PyMapLib User Manual

GABBs Python Maps Library

Scientific Solutions Group

Rosen Center for Advanced Computing

Purdue University

2016-11-29

Content

1. What is PyMapLib (GABBs Python Maps Library)?

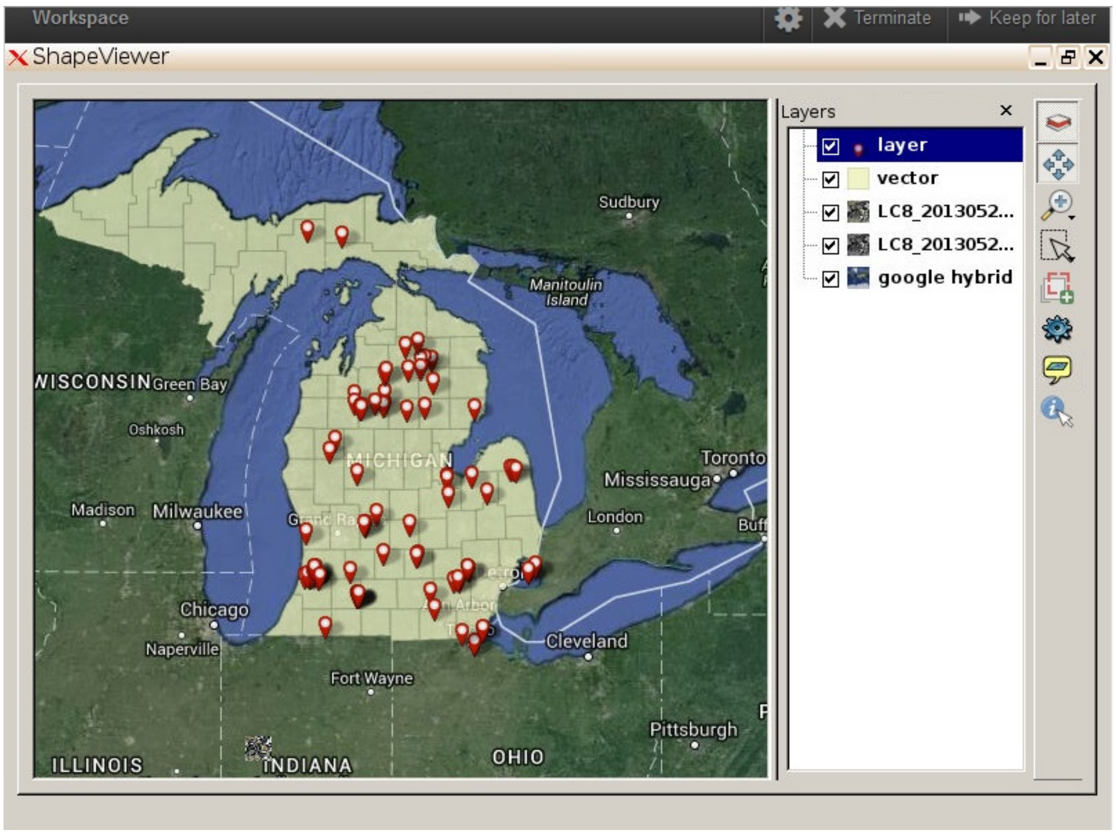
As a part of the NSF funded Geospatial Data Analysis Building Blocks (GABBs) project, PyMapLib is designed as a generic framework for geospatial data visualization to support scientific computation and data tools. Based on a diverse set of use cases, the primary goal of PyMapLib is to lower the barrier of developing map-enabled geospatial tools by non-expert scientific users, requiring only minimal programming effort. More specifically, the library (1) efficiently supports common geospatial data types and protocols, including vector, raster, delimited text, GeoJSON, spatial database sources, and OGC web services (TMS, WMS, WFS), (2) allows users to interact with the map by either selecting inputs or displaying outputs, (3) supports simple operations of geospatial data processing and analysis, and (4) is lightweight and requires minimal programming effort for application developers.

GABBs Python Maps Library allows users and scientists to display maps, images, vectors and other geospatial information within their Tools. It includes the GABBs Python Maps API and GABBs Map Widgets. GABBs Python Maps API enables developers to customize maps and the information on maps. GABBs MapWidgets provide a container for displaying maps without programming, which is an easy way to integrate maps into an application.

2. Basics

In this section we will show how to create a simple map using the GABBs Python Maps Library. We will start with a simple example and then go through the code step by step.

2.1. Create a simple map



This example creates a map that shows a vector of Michigan, U.S, and a raster image of Purdue, West Lafayette, Indiana, U.S.

