCandidates : xs:string => Option.KeepCandidates
   Option_CloseMatchesOnly : xs:string => Option.CloseMatchesOnly
   Option_OutputCasing : xs:string => Option.OutputCasing
   Option_OutputRecordType : xs:string => Option.OutputRecordType
   Option_OutputFields : xs:string => Option.OutputFields
   Option_OutputFormattedOnFail : xs:string => Option.OutputFormattedO
nFail
   Option_OutputPostalCodeSeparator : xs:string => Option.OutputPostal
CodeSeparator
   Option_OutputVerbose : xs:string => Option.OutputVerbose
   Option_FIND_APPROXIMATE_PBKEY : xs:string => Option.FIND_APPROXIMAT
E_PBKEY
   Option_SearchDistance : xs:string => Option.SearchDistance
   Option_FIND_SEARCH_AREA : xs:string => Option.FIND_SEARCH_AREA
   Option_FIND_SEARCH_AREA_DISTANCE : xs:string => Option.FIND_SEARCH_
AREA_DISTANCE
```

### *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:



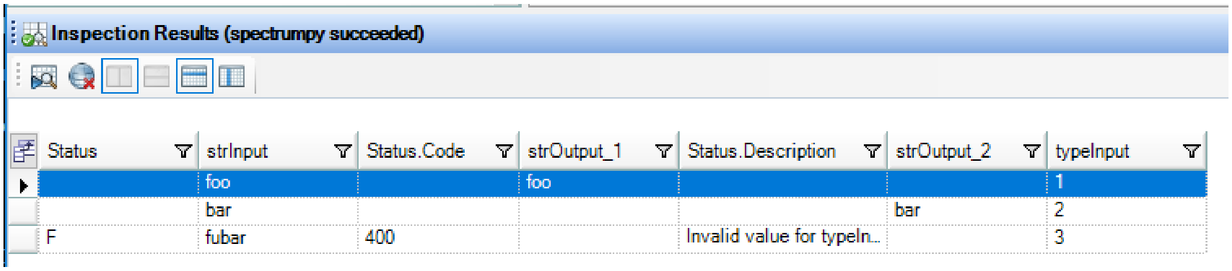
Given the following sample input



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_typeInp
print(s)
s = myServer.SpectrumServices().spectrumpy(Data_strInput="bar",Data_typeInp
print(s)
s = myServer.SpectrumServices().spectrumpy(Data_strInput="fubar",Data_typeI
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 [ ]: