Tweeter travel data service API

This document describes an implementation of tweeter travel data service that provides data about movements of Twitter users. The service currently offers data that are aggregated at varying ranges of spatial granularity. The aggregation unit is 1km*1km grid cells at the granularity level 0, and the cell dimension doubles as the granularity level increases by one level. The service supports granularity levels from 0 (dense) to 9 (coarse). For now, the source data are limited to tweets for the last week of June, 2013. This document explains the interface of the Tweeter travel data service as well as the implementation and deployment of the service.

INTERFACE

Input parameters

- boundary: a string representation of spatial extent of a study region. It should be formatted as '[(lng1,lat1),(lng2,lat2)]' where the two coordinates is the lower-left and upper-right corners of the study region. An example of the boundary parameter for the Illinois state is as follows:

```
[(-91.51666,36.96666),(-87.5,42.5)]
```

- granularity: an integer indicating the level of spatial granularity. This value ranges from 0 to 9.
- travelType: a string indicating the type of travel. The value should be either 'inflow' or 'outflow'.
- travelFields: a string indicating the types of travel data to be returned. It should be a combination of the following values: 'all', 'flu', 'air', and 'airflu'. When multiple values are entered, they should be concatenated by comma (',').
- outputFormat: a string indicating the format of output location data. It should be either 'geojson' or 'esrijson'.

Output parameters

- locationData: a JSON object containing the origins or destinations from/to which tweeters travel. Each origin or destination is represented as Point with a single attribute, ID. If the value for the outputFormat input parameter is 'geojson', the returned object will be structured according GeoJSON format. If the value for the outputFormat parameter is 'esrijson', the return value will be structured according to Esri JSON format. Examples of the returned value are as follows:

```
# Esri JSON example
[
{"geometry": {"y": 37.6607635, "x": -90.46698549999999}, "attributes": {"id":
51380225}},
{"geometry": {"y": 38.7273875, "x": -90.45865249999999}, "attributes": {"id":
53084162}},
{"geometry": {"y": 38.7273875, "x": -90.45031949999999}, "attributes": {"id":
53084163}},
....,
{"geometry": {"y": 38.7273875, "x": -90.44198649999998}, "attributes": {"id":
53084164}}
]

# GeoJSON example
{
"type": "FeatureCollection",
"features": [
{"geometry": {"type": "Point", "coordinates": [-90.46698549999999, 37.6607635]},
"type": "Feature", "properties": {"id": 51380225}},
{"geometry": {"type": "Point", "coordinates": [-90.45865249999999, 38.7273875]},
"type": "Feature", "properties": {"id": 53084162}},
{"geometry": {"type": "Point", "coordinates": [-90.45031949999999, 38.7273875]},
"type": "Feature", "properties": {"id": 53084163}},
{"geometry": {"type": "Point", "coordinates": [-90.44198649999998, 38.7273875]},
"type": "Feature", "properties": {"id": 53084164}},
...,
{"geometry": {"type": "Point", "coordinates": [-90.43365349999999, 38.7273875]},
"type": "Feature", "properties": {"id": 53084165}}
]
}
```

- travelData: a JSON object containing the magnitude of travels from an origin to a destination. Each travel is represented as a line or path that connects an origin to a destination and have attributes of travel magnitudes. If the value for the outputFormat input parameter is 'geojson', the returned object will be structured according GeoJSON format. If the value for the outputFormat parameter is 'esrijson', the return value will be structured according to Esri JSON format. Examples of the returned value are as follows:

```
# Esri JSON example
[
{"geometry": {"paths": [[[-89.16703749999999, 36.9941235],
[-89.17537049999999, 36.9941235]]]}, "attributes": {"all": 1}}, {"geometry":
{"paths": [[[-89.15870449999998, 36.9941235], [-89.14203849999998,
36.9857905]]]}, "attributes": {"all": 1}},
...,
{"geometry": {"paths": [[[-89.15870449999998, 36.9941235],
[-89.20870249999999, 37.0191225]]]}, "attributes": {"all": 1}}, {"geometry":
{"paths": [[[-89.15870449999998, 36.9941235], [-89.16703749999999,
37.0024565]]]}, "attributes": {"all": 3}}
]

# GeoJSON example
{
"type": "FeatureCollection",
"features": [
{"geometry": {"type": "LineString", "coordinates": [[-89.16703749999999,
36.9941235], [-89.17537049999999, 36.9941235]]}, "type": "Feature",
"properties": {"all": 1}},
{"geometry": {"type": "LineString", "coordinates": [[-89.15870449999998,
36.9941235], [-89.14203849999998, 36.9857905]]}, "type": "Feature",
"properties": {"all": 1}},
...
{"geometry": {"type": "LineString", "coordinates": [[-89.15870449999998,
36.9941235], [-89.16703749999999, 37.0024565]]}, "type": "Feature",
"properties": {"all": 3}}
]
}
```

