Geocode an address and plot it on an orthophoto in the coterminous US using MapScript

> You want to use Python MapScript to geocode addresses and return a DOQ map with the address plotted on it as a point.

Geocoding is very popular these days. Users frequently ask questions on the MapServer list how to incorporate geocoding into existing applications. Websites like MapQuest, Yahoo! Maps, and Google Maps have popularized the concept of using an address as an initial map navigation tool.

The process of geocoding, or turning a street address into a longitude/latitude pair, is a difficult one. It requires two key components – complete and specialized data, and algorithms to turn that data into coordinates. Luckily, Schuyler Erle of *Mapping Hacks* has an Open Source geocoding solution available. Using the US Census Tiger street data, http://geocoder.us provides Perl software and a SOAP and XMLRPC remote query mechanism for geocoding. This means you can download your own copy of the Tiger database and provide geocoding for your own commercial applications. (In fact, if you plan to use the software commercially, you **must** build your own database rather than utilizing the remote query interface provided by geocoder.us.)

Getting a lat/lon for an address

We will be using the remote query functionality of geocoder.us today. Start by importing the *xmlrpc* library and setting up a proxy to geocoder.us's XMLRPC service.

```
1   import xmlrpclib
2   geocode_url = 'http://rpc.geocoder.us/service/xmlrpc'
3   p = xmlrpclib.ServerProxy(geocode_url)
4   address = "615 WASHINGTON AVE SOUTHEAST, MINNEAPOLIS, MN"
```

Next, execute a call to the XMLRPC service

```
5   result = p.geocode(address)
6   print result
7   C:\>geocode.py
8   [{'city': 'Minneapolis', 'prefix': '', 'suffix': 'SE', 'zip': 55455,
    'number': 6
9   15, 'long': -93.229894000000002, 'state': 'MN', 'street': 'Washington',
    'lat': 4
10  4.973664999999997, 'type': 'Ave'}]
```

Just like that, we have our coordinates. The XMLRPC service returns a list of dictionaries that we can use to get our coordinate information. Next, we'll import mapscript and setup a mapObj that will draw our map.

© *Howard Butler*

Generating a basic DOQ map with TerraServer and MapScript

```
11  try:
12      import mapscript.mapscript as mapscript
13  except ImportError:
14      import mapscript
15
16  amap = mapscript.mapObj()
```

Our mapObj is named amap because *map* is a function name in python. Normally, you would instantiate a mapObj with an existing mapfile. In our case, however, we want the entire thing to be self-contained in the script. We will build up our mapObj, layerObj's, and styling all in mapscript…

The other kind of funny thing we do here compensates for the way the mapscript package is installed on the workshop machines. We try to import our workshop mapscript, and if it isn't there in the package format, we just try to import it the regular way.

```
17  amap.height = 800
18  amap.width = 1100
19
20  debug = 1
21  ms_debug = 0
22
23  if ms_debug:
24      amap.debug = mapscript.MS_ON
25  amap.setProjection('init=epsg:2163')
```

We next define our map width and height. In addition, we define some debugging variables. This will allow us to see and set some diagnostics as we develop that we can easily turn off once we have a finished script. A more sophisticated approach would utilize Python's *logging* module, but this is a short hack, right?

```
26  lat, lon = result[0]['lat'], result[0]['long']
27  pt = mapscript.pointObj()
28  pt.x = lon
29  pt.y = lat
30
31  if debug:
32      print '-------  DD Coordinates  -------'
33      print 'x: %s y: %s' % (pt.x, pt.y)
34      print '--------------------------------'
```

We need to take the coordinates that the geocoder gave us and turn them into a pointObj. We'll project that point into our mapObj's coordinate system, use it to define the extent of the map, and then use it to plot a point showing the location of the address.

© Howard Butler

```
35  ddproj = mapscript.projectionObj('proj=latlong,ellps=WGS84')
36  origproj = mapscript.projectionObj(amap.getProjection())
37  pt.project(ddproj,origproj)
38
39  if debug:
40      print '-----  Albers Coordinates  -----'
41      print 'x: %s y: %s' % (pt.x, pt.y)
42      print '--------------------------------'
```

Now that we've projected a point into our mapObj's coordinate system, we'll make an extent by adding some buffer (in map units, not decimal degrees).

```
43  buffer = 600
44  extent = mapscript.rectObj()
45  extent.minx = pt.x - buffer
46  extent.miny = pt.y - buffer
47  extent.maxx = pt.x + buffer
48  extent.maxy = pt.y + buffer
49  amap.setExtent(extent.minx, extent.miny,
50                 extent.maxx, extent.maxy)
```

The last few mapObj properties we need to set have to do with the output format and giving the webObj a place to store temporarily downloaded WMS requests.

```
51  outputformat = mapscript.outputFormatObj('GD/JPEG')
52  amap.setOutputFormat( outputformat)
53
54  amap.web.imagepath = os.environ['TEMP']
```

