ipyleaflet

Jul 29, 2020
Installation

1 Using pip ........................ 1
2 Using conda ...................... 3
3 JupyterLab extension .......... 5
4 Development installation ...... 7
5 Usage ............................. 9
6 Map .............................. 11
   6.1 Usage ........................ 11
   6.2 Save to HTML ............... 12
   6.3 Attributes ................. 13
   6.4 Methods .................... 14
7 Basemaps ....................... 15
8 Tile Layer ...................... 17
   8.1 Example .................... 17
   8.2 Usage ...................... 17
   8.3 Attributes ................ 18
9 Local Tile Layer ............... 19
   9.1 Example .................... 19
   9.2 Attributes ................ 19
10 Marker ......................... 21
   10.1 Example ................... 21
   10.2 Attributes ................ 22
   10.3 Methods ................... 22
11 Icon ............................ 23
   11.1 Example ................... 23
   11.2 Attributes ................ 23
12 AwesomeIcon .................. 25
   12.1 Example ................... 25
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>38.2</td>
<td>Attributes</td>
<td>86</td>
</tr>
<tr>
<td>39</td>
<td>Search Control</td>
<td>87</td>
</tr>
<tr>
<td>39.1</td>
<td>Example</td>
<td>87</td>
</tr>
<tr>
<td>39.2</td>
<td>Attributes</td>
<td>88</td>
</tr>
<tr>
<td>40</td>
<td>ipyleaflet related projects</td>
<td>89</td>
</tr>
</tbody>
</table>
CHAPTER 1

Using pip

```
pip install ipyleaflet
jupyter nbextension enable --py --sys-prefix ipyleaflet  # can be skipped for notebook 5.3 and above
```
CHAPTER 2

Using conda

conda install -c conda-forge ipyleaflet
If you have JupyterLab, you will also need to install the JupyterLab extension:

```
jupyter labextension install @jupyter-widgets/jupyterlab-manager jupyter-leaflet
```
Development installation

For a development installation (requires npm):

```bash
git clone https://github.com/jupyter-widgets/ipyleaflet.git
cd ipyleaflet
pip install -e .
jupyter nbextension install --py --symlink --sys-prefix ipyleaflet
jupyter nbextension enable --py --sys-prefix ipyleaflet
jupyter labextension install @jupyter-widgets/jupyterlab-manager js # If you are developing on JupyterLab
```

Note for developers:

- the `-e` pip option allows one to modify the Python code in-place. Restart the kernel in order to see the changes.
- the `--symlink` argument on Linux or OS X allows one to modify the JavaScript code in-place. This feature is not available with Windows.

For automatically building the JavaScript code every time there is a change, run the following command from the `ipyleaflet/js/` directory:

```
npm run watch
```

If you are on JupyterLab you also need to run the following in a separate terminal:

```
jupyter lab --watch
```

Every time a JavaScript build has terminated you need to refresh the Notebook page in order to load the JavaScript code again.
ipyleaflet is an interactive widgets library, it is based on ipywidgets. This means that everything in ipyleaflet (e.g. the Map, TileLayers, Markers...) is interactive: you can dynamically update attributes from Python or from the Notebook interface.

For example, you can create a Marker layer and interact with it:

```python
from ipyleaflet import Map, Marker

center = (52.204793, 360.121558)
m = Map(center=center, zoom=15)

marker = Marker(location=center, draggable=True)
m.add_layer(marker);

display(m)

# Now that the marker is on the Map, you can drag it with your mouse,
# it will automatically update the `marker.location` attribute in Python

# You can also update the marker location from Python, that will update the
# marker location on the Map:
marker.location = (50, 356)
```

ipywidgets is powered by traitlets, this brings an observer pattern implementation which allows you to react on widget attribute changes.

For example, you can define a Python callback that will be called whenever the marker location has changed:

```python
def on_location_changed(event):
    # Do some computation given the new marker location, accessible from `event['new`
    pass

marker.observe(on_location_changed, 'location')
```
Please check out the traitlets documentation for more details about the observer pattern implementation.

Note: Everything in ipyleaflet is an interactive widget, from the Map class to Layer and Control classes. This means that what we achieved here with marker.location, you can achieve it with map.zoom, layer.url, or heatmap.locations

You can try ipyleaflet online using binder, no need to install anything on your computer:
6.1 Usage

```python
from ipyleaflet import Map, basemaps, basemap_to_tiles

m = Map(
    basemap=basemap_to_tiles(basemaps.NASAGIBS.ModisTerraTrueColorCR, "2017-04-08"),
    center=(52.204793, 360.121558),
    zoom=4
)

m
```

You can find the list of available basemaps in the *Basemaps* page.

You can add multiple layers and controls to the map, using the add_layer/add_control methods. All those layers and controls are widgets themselves. So you can dynamically update their attributes from Python or by interacting with the map on the page (see Usage)

```python
from ipyleaflet import Map, Marker, basemaps, basemap_to_tiles

m = Map(
    basemap=basemap_to_tiles(basemaps.NASAGIBS.ModisTerraTrueColorCR, "2017-04-08"),
    center=(52.204793, 360.121558),
    zoom=4
)

m.add_layer(Marker(location=(52.204793, 360.121558)))

m
```
6.2 Save to HTML

You can save the Map and all its layers and controls to an HTML page using the `save` method:

```python
m.save('my_map.html', title='My Map')
```