Service API

This service supports GetCapabilities, DescribeProcess, and Execute requests as in any other OGC WPS services. GetCapabilities request is less relevant to the specifics of the service. This section thus details how to make DescribeProcess and Execute requests.

DescribeProcess

- Request

```
http://gwdev5.cigi.illinois.edu:8080/cgi-bin/pywps.cgi?service=wps&version=1.0
.0&request=describeprocess&identifier=traveldataprocess
```

- Response

```
<?xml version="1.0" encoding="utf-8"?>
<wps:ProcessDescriptions xmlns:wps="http://www.opengis.net/wps/1.0.0"
xmlns:ows="http://www.opengis.net/ows/1.1"
```

```
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/wps/1.0.0
http://schemas.opengis.net/wps/1.0.0/wpsDescribeProcess_response.xsd"
service="WPS" version="1.0.0" xml:lang="en-CA">
    <ProcessDescription wps:processVersion="0.1" storeSupported="true"
statusSupported="true">
        <ows:Identifier>traveldataprocess</ows:Identifier>
        <ows:Title>Tweeter travel data service</ows:Title>
        <ows:Abstract>Tweeter travel data service</ows:Abstract>
        <DataInputs>
            <Input minOccurs="1" maxOccurs="1">
                <ows:Identifier>boundary</ows:Identifier>
                <ows:Title>the spatial extent within which tweeters travel; a
list of two coordinates</ows:Title>
                <LiteralData>
                    <ows:DataType
ows:reference="http://www.w3.org/TR/xmlschema-2/#string">string</ows:DataType>
                    <ows:AnyValue />
                </LiteralData>
            </Input>
            <Input minOccurs="1" maxOccurs="1">
                <ows:Identifier>travelFields</ows:Identifier>
                <ows:Title>types of travel data; can be one or more types
among all, flu, air, and airflu</ows:Title>
                <LiteralData>
                    <ows:DataType
ows:reference="http://www.w3.org/TR/xmlschema-2/#string">string</ows:DataType>
                    <ows:AnyValue />
                </LiteralData>
            </Input>
            <Input minOccurs="1" maxOccurs="1">
                <ows:Identifier>outputFormat</ows:Identifier>
                <ows:Title>format of output location data; either geojson or
esrijson</ows:Title>
                <LiteralData>
                    <ows:DataType
ows:reference="http://www.w3.org/TR/xmlschema-2/#string">string</ows:DataType>
                    <ows:AnyValue />
                </LiteralData>
            </Input>
            <Input minOccurs="1" maxOccurs="1">
                <ows:Identifier>travelType</ows:Identifier>
                <ows:Title>type of travel; either inflow (travel to the
provided locations from others) or outflow (travel from the provided locations
to others) </ows:Title>
                <LiteralData>
                    <ows:DataType
ows:reference="http://www.w3.org/TR/xmlschema-2/#string">string</ows:DataType>
                    <ows:AnyValue />
                </LiteralData>
            </Input>
            <Input minOccurs="1" maxOccurs="1">
                <ows:Identifier>granularity</ows:Identifier>
                <ows:Title>the level of spatial granularity at which tweeters'
travels are aggregated; an integer from 0 and 9; at level 0, the size of
aggregation unit is 1km*1km</ows:Title>
                <LiteralData>
                    <ows:DataType
```

```
ows:reference="http://www.w3.org/TR/xmlschema-2/#integer">integer</ows:DataTyp
e>
                    <ows:AnyValue />
                </LiteralData>
            </Input>
        </DataInputs>
        <ProcessOutputs>
            <Output>
                <ows:Identifier>locationData</ows:Identifier>
                <ows:Title>centers of cells within the input
boundary</ows:Title>
                <ComplexOutput>
                    <Default>
                        <Format>
                            <MimeType>text/plain</MimeType>
                        </Format>
                    </Default>
                    <Supported>
                        <Format>
                            <MimeType>text/plain</MimeType>
                        </Format>
                    </Supported>
                </ComplexOutput>
            </Output>
            <Output>
                <ows:Identifier>travelData</ows:Identifier>
                <ows:Title>a list of lines whose attributes are the magnitude
of travels</ows:Title>
                <ComplexOutput>
                    <Default>
                        <Format>
                            <MimeType>text/plain</MimeType>
                        </Format>
                    </Default>
                    <Supported>
                        <Format>
                            <MimeType>text/plain</MimeType>
                        </Format>
                    </Supported>
                </ComplexOutput>
            </Output>
```

```
        </ProcessOutputs>
    </ProcessDescription>
</wps:ProcessDescriptions>
```