Now that we have a mapObj, we create a layerObj to describe the TerraServer WMS connection. The gotchas here to remember are to make sure you set metadata for the *wms_srs* and *wms_title* and make sure to set the projection of the layer to EPSG 4326.

```
55  layer = mapscript.layerObj(amap)
56  layer.connectiontype = mapscript.MS_WMS
57  layer.type = mapscript.MS_LAYER_RASTER
58  layer.metadata.set('wms_srs','EPSG:4326')
59  layer.metadata.set("wms_title", "USGS Digital Ortho-Quadrangles")
60  ts_url =
    "http://terraservice.net/ogcmap.ashx?VERSION=1.1.1&SERVICE=wms&LAYERS=DOQ&F
    ORMAT=jpeg&styles="
61  layer.connection = ts_url
62  layer.setProjection('init=epsg:4326')
63  layer.status = mapscript.MS_ON
```

Currently, we have a map with only a black and white orthophoto from TerraServer, centered on the latitiude/longitude point that geocoder.us returned to us. Pretty boring, I admit.

© Howard Butler

```
64  img = amap.draw()
65  f = open(r'c:\foo.jpg','wb')
66  f.write(img.getBytes())
67  f.close()
```

Symbology

MapScript doesn't currently support the ability to add a FontSet using only MapScript…it needs a mapObj that was defined from a mapfile to do that. This ability will be added in the near future. Because we're attempting to build a map with no mapfile, we'll instead use a pixelmap as our symbol. We won't use one on our local machine, however – we'll use *urllib2* and *StringIO* to download and save our pixelmap symbol and dynamically incorporate it in our map.

I googled for a house symbol on Google's image search. I found one at http://www.worldcommunitygrid.org/images/agent/house.jpg, but you could substitute any URL to a small jpg that you wanted. Next, we'll download the image and stuff it into a cStringIO instance. This will allow MapScript's image reading machinery to treat it as it would any normal file.

```
68  url = 'http://www.worldcommunitygrid.org/images/agent/house.jpg'
69  f = urllib2.urlopen(url).read()
70  f = cStringIO.StringIO(f)
```

The next bits were cribbed from http://ms.gis.umn.edu/docs/howto/mapscript_imagery. We create a new symbol, set it to type MS_SYMBOL_PIXMAP, give the symbol the imagery, and append it to the mapObj's symbolset.

```
71  symbol = mapscript.symbolObj('from_img')
72  symbol.type = mapscript.MS_SYMBOL_PIXMAP
73  img = mapscript.imageObj(f)
74  symbol.setImage(img)
75  symbol_index = amap.symbolset.appendSymbol(symbol)
```

With the symbol in hand, we can now go through the process of creating a point and a MS_INLINE layer to place our house pixelmap on the map. We first have to build up a shapeObj. shapeObjs are composed of lineObjs, which themselves are composed of points.

```
76  line = mapscript.lineObj()
77  line.add(pt)
78  shape=mapscript.shapeObj(mapscript.MS_SHAPE_POINT)
79  shape.add(line)
80  shape.setBounds()
```

Now that we have our shapeObj, we can build up an inline layer, add our point to it, and set some properties on the layer.

© *Howard Butler*

```
81  inline_layer = mapscript.layerObj(amap)
82  inline_layer.addFeature(shape)
83  inline_layer.setProjection(amap.getProjection())
84  inline_layer.name = "housept"
85  inline_layer.type = mapscript.MS_LAYER_POINT
86  inline_layer.connectiontype=mapscript.MS_INLINE
87  inline_layer.status = mapscript.MS_ON
88  inline_layer.transparency = mapscript.MS_GD_ALPHA
```

With our layer built, we can add our symbology to it. Notice we create a classObj on the inline layer, give it a name, and then create a styleObj that points to our symbol. This is the preferred way of doing styling in MapScript (styleObjs inside of classObjs) instead of merely putting the symbology on the classObj.

```
89  cls = mapscript.classObj(inline_layer)
90  cls.name='classname'
91  style = mapscript.styleObj(cls)
92  style.symbol = amap.symbolset.index('from_img')
```

Finally, draw our map again and save it to a temporary file.

```
93  img = amap.draw()
94
95  f = open(r'c:\temp\foo.jpg','wb')
96  f.write(img.getBytes())
97  f.close()
```

Figure 1. Our final output.