**Note:** The saved file is a static HTML page, so there is no possible interaction with Python anymore. This means that all the Python callbacks you defined (e.g. on marker move) cannot be executed. If you want to serve the Map widget to an HTML page while keeping a Python kernel alive on the server, you might want to look at Voila.
# 6.3 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
<th>Doc</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>layers</td>
<td>(default_layer)</td>
<td>Tuple of layers</td>
<td></td>
</tr>
<tr>
<td>controls</td>
<td>()</td>
<td>Tuple of controls</td>
<td></td>
</tr>
<tr>
<td>center</td>
<td>(0.0, 0.0)</td>
<td>Initial geographic center of the map</td>
<td></td>
</tr>
<tr>
<td>zoom</td>
<td>12</td>
<td>Initial map zoom level</td>
<td></td>
</tr>
<tr>
<td>max_zoom</td>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>min_zoom</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>zoom_snap</td>
<td>1</td>
<td>Forces the map’s zoom level to always be a multiple of this</td>
<td></td>
</tr>
<tr>
<td>zoom_delta</td>
<td>1</td>
<td>Controls how much the map’s zoom level will change after pressing + or - on the keyboard, or using the zoom controls</td>
<td></td>
</tr>
<tr>
<td>crs</td>
<td>projections.EPSG3857</td>
<td>Coordinate reference system, which can be ‘Earth’, ‘EPSG3395’, ‘EPSG3857’, ‘EPSG4326’, ‘Base’, ‘Simple’ or you can define your own projection. (See CustomProjections notebook)</td>
<td></td>
</tr>
<tr>
<td>dragging</td>
<td>True</td>
<td>Whether the map be draggable with mouse/touch or not</td>
<td></td>
</tr>
<tr>
<td>touch_zoom</td>
<td>True</td>
<td>Whether the map can be zoomed by touch-dragging with two fingers on mobile</td>
<td></td>
</tr>
<tr>
<td>scroll_wheel_zoom</td>
<td>False</td>
<td>Whether the map can be zoomed by using the mouse wheel</td>
<td></td>
</tr>
<tr>
<td>double_click_zoom</td>
<td>True</td>
<td>Whether the map can be zoomed in by double clicking on it and zoomed out by double clicking while holding shift</td>
<td></td>
</tr>
<tr>
<td>box_zoom</td>
<td>True</td>
<td>Whether the map can be zoomed to a rectangular area specified by dragging the mouse while pressing the shift key</td>
<td></td>
</tr>
<tr>
<td>tap</td>
<td>True</td>
<td>Enables mobile hacks for supporting instant taps</td>
<td></td>
</tr>
<tr>
<td>tap_tolerance</td>
<td>15</td>
<td>The max number of pixels a user can shift his finger during touch for it to be considered a valid tap</td>
<td></td>
</tr>
<tr>
<td>world_copy_jump</td>
<td>True</td>
<td>With this option enabled, the map tracks when you pan to another “copy” of the world and seamlessly jumps to</td>
<td></td>
</tr>
<tr>
<td>close_popup_on_click</td>
<td>True</td>
<td>Set it to False if you don’t want popups to close when user clicks the map</td>
<td></td>
</tr>
<tr>
<td>bounce_at_zoom_limits</td>
<td>True</td>
<td>Set it to False if you don’t want the map to zoom beyond min/max zoom and then bounce back when pinch-zooming</td>
<td></td>
</tr>
<tr>
<td>keyboard</td>
<td>True</td>
<td>Makes the map focusable and allows users to navigate the map with keyboard arrows and +/- keys</td>
<td></td>
</tr>
<tr>
<td>keyboard_pan_offset</td>
<td>80</td>
<td></td>
<td></td>
</tr>
<tr>
<td>keyboard_zoom_offset</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>inertia</td>
<td>True</td>
<td>If enabled, panning of the map will have an inertia effect</td>
<td></td>
</tr>
<tr>
<td>inertia_deceleration</td>
<td>3000</td>
<td>The rate with which the inertial movement slows down, in pixels/second²</td>
<td></td>
</tr>
<tr>
<td>inertia_max_speed</td>
<td>1500</td>
<td>Max speed of the inertial movement, in pixels/second</td>
<td></td>
</tr>
<tr>
<td>zoom_control</td>
<td>True</td>
<td></td>
<td></td>
</tr>
<tr>
<td>attribution_control</td>
<td>True</td>
<td></td>
<td></td>
</tr>
<tr>
<td>zoom_animation_threshold</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## 6.4 Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Arguments</th>
<th>Doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_layer</td>
<td>Layer instance</td>
<td>Add a new layer to the map</td>
</tr>
<tr>
<td>remove_layer</td>
<td>Layer instance</td>
<td>Remove a layer from the map</td>
</tr>
<tr>
<td>substitute_layer</td>
<td>Layer instance</td>
<td>Substitute a layer with a new layer</td>
</tr>
<tr>
<td>clear_layers</td>
<td></td>
<td>Remove all layers from the map</td>
</tr>
<tr>
<td>add_control</td>
<td>Control instance</td>
<td>Add a new control to the map</td>
</tr>
<tr>
<td>remove_control</td>
<td>Control instance</td>
<td>Remove a control from the map</td>
</tr>
<tr>
<td>clear_controls</td>
<td></td>
<td>Remove all controls from the map</td>
</tr>
<tr>
<td>on_interaction</td>
<td>callable object</td>
<td>Add a callback on interaction</td>
</tr>
<tr>
<td>save</td>
<td>output file</td>
<td>Save the map to an HTML file</td>
</tr>
</tbody>
</table>
You can find on this page the default basemaps available in ipyleaflet, of course you can use another provider creating your own TileLayer layer.