Execute

- Request

```
http://gwdev5.cigi.illinois.edu:8080/cgi-bin/pywps.cgi?service=wps&version=1.0
.0&request=execute&identifier=traveldataprocess&datainputs=[boundary=[(-91.516
66,36.96666),(-87.5,42.5)];granularity=0;travelType=outflow;travelFields=all;o
utputFormat=esrijson]
```

- Response

```
<?xml version="1.0" encoding="utf-8"?>
<wps:ExecuteResponse xmlns:wps="http://www.opengis.net/wps/1.0.0"
xmlns:ows="http://www.opengis.net/ows/1.1"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/wps/1.0.0
http://schemas.opengis.net/wps/1.0.0/wpsExecute_response.xsd" service="WPS"
version="1.0.0" xml:lang="en-CA"
serviceInstance="http://gwdev5.cigi.illinois.edu:8080/cgi-bin/pywps.cgi?servic
e=WPS&amp;request=GetCapabilities&amp;version=1.0.0">
    <wps:Process wps:processVersion="0.1">
        <ows:Identifier>traveldataprocess</ows:Identifier>
        <ows:Title>Tweeter travel data service</ows:Title>
        <ows:Abstract>Tweeter travel data service</ows:Abstract>
    </wps:Process>
    <wps:Status creationTime="2013-07-17T21:23:27Z">
        <wps:ProcessSucceeded>PyWPS Process traveldataprocess successfully
calculated</wps:ProcessSucceeded>
    </wps:Status>
    <wps:ProcessOutputs>
        <wps:Output>
            <ows:Identifier>locationData</ows:Identifier>
            <ows:Title>centers of cells within the input boundary</ows:Title>
            <wps:Reference
href="http://gwdev5.cigi.illinois.edu:8080/wps/wpsoutputs/locationData-01e299f
8-ef51-11e2-a1c2-005056af2cee.json" mimeType="text/plain" />
        </wps:Output>
        <wps:Output>
            <ows:Identifier>travelData</ows:Identifier>
            <ows:Title>a list of lines whose attributes are the magnitude of
travels</ows:Title>
            <wps:Reference
href="http://gwdev5.cigi.illinois.edu:8080/wps/wpsoutputs/travelData-01e299f8-
ef51-11e2-a1c2-005056af2cee.json" mimeType="text/plain" />
        </wps:Output>
    </wps:ProcessOutputs>
</wps:ExecuteResponse>
```

IMPLEMENTATION OF THE TWEETER TRAVEL DATA SERVICE

The tweeter travel data service was implemented by using pymongo and PyWPS. This implementation was done on a virtual machine with Ubuntu Server 12. This section describes three steps of the implementation.

Step 1. Server configuration: Web Server and PyWPS

To host the tweeter data service, a server needs apache web server, PyWPS, and pymongo. This subsection describes the installation of apache web server and PyWPS on Ubuntu Server 12. The following web pages can be referred to obtain more detailed information of this step.