Example code:

```
# Create a Map Container
self.mapContainer = gabbs.maps.MapContainer()
self.mapContainer.setPanControl(True)
self.mapContainer.setLayerControl(True)
self.mapContainer.setZoomControl(True, size = "CUSTOM", options = "ZOOMIN, ZOOMOUT")
self.mapContainer.setSelectControl(True, size = "CUSTOM", options = "SINGLE, RECTANGLE, POLYGON")
self.mapContainer.setPlugin("drawingtool")
self.mapContainer.setCaptureTool(True)
# Add canvas as a widget to the layout
self.layout.addWidget(self.mapContainer)
# Create a Map object
self.map = gabbs.maps.Map("mapName")
self.map.setMapCenter(-86, 39)
self.map.setMapZoom(7)
self.map.setMapScale(5, 10)
self.mapContainer.addLayer(self.map)
# Create a Vector object
self. polygon = gabbs.maps.Vector(os.path.join(".", "data", "Counties", "Counties.shp"), "Counties")
self. polygon.setCustomStyle("/home/mygeohub/shin152/mapTest/border.qml")
```

```
    self. polygon.setCustomScale([6,7])
    self.mapContainer.addLayer(self.polygon)
    # Create a Raster object
    self. raster = gabbs.maps.Raster(os.path.join("/home/mygeohub/wanwei ", "data", "LC8_20130524.tif"))
    self. raster.setCustomStyle("/home/mygeohub/shin152/mapTest/raster.qml")
    self. raster.setCustomScale([4,10])
    self.mapContainer.addLayer(self. raster)
    # Create a Point object
    markerProp = {"layerName":        "layer",
                  "delimiter":        ",",
                  "xField":           "Longitude",
                  "yField":           "Latitude",
                  "useSystemStyle":   True,
                  "styleName":        "SVG_MARKER_RED_MARKER" }
    self.point = gabbs.maps.DelimitedText("/home/mygeohub/wanwei/data/Burn_Point_Locations.csv",
                                          name = “points”, option = markerProp)
    self.mapContainer.addLayer(self.point)
```

The rest of this section describes the above example step by step.

2.2. Load the GABBs Maps API

The GABBs Maps API is a Python library. It can be added to a Python main program as follows:

```
import gabbs.maps
# Initialize gabbs maps library
gabbs.maps.gbsLoadLibrary()
#adding your codes here
…
# Unload gabbs maps library
gabbs.maps.gbsUnloadLibrary()
```

2.3. Create a map container

Create a *mapContainer* object to contain the map. The *mapContainer* is a widget in PyQT. Use layout to size the element.
Note: The map will always "inherit" its size from its container layout.
In the initialize step, create an object by calling *gabbs.maps.MapContainer().* The *MapContainer* has several methods to enable/disable control panels on the toolbar. The following lines show example code fragments that initialize the map container. The order of the control panels will be the same as the order of function calls.

```
    # Create a Map Container
    self.mapContainer = gabbs.maps.MapContainer()
    self.mapContainer.setMapTipControl(True)
    self.mapContainer.setPanControl(True)
    self.mapContainer.setLayerControl(True)
    self.mapContainer.setZoomControl(True, size = "CUSTOM", options = "ZOOMIN, ZOOMOUT, ZOOMHOME")
    self.mapContainer.setSelectControl(True, size = "CUSTOM", options = "SINGLE, RECTANGLE, RADIUS")
    self.mapContainer.setPlugin("drawingtool")
    self.mapContainer.setCaptureTool(True)
    self.mapContainer.setOverviewControl(True)
    self.mapContainer.setStatusControl(True)
    # Add canvas as a widget to the layout
    self.layout.addWidget(self.mapContainer)
```

- Zoom control
A user can add zoom controls in the control panel by setting *setZoomControl().* Available options are ZOOMIN, ZOOMOUT, ZOOMHOME, ZOOMFULL and ZOOMLAYER.

- Selection control
Like the zoom control, a user can set specific selection controls using *setSelectControl().* Available options are SINGLE, RECTANGLE, POLYGON, FREEHAND and RADIUS.

- Drawing tool
The drawing tool can be used to draw a rectangle on the map area to get four coordinates of the rectangle.
Note that this is different from the selection tool. The selection tool is used for selecting overlays on the map while the drawing tool grabs the four corners from the rectangle drawn on the map.
Drawing tool can be added to the control panel by specifying *setPlugin("drawingtool")* from the *mapContainer*.