**Note:** If one map on this page is completely grey, please report it by opening an issue: https://github.com/jupyter-widgets/ipyleaflet/issues/new

```python
from ipyleaflet import Map, basemaps

center = [38.128, 2.588]
zoom = 5

Map(basemap=basemaps.OpenStreetMap.Mapnik, center=center, zoom=zoom)
Map(basemap=basemaps.OpenStreetMap.BlackAndWhite, center=center, zoom=zoom)
Map(basemap=basemaps.OpenStreetMap.France, center=center, zoom=zoom)
Map(basemap=basemaps.OpenStreetMap.HOT, center=center, zoom=zoom)
Map(basemap=basemaps.OpenTopoMap, center=center, zoom=zoom)
Map(basemap=basemaps.Hydda.Full, center=center, zoom=zoom)
Map(basemap=basemaps.Hydda.Base, center=center, zoom=zoom)
Map(basemap=basemaps.Esri.WorldStreetMap, center=center, zoom=zoom)
Map(basemap=basemaps.Esri.DeLorme, center=center, zoom=zoom)
```
```python
Map(basemap=basemaps.Esri.WorldTopoMap, center=center, zoom=zoom)
Map(basemap=basemaps.Esri.WorldImagery, center=center, zoom=zoom)
Map(basemap=basemaps.Esri.NatGeoWorldMap, center=center, zoom=zoom)
Map(basemap=basemaps.HikeBike.HikeBike, center=center, zoom=zoom)
Map(basemap=basemaps.MtbMap, center=center, zoom=zoom)
Map(basemap=basemaps.CartoDBPOSITRON, center=center, zoom=zoom)
Map(basemap=basemaps.CartoDB.DarkMatter, center=center, zoom=zoom)
Map(basemap=basemaps.NASAGIBS.ModisTerraTrueColorCR, center=center, zoom=zoom)
Map(basemap=basemaps.NASAGIBS.ModisTerraBands367CR, center=center, zoom=zoom)
Map(basemap=basemaps.NASAGIBS.ModisTerraBands721CR, center=center, zoom=zoom)
Map(basemap=basemaps.NASAGIBS.ModisAquaTrueColorCR, center=center, zoom=zoom)
Map(basemap=basemaps.NASAGIBS.ModisAquaBands721CR, center=center, zoom=zoom)
Map(basemap=basemaps.NASAGIBS.ViirsTrueColorCR, center=center, zoom=zoom)
Map(basemap=basemaps.NASAGIBS.ViirsEarthAtNight2012, center=center, zoom=zoom)
Map(basemap=basemaps.Strava.All, center=center, zoom=zoom)
Map(basemap=basemaps.Strava.Ride, center=center, zoom=zoom)
Map(basemap=basemaps.Strava.Run, center=center, zoom=zoom)
Map(basemap=basemaps.Strava.Water, center=center, zoom=zoom)
Map(basemap=basemaps.Strava.Winter, center=center, zoom=zoom)
Map(basemap=basemaps.Stamen.Terrain, center=center, zoom=zoom)
Map(basemap=basemaps.Stamen.Toner, center=center, zoom=zoom)
Map(basemap=basemaps.Stamen.Watercolor, center=center, zoom=zoom)
```
8.1 Example

```python
from ipyleaflet import Map, basemaps, basemap_to_tiles
m = Map(center=(52.204793, 360.121558), zoom=9)
dark_matter_layer = basemap_to_tiles(basemaps.CartoDB.DarkMatter)
m.add_layer(dark_matter_layer)
m
```

8.2 Usage

Creating a TileLayer is straightforward, a dictionary containing basic tile layers is provided. This dictionary is named basemaps.

A TileLayer instance can be created using the basemap_to_tiles function, specifying the wanted map (e.g. basemaps.CartoDB.DarkMatter, basemaps.Strava.Winter, basemaps.NASAGIBS.ModisTerraTrueColorCR,...).

Sometimes one could want to specify the date of the given images, for instance with NASA images:

```python
nasa_layer = basemap_to_tiles(basemaps.NASAGIBS.ModisTerraTrueColorCR, "2018-04-08");
m.add_layer(nasa_layer);
```
### 8.3 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>“https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png”</td>
</tr>
<tr>
<td>min_zoom</td>
<td>0</td>
</tr>
<tr>
<td>max_zoom</td>
<td>18</td>
</tr>
<tr>
<td>min_native_zoom</td>
<td>0</td>
</tr>
<tr>
<td>max_native_zoom</td>
<td>18</td>
</tr>
<tr>
<td>tile_size</td>
<td>256</td>
</tr>
<tr>
<td>attribution</td>
<td>“Map data (c) &lt;a href='https://openstreetmap.org'&gt;OpenStreetMap&lt;/a&gt; contributors”</td>
</tr>
<tr>
<td>detect_retina</td>
<td>False</td>
</tr>
<tr>
<td>opacity</td>
<td>1.0</td>
</tr>
<tr>
<td>visible</td>
<td>True</td>
</tr>
<tr>
<td>no_wrap</td>
<td>False</td>
</tr>
<tr>
<td>show_loading</td>
<td>False</td>
</tr>
<tr>
<td>loading</td>
<td>False (dynamically updated)</td>
</tr>
</tbody>
</table>
9.1 Example

```python
from ipyleaflet import Map, LocalTileLayer

m = Map(center=(52.204793, 360.121558), zoom=9)
m.add_layer(LocalTileLayer(path='tiles/{z}/{x}/{y}.png'))
m
```