- http://geotechrichard.files.wordpress.com/2011/09/pywps_basic_install.pdf
- http://wiki.ieee-earth.org/Documents/GEOSS_Tutorials/GEOSS_Provider_Tutorials/Web_Processing_Service_Tutorial_for_GEOSS_Providers/Section_4%3A_Provisioning%2F%2FUsing_the_Service_or_Application/Steps_for_Plymouth_Marine_Laboratory_implementation_of_Python_WPS

1) Install and configure apache

```
sudo su # in the below, root user is assumed
apt-get install apache2 apache2-doc apache2-utils
# make sure apache can run CGI applications
# make sure to include the following directive in
/etc/apache2/sites-enabled/000-default
ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
<Directory "/usr/lib/cgi-bin">
 AllowOverride None
 Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
 Order allow,deny
 Allow from all
</Directory>
```

2) Install python pip for fast installation of third-party python packages

```
apt-get install python-pip
```

3) Install dependencies of PyWPS (i.e., htmltmpl and magic)

```
apt-get install python-htmltmpl python-magic python-lxml
```

4) Download and install PyWPS-3.2.1

In this example, PyWPS-3.2.1 is downloaded into /usr/local and the associated files for configuration and processes are stored in /usr/local/wps.

```
cd /usr/local
wget --no-check-certificate https://github.com/geopython/PyWPS/archive/master.zip .
unzip master.zip
cp -R PyWPS-master pywps-3.2.1
cd pywps-3.2.1
python setup.py install --dry-run
#to test the installation, run wps.py; a html output should show up
```

5) Configure and test PyWPS-3.2.1

5-1) Copy the default configuration file for PyWPS

```
mkdir /usr/local/wps
cp /usr/local/pywps-3.2.1/pywps/default.cfg /usr/local/wps/pywps.cfg
```

5-2) Create folders to hold user-defined PyWPS processes and a log file

```
mkdir /usr/local/wps/processes
mkdir /usr/local/wps/log
chmod -R 777 /usr/local/wps/log/
```

5-3) Create a test WPS process

```
cp /usr/local/pywps-3.2.1/tests/processes/ultimatequestionprocess.py
/usr/local/wps/processes/ultimatequestionprocess.py
cd /usr/local/wps/processes
vim __init__py
in __init__.py, add a line saying __all__ = ["ultimatequestionprocess"]
```

5-4) Create a test WPS process

```
cd ..
vim pywps.cfg
add/modify pywps.cfg:
[wps]
serveraddress=http://gwdev5.cigi.illinois.edu:8080/cgi-bin/pywps.cgi
[server]
maxinputparamlength=4096 # for easy test of get requests, increase the length of
input payload
maxfilesize=3mb
tempPath=/usr/local/wps/tmp
processesPath=/usr/local/wps/processes
outputUrl=http://gwdev5.cigi.illinois.edu:8080/wps/wpsoutputs
outputPath=/var/www/wps/wpsoutputs
logFile=/usr/local/wps/log/pywps.log
logLevel=DEBUG
```

5-5) Deploy PyWPS processes as CGI applications

For simplicity, this example deploys PyWPS processes as CGI applications. In Ubuntu, CGI applications are usually located in /usr/lib/cgi-bin

```
cp /usr/local/pywps-3.2.1/webservices/cgi/pywps.cgi /usr/lib/cgi-bin
vim /usr/lib/cgi-bin/pywps.cgi
in pywps.cigi, type:
#!/bin/shcd /
export PYWPS_CFG=/usr/local/wps/pywps.cfg
export PYWPS_PROCESSES=/usr/local/wps/processes
/usr/local/bin/wps.py

chmod +x /usr/lib/cgi-bin/pywps.cgi
```

5-6) Create a web-accessible folder where output data from WPS processes will be written

```
mkdir -p /var/www/wps/wpsoutputs
chown -R www-data /var/www/wps
chmod -R g+w /var/www/wps
chmod 777
/usr/local/lib/python2.7/dist-packages/pywps-3.2.1-py2.7.egg/pywps/Templates/1_0_0/
```

5-7) Create a folder where WPS processes can write temporary data

```
mkdir -p /usr/local/wps/tmp
chmod 777 /usr/local/wps/tmp
```