- Capture tool
The capture tool creates an image of what is displayed in the map area. This feature can be added to the control panel by *setCaptureTool(True)* and the output file will be saved as ~/mapsTempOutput/img.png

2.4. Create a map object.

PyMapLib uses OSM(Open Street Map) as the base map. To load the base map into the map container, call *gabbs.maps.Map("mapName")*. The following lines show the initial steps to load OSM.

```
    # Create a Map object
    self.map = gabbs.maps.Map("mapName")
    self.map.setMapCenter(-86, 39)
    self.map.setMapZoom(7)
    self.map.setMapScale(5, 10)
    self.mapContainer.addLayer(self.map)
```

The map center will be set as the longitude and latitude of the parameters from *setMapCenter(lat, lon)* . Also, *setMapZomm(level)* specifies the initial zoom level for the map. A value of zero shows a map of the Earth fully zoomed out and higher zoom levels zoom in to the higher resolution. The method *setMapScale(min, max)* is to limit the minimum/maximum value of the zoom level. In this case, user can adjust the zoom level between 5 to 10.

3. PyMapLib Maps Overlays

Overlays are objects on the map that are bound to latitude/longitude coordinates. Scientists can use overlay to display result images. PyMapLib supports several types of overlays:

Vector:
A vector file can have features consisting of (1) polyline which includes a series of straight lines on a map, (2) polygon which includes a series of straight lines on a map and the shape is "closed", (3) circle, and (4) rectangle.

Raster:
This includes any GDAL supported image file format, such as tiff, geoTiff. The file may be single band or multiband.

Delimited Text:
This includes CSV, or other text files. This is often symbolized as markers on a map which can also display custom icon images.

In the future, GABBs Maps will support info windows, displaying content within a popup balloon on top of a map, and other custom overlays.

3.1. Vector

A vector polygon is similar to a polyline in that it consists of a series of coordinates in an ordered sequence. However, polygons are designed to define regions within a closed loop. Polygons are made of straight lines, and the shape is "closed" (all the lines connect up).

A polygon overlay supports the following properties:
strokeColor - specifies a hexadecimal color for the line (format: "#FFFFFF")
strokeOpacity - specifies the opacity of the line (a value between 0.0 and 1.0)
strokeWeight - specifies the weight of the line's stroke in pixels
fillColor - specifies a hexadecimal color for the area within the enclosed region (format: "#FFFFFF")

fillOpacity - specifies the opacity of the fill color (a value between 0 and 100)
visible- defines whether the layer is visible by users (true/false)

Here is an example:

```
    # Create a Vector object
    self. polygon = gabbs.maps.Vector(os.path.join(".", "data", "Counties", "Counties.shp"), "Counties")
    polygonProp = {"layerName":          "vector",
                   "strokeColor":    "#0000FF",
                   "strokeOpacity": 0.8,
                   "strokeWeight":    2,
                   "fillColor":        (40, 200, 160),
                   "fillOpacity":      0.5}
    self.polygon.setLayerProperty(polygonProp)
    self.mapContainer.addLayer(self.polygon)

    # Create a Vector object (with custom style)
    self. polygon = gabbs.maps.Vector(os.path.join(".", "data", "Counties", "Counties.shp"), "Counties")
    self. polygon.setCustomStyle("/home/mygeohub/shin152/mapTest/border.qml")
    self. polygon.setCustomScale([6,7])
    self.mapContainer.addLayer(self.polygon)
```

3.2. Raster

A raster overlay supports the following properties:

opacity - specifies the opacity of the fill color (a value between 0.0 and 1.0)
visible- defines whether the layer is visible by users (true/false)

Here is an example:

```
    # Create a Raster object
    self. raster = gabbs.maps.Raster(os.path.join("/home/mygeohub/wanwei ", "data", "LC8_20130524.tif"))
    self. raster.setCustomStyle("/home/mygeohub/shin152/mapTest/raster.qml")
    self. raster.setCustomScale([4,10])
    self.mapContainer.addLayer(self. raster)
```

3.3. Delimited Text

The Delimited Text overlay constructor creates a point layer. (Note that the position property must be set for the marker to display). Add the layer to the map by using the *addLayer()* method of *mapContainer* object.

Here is an example:

```
    # Create a Point object
    markerProp = {"layerName":          "layer",
                    "delimiter":           ",",
                    "xField":              "Longitude",
                    "yField":              "Latitude",
                    "useSystemStyle":   True,
                    "styleName":          "SVG_MARKER_RED_MARKER" }
    self.point = gabbs.maps.DelimitedText("/home/mygeohub/wanwei/data/Burn_Point_Locations.csv",
                                          name = "points", option = markerProp)
    self.mapContainer.addLayer(self.point)
```

3.4 Add / Remove Layer

The following methods allow you to add or remove a layer:

```
    self.mapContainer.addLayer(self.polygon)
    gabbs.maps.gbsRemoveLayer(self. polygon)
```

3.5 Customized layer

A layer's style can be customized using a QML file by specifying in setCustomStyle(pathToStyleFile). Here is a useful tutorial how to style vector data in QGIS: http://qgis.spatialthoughts.com/2012/02/tutorial-styling-vector-data-in-qgis.html

3.6 Zoom-controlled visibility

A method setCustomScale([min, max]) makes the map library to show a layer only in specified zoom levels. For example,

```
    self. raster.setCustomScale([4,10])
```

The raster layer with this custom scale will be visible only in the zoom level from 4 to 10.