© Howard Butler

Code Listing

```
1   # ---------------------------------------
2   # Geocoding and MapScript
3   # (c) 2005 Howard Butler
4   # hobu@iastate.edu
5   # ---------------------------------------
6
7   import os
8   import xmlrpclib
9
10  # geocode our address
11  geocode_url = 'http://rpc.geocoder.us/service/xmlrpc'
12  p = xmlrpclib.ServerProxy(geocode_url)
13  address = "615 WASHINGTON AVE SOUTHEAST, MINNEAPOLIS, MN"
14
15  result = p.geocode(address)
16  print result
17
18  # import mapscript as package first ... then the regular way
19  try:
20      import mapscript.mapscript as mapscript
21  except ImportError:
22      import mapscript
23
24  amap = mapscript.mapObj()
25
26  amap.height = 800
27  amap.width = 1100
28
29  debug = 1
30  ms_debug = 0
31
32  if debug:
33      print
34      print '------- Address  --------'
35      print '%s' % address
36      print '-------------------------'
37
38  if ms_debug:
39      amap.debug = mapscript.MS_ON
40
41  # set projection to US laea
42  amap.setProjection('init=epsg:2163')
43
44  # grab the first address geocoder.us gives back to us
45  # and turn it into a pointObj
46  lat, lon = result[0]['lat'], result[0]['long']
47  pt = mapscript.pointObj()
48  pt.x = lon
49  pt.y = lat
```


© Howard Butler

```
50
51  if debug:
52      print '-------  DD Coordinates  -------'
53      print 'x: %s y: %s' % (pt.x, pt.y)
54      print '--------------------------------'
55
56  # project our point into the mapObj's projection
57  ddproj = mapscript.projectionObj('proj=latlong,ellps=WGS84')
58  origproj = mapscript.projectionObj(amap.getProjection())
59  pt.project(ddproj,origproj)
60
61  if debug:
62      print '-----  Albers Coordinates  -----'
63      print 'x: %s y: %s' % (pt.x, pt.y)
64      print '--------------------------------'
65
66  # create an extent for our mapObj by buffering our projected
67  # point by the buffer distance.  Then set the mapObj's extent.
68  buffer = 600
69  extent = mapscript.rectObj()
70  extent.minx = pt.x - buffer
71  extent.miny = pt.y - buffer
72  extent.maxx = pt.x + buffer
73  extent.maxy = pt.y + buffer
74  amap.setExtent(extent.minx, extent.miny,
75                 extent.maxx, extent.maxy)
76
77  # set the output format to jpeg
78  outputformat = mapscript.outputFormatObj('GD/JPEG')
79  amap.setOutputFormat(outputformat)
80
81  # give the WMS client a place to put temp files
82  amap.web.imagepath = os.environ['TEMP']
83
84  # define the TerraServer WMS layer
85  layer = mapscript.layerObj(amap)
86  layer.connectiontype = mapscript.MS_WMS
87  layer.type = mapscript.MS_LAYER_RASTER
88  layer.metadata.set('wms_srs','EPSG:4326')
89  layer.metadata.set("wms_title", "USGS Digital Ortho-Quadrangles")
90  ts_url =
    "http://terraservice.net/ogcmap.ashx?VERSION=1.1.1&SERVICE=wms&LAYERS=DOQ&F
    ORMAT=jpeg&styles="
91  layer.connection = ts_url
92  layer.setProjection('init=epsg:4326')
93  layer.status = mapscript.MS_ON
94  if ms_debug:
95      layer.debug = mapscript.MS_ON
96
97  # import the libraries we'll need to make our pixelmap symbol
98  import urllib2
```


© Howard Butler

```
99  import cStringIO
100
101 # get a jpeg image from somewhere on the web and read it into
102 # a StringIO.
103 url = 'http://www.worldcommunitygrid.org/images/agent/house.jpg'
104 f = urllib2.urlopen(url).read()
105 f = cStringIO.StringIO(f)
106
107 # create the symbol using the image
108 symbol = mapscript.symbolObj('from_img')
109 symbol.type = mapscript.MS_SYMBOL_PIXMAP
110 img = mapscript.imageObj(f)
111 symbol.setImage(img)
112 symbol_index = amap.symbolset.appendSymbol(symbol)
113
114 # create a shapeObj out of our address point so we can
115 # add it to the map.
116 line = mapscript.lineObj()
117 line.add(pt)
118 shape=mapscript.shapeObj(mapscript.MS_SHAPE_POINT)
119 shape.add(line)
120 shape.setBounds()
121
122 # create our inline layer that holds our address point
123 inline_layer = mapscript.layerObj(amap)
124 inline_layer.addFeature(shape)
125 inline_layer.setProjection(amap.getProjection())
126 inline_layer.name = "housept"
127 inline_layer.type = mapscript.MS_LAYER_POINT
128 inline_layer.connectiontype=mapscript.MS_INLINE
129 inline_layer.status = mapscript.MS_ON
130 inline_layer.transparency = mapscript.MS_GD_ALPHA
131
132 # add the image symbol we defined above to the inline
133 # layer.
134 cls = mapscript.classObj(inline_layer)
135 cls.name='classname'
136 style = mapscript.styleObj(cls)
137 style.symbol = amap.symbolset.index('from_img')
138
139 # draw the map and save it somewhere.
140 img = amap.draw()
141 f = open(r'c:\temp\foo.jpg','wb')
142 f.write(img.getBytes())
143 f.close()
```


© Howard Butler