5-8) Check utility tools for local testing of WPS processes

```
# check the wps.py executable
# In terminal, type the following command. You need to see some hypertext outputs:
wps.py

# to test wps interactions in a local mode, type the following commands:
export PYWPS_CFG=/usr/local/wps/pywps.cfg
export PYWPS_PROCESSES=/usr/local/wps/processes
wps.py "service=wps&request=getcapabilities"
wps.py
"service=wps&version=1.0.0&request=describeprocess&identifier=ultimatequestionproces
s"
wps.py
"service=wps&version=1.0.0&request=execute&identifier=ultimatequestionprocess"
```

Step 2. Server configuration: pymongo and others

The tweeter data service uses OGC WPS interface to CIGI's database of tweeter's movements. To allow for access to the database, pymongo, the python driver of MongoDB, needs to be installed first, along with pytz. The service also needs the bespoke python library that was developed to collect tweets data from Twitter's streaming API. This library is called TweetsDB and was installed on the WPS server. However, at this point the source code or repository URL of TweetsDB is not released here since the library is still under development. Since TweetsDB uses PySAL, here the installation of PySAL and its dependencies is described.

1) install pymongo and pytz

```
sudo pip install pymongo
sudo pip install pytz
```

2) Install git, subversion, numpy and scipy (PySAL's dependencies)

```
sudo apt-get install subversion
sudo apt-get install git
sudo apt-get install python-dev
sudo apt-get install python-numpy
sudo apt-get install python-scipy
```

3) Retrieve the latest versions of PySAL

In this example, they are installed in /usr/local

```
git clone https://github.com/pysal/pysal.git pysal
```

3) Install and configure TweetsDB

```
cd /usr/local
svn co https://svn.cigi.uiuc.edu/socialmedia/TweetsDB/branches/hourly-flows TweetsDB
```