4. PyMapLib Map Events and Actions

4.1. Add an action to the map

PyMapLib allows users to add an action that will execute a script function once a mouse click event is triggered when using map tools. It is also possible to link the action to a specific layer of the maps on demand. It provides an easy and safe way to add user defined functions to map events. In addition, user can get map attributes by using built-in key words in the action. Please see **4.2 GABBs Map Action Reference** for details.

The gbsAddAction function creates a GABBs Maps action. The action script to perform when an event is triggered is added as a parameter to the action.

Here is an example that shows a message box of clicked mouse coordinates when user clicks on the map. It defines a "python" type of action to call the QMessageBox function. In the call to gbsAddAction(), self.polygon is the target layer, "python" is the action type, "massage" is the action name, “action1” is a string that contains the script to be executed.

```
action1 = """
from PyQt4.QtCore import *
from PyQt4.QtGui import *
QMessageBox.information(None, "Info", "left button clicked x:[% $clickx %], y: [% $clicky %] ")
"""
gabbs.maps.gbsAddAction(self.polygon, "python", "massage", action1)
```

Below is another example that echoes an attribute value when a feature is clicked. It defines a “generic” type of action to call the system echo command:

```
action2 = """ echo "[% "fpa_name" %]" """
gabbs.maps.addAction(self.point, "generic", "name", action2)
```

4.2 GABBs Maps Action Reference

Action Type:

Generic“generic”

tr("Echo attribute's value"), "echo \"[% \"MY_FIELD\" %]\"", "", true)
tr("Run an application"), "ogr2ogr -f \"ESRI Shapefile\" \"[% \"OUTPUT_PATH\" %]\" \"[% \"INPUT_FILE\" %]\"", "", true);

Python“python”

Python, tr("Get feature id"), "QtGui.QMessageBox.information(None, \"Feature id\", \"feature id is [% $id %]\")", "", false);
Python, tr("Selected field's value (Identify features tool)"), "QtGui.QMessageBox.information(None, \"Current field's value\", \"[% $currentfield %]\")", "", false);
Python, tr("Clicked coordinates (Run feature actions tool)"), "QtGui.QMessageBox.information(None, \"Clicked coords\", \"layer: [% $layerid %]\\ncoords: ([% $clickx %],[% $clicky %])\")", "", false);

OpenUrl“open”

OpenUrl, tr("Open file"), "[% \"PATH\" %]", "", false);
OpenUrl, tr("Search on web based on attribute's value"), "http://www.google.com/search?q=[% \"ATTRIBUTE\" %]", "", false);

Action Keywords:

[%$clickx %],[% $clicky%]

$clickx, $clicky keywords will be automatically replaced by the mouse cursor position when a click event is triggered.

[% \"ATTRIBUTE\" %]

ATTRIBUTE keyword need to be the name of anattribute field, and the value of the attribute field of certain feature will be required by the action tool.

4.3. Add an Event Listener to the Map

Add an event listener that will execute the user function on certain signal from the map object. It is also possible to load the GABBs Maps API on demand.

This feature is currently under development.

5. PyMapLib Maps API Reference

PyMapLib Python Maps API gabbs.maps.			
No	API	Return Value	Usage
1.	gbsLoadLibrary()	Void	Initialize GABBs maps library
2.	gbsUnloadLibrary()	Void	Exit GABBs maps library
3.	MapContainer(dict canvasProp)	MapContainer	Create a Map Container, aka map canvas
4.	Map(dict mapProp)	Map	Create a Map layer
5.	Raster(dict rasterProp)	Raster	Create a Rasterlayer
6.	Vector(dict vectorProp)	Vector	Create a Vectorlayer
7.	DelimitedText(dict pointProp)	DelimitedText	Create a DelimitedText, CSV or other WKTlayer
8.	MapContainer.addLayer(Layer)	Void	Add layer to map canvas
9.	MapContainer.removeLayer(Layer)	Void	Remove layer to map canvas
10.	Layer.show()	Void	Show layer
11.	Layer.hide()	Void	Hide layer
12.	Layer.setProperty()	Void	Update layer's properties and styles
13.	gbsGetSelectedAttributes()	List[]	Get all attributes in selected features of current layer
14.	gbsGetSelectedBounds()	QRectF	Get the bounding box's coordinates of all selected features of current layer