Step 3. Implementation of the tweeter travel data WPS process

1) FlowDataQuery module

To facilitate the formulation of MongoDB query statements from user-provided input parameters, a module named FlowDataQuery were developed within /usr/local/wps/processes.

```
# FlowDataQuery

import pymongo
from TweetsDB.common.Utils import getCellId, getCellCenter, getRowColumn
from TweetsDB.common.Config import COL_NO
DBHOST = 'HOST'
DBPORT = 27017
DB = 'DB'
COLL = 'testTravelData'

class FlowDataQuery(object):

    locations = []
    granularity = 3
    travelType = 'outflow'
    boundary = []
    travelFields = ['all', 'flu', 'air', 'airflu']
    locationFormat = 'esrijson'

    def __init__(self, boundary, granularity, travelType, travelFields,
locationFormat):
        self.cellIds = {}
        self.granularity = granularity
        self.travelType = travelType
        self.flowField = None
        self.boundary = boundary
        self.travelFields = travelFields
        self.attrIndexes = []
        self.locationFormat = locationFormat

        self.query = None
        self.fields = None

        self.con = None
        self.db = None
        self.col = None
        self.dbCursor = None

    def run(self):
        self.setQueryCondition()
        self.runQuery()
```

```
        res = None
        if self.dbCursor:
            res = self.processResults()
        self.closeDB()
        return res

    def getCellIds(self):
        lowerLeft = self.boundary[0]
        upperRight = self.boundary[1]
        lowerLeftCellId = getCellId(lowerLeft[1], lowerLeft[0], self.granularity)
        lowerLeftRow, lowerLeftColumn = tuple(getRowColumn(lowerLeftCellId,
self.granularity))
        upperRightCellId = getCellId(upperRight[1], upperRight[0], self.granularity)
        upperRightRow, upperRightColumn = tuple(getRowColumn(upperRightCellId,
self.granularity))
        cellIds = []
        for r in xrange(lowerLeftRow, upperRightRow + 1):
            for c in xrange(lowerLeftColumn, upperRightColumn + 1):
                cellIds.append(int(r*COL_NO[self.granularity] + c))
        self.cellIds = cellIds

    def setQueryCondition(self):
        self.getCellIds()
        cellIds = self.cellIds
        cellIds = sorted(cellIds)
        self.query = {'level':self.granularity, 'cell_id':{"$in":cellIds}}
        self.fields = {'cell_id':1, 'level':1}
        if self.travelType.lower() == 'outflow':
            self.flowField = 'out_flows'
            self.fields[self.flowField] = 1
        elif self.travelType.lower() == 'inflow':
            self.flowField = 'in_flows'
            self.fields[self.flowField] = 1

    def runQuery(self):
        self.con = pymongo.MongoClient(DBHOST, DBPORT)
        self.db = self.con[DB]
        self.col = self.db[COLL]
        dbCursor = self.col.find(self.query)
        #print dbCursor.count()
        if dbCursor.count() > 0:
            self.dbCursor = dbCursor

    def closeDB(self):
        self.con.close()

    def checkAttrIndexes(self):
        if 'all' in self.travelFields:
            self.attrIndexes.append(0)
        if 'flu' in self.travelFields:
            self.attrIndexes.append(1)
        if 'air' in self.travelFields:
            self.attrIndexes.append(2)
        if 'airflu' in self.travelFields:
            self.attrIndexes.append(3)
        self.attrIndexes.sort()

    def withinBoundary(self, aPoint):
        if not self.boundary:
```

```
            return True
        left, lower = self.boundary[0]
        right, upper = self.boundary[1]
        lng, lat = aPoint
        if lng < left or lng > right:
            return False
        if lat < lower or lat > upper:
            return False
        return True

    def getFlowGeoJSON(self, flowData):
        featColl = {"type":"FeatureCollection", "features":[]}
        for loc1, loc2, attr in flowData:
            feature = {"type":"Feature"}
            feature["geometry"] = {"type":"LineString"}
            feature["geometry"]["coordinates"] = [list(loc1), list(loc2)]
            feature["properties"] = {}
            for attrInx in self.attrIndexes:
                prop = self.travelFields[attrInx]
                feature["properties"][prop] = attr[attrInx]
            featColl["features"].append(feature)
        return featColl

    def getFlowEsriJSON(self, flowData):
        featColl = []
        for loc1, loc2, attr in flowData:
            feature = {"geometry":{}, "attributes":{}}
            feature["geometry"]["paths"] = [[list(loc1), list(loc2)]]
            for attrInx in self.attrIndexes:
                prop = self.travelFields[attrInx]
                feature["attributes"][prop] = attr[attrInx]
            featColl.append(feature)
        return featColl

    def getLocationGeoJSON(self, locations):
        featColl = {"type":"FeatureCollection", "features":[]}
        for cellId in locations:
            feature = {"type":"Feature"}
            feature["geometry"] = {"type":"Point"}
            feature["geometry"]["coordinates"] = list(locations[cellId])
            feature["properties"] = {"id": cellId}
            featColl["features"].append(feature)
        return featColl

    def getLocationEsriJSON(self, locations):
        featColl = []
        for cellId in locations:
            feature = {}
            loc = locations[cellId]
            feature["geometry"] = {"x":loc[0],"y":loc[1]}
            feature["attributes"] = {"id": cellId}
            featColl.append(feature)
        return featColl

    def processResults(self):
        results = None
        self.checkAttrIndexes()
        finalCells = {}
        output = []
```

```
for cell in self.dbCursor:
    cellId = cell['cell_id']
    cellLoc = getCellCenter(cellId, self.granularity)
    if cellId not in finalCells:
        finalCells[cellId] = cellLoc
    flowData = cell[self.flowField]
    for otherCell in flowData:
        otherCellLoc = getCellCenter(int(otherCell), self.granularity)
        if not self.withinBoundary(otherCellLoc):
            continue
        if otherCell not in finalCells:
            finalCells[otherCell] = otherCellLoc
        attr = [flowData[otherCell][i] for i in self.attrIndexes]
        output.append((cellLoc, otherCellLoc, attr))

locationData = None
if len(finalCells) > 0:
    if self.locationFormat == 'geojson':
        locationData = self.getLocationGeoJSON(finalCells)
    elif self.locationFormat == 'esrijson':
        locationData = self.getLocationEsriJSON(finalCells)

flowGeoJSON = None
if len(output) > 0:
    if self.locationFormat == 'geojson':
        flowGeoJSON = self.getFlowGeoJSON(output)
    elif self.locationFormat == 'esrijson':
```

```
                flowGeoJSON = self.getFlowEsriJSON(output)

        return locationData, flowGeoJSON
```

2) Write the tweeter travel data WPS process

Write the following module titled traveldataprocess.py in /usr/local/wps/processes

```
# traveldataprocess.py
from pywps.Process import WPSProcess
import types, sys, json, logging, tempfile
class Process(WPSProcess):
    def __init__(self):
         # init process
        WPSProcess.__init__(self,
            identifier="traveldataprocess",
            title="Tweeter travel data service",
            version = "0.1",
            storeSupported = "true",
            statusSupported = "true",
            abstract="Tweeter travel data service",
            grassLocation =False)

        # inputs
        self.boundary = self.addLiteralInput(identifier="boundary",
            title="the spatial extent within which tweeters travel; a list of two
coordinates",
            type=types.StringType)
        self.granularity = self.addLiteralInput(identifier="granularity",
            title="the level of spatial granularity at which tweeters' travels are
aggregated; an integer from 0 and 9; at level 0, the size of aggregation unit is
1km*1km",
            type=types.IntType)
        self.travelType = self.addLiteralInput(identifier="travelType",
            title="type of travel; either inflow (travel to the provided locations
from others) or outflow (travel from the provided locations to others) ",
            type=types.StringType)
        self.travelFields = self.addLiteralInput(identifier="travelFields",
            title= "types of travel data; can be one or more types among all, flu,
air, and airflu",
            type=types.StringType)
        self.outputFormat = self.addLiteralInput(identifier="outputFormat",
            title= "format of output location data; either geojson or esrijson",
            type=types.StringType)
        self.locationData = self.addComplexOutput(identifier="locationData",
            title="centers of cells within the input boundary",
            formats='text/plain',
            asReference=True)
        self.travelData = self.addComplexOutput(identifier="travelData",
            title="a list of lines whose attributes are the magnitude of travels",
            formats='text/plain',
            asReference=True)

    def execute(self):
        from FlowDataQuery import FlowDataQuery
```

```
        self.status.set("Preparing....", 0)
        boundary = eval(self.boundary.value)
        granularity = 0
        if self.granularity.value:
            granularity = self.granularity.value
        travelType = "outflow"
        if self.travelType.value:
            travelType = self.travelType.value
        locationFormat = "esrijson"
        if self.outputFormat.value:
            locationFormat = self.outputFormat.value.lower()
        travelFields = ['all']
        if self.travelFields not in ["", None, "all"]:
            travelFields = self.travelFields.value.split(',')
        queryTool = FlowDataQuery(boundary, granularity, travelType, travelFields,
locationFormat)

        self.status.set("Starting to query....", 10)
        locationData, flowData = queryTool.run()
        self.status.set("Returning output....", 90)
        jsonExt = '.geojson'
        if locationFormat == 'esrijson':
            jsonExt = '.json'


locDataFile=tempfile.NamedTemporaryFile(suffix=jsonExt,prefix='locData',delete=False
)
        locDataFile.write(json.dumps(locatioData))
        self.locationData.setValue(locDataFile.name)


travelDataFile=tempfile.NamedTemporaryFile(suffix=jsonExt,prefix='travelData',delete
=False)
```

```
        travelDataFile.write(json.dumps(flowData))
        self.travelData.setValue(travelDataFile.name)
        return
```

3) Register the tweeter travel data WPS process

Modify __init__.py in /usr/local/wps/processes

```
vim __init__py
in __init__.py, replace __all__ = ["ultimatequestionprocess"] with __all__ =
["traveldataprocess"]
```

5) Test

```
# to test wps interactions in a local mode, type the following commands:
export PYWPS_CFG=/usr/local/wps/pywps.cfg
export PYWPS_PROCESSES=/usr/local/wps/processes
wps.py "service=wps&request=getcapabilities"
wps.py
"service=wps&version=1.0.0&request=describeprocess&identifier=traveldataprocess"
wps.py
"service=wps&version=1.0.0&request=execute&identifier=traveldataprocess&datainputs=[
boundary=[(-91.51666,36.9666
6),(-87.5,42.5)];granularity=0;travelType=outflow;travelFields=all;outputFormat=esri
json]"
```