Note that the behavior is different in Jupyter Notebook and in JupyterLab.

In the classic Jupyter Notebook, the path is relative to the Notebook you are working on.

In JupyterLab, the path is relative to the server (where you started JupyterLab) and you need to prefix the path with "files/".

9.2 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
<th>Doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>&quot;&quot;</td>
<td>Relative URL (e.g. ‘tiles/{z}/{x}/{y}.png’ or ‘files/tiles/{z}/{x}/{y}.png’ in JupyterLab)</td>
</tr>
</tbody>
</table>
10.1 Example

```python
from ipyleaflet import Map, Marker

center = (52.204793, 360.121558)

m = Map(center=center, zoom=15)

marker = Marker(location=center, draggable=False)
m.add_layer(marker);

m
```
# 10.2 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
<th>Doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>(0.0, 0.0)</td>
<td></td>
</tr>
<tr>
<td>z_index_offset</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>draggable</td>
<td>True</td>
<td>Whether the marker is draggable with mouse/touch or not</td>
</tr>
<tr>
<td>keyboard</td>
<td>True</td>
<td>Whether the marker can be tabbed to with a keyboard and clicked by pressing enter</td>
</tr>
<tr>
<td>title</td>
<td>&quot;&quot;</td>
<td>Text for the browser tooltip that appear on marker hover (no tooltip by default)</td>
</tr>
<tr>
<td>alt</td>
<td>&quot;&quot;</td>
<td>Text for the alt attribute of the icon image (useful for accessibility)</td>
</tr>
<tr>
<td>rise_on_hover</td>
<td>False</td>
<td>The z-index offset used for the rise_on_hover feature</td>
</tr>
<tr>
<td>opacity</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>visible</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>rise_offset</td>
<td>250</td>
<td>The z-index offset used for the rise_on_hover feature</td>
</tr>
<tr>
<td>rotation_angle</td>
<td>0</td>
<td>The rotation angle of the marker in degrees</td>
</tr>
<tr>
<td>rotation_origin</td>
<td>'bottom center'</td>
<td>The rotation origin of the marker</td>
</tr>
<tr>
<td>icon</td>
<td>None</td>
<td>The icon for the marker</td>
</tr>
</tbody>
</table>

# 10.3 Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Arguments</th>
<th>Doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>on_move</td>
<td>Callable object</td>
<td>Adds a callback on move event</td>
</tr>
</tbody>
</table>
11.1 Example

```python
from ipyleaflet import Marker, Icon, Map

center = (52.204793, 360.121558)

m = Map(center=center, zoom=10)
icon = Icon(icon_url='https://leafletjs.com/examples/custom-icons/leaf-green.png',
           icon_size=[38, 95], icon_anchor=[22, 94])
mark = Marker(location=center, icon=icon, rotation_angle=90, rotation_origin='22px 94px')
m.add_layer(mark);
m
```

11.2 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
<th>Doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>icon_url</td>
<td>''</td>
<td>url for icon</td>
</tr>
<tr>
<td>shadow_url</td>
<td>None</td>
<td>url for icon shadow</td>
</tr>
<tr>
<td>icon_size</td>
<td>(10, 10)</td>
<td>size icon will be rendered</td>
</tr>
<tr>
<td>shadow_size</td>
<td>(10, 10)</td>
<td>size icon shadow will be rendered</td>
</tr>
<tr>
<td>icon_anchor</td>
<td>(0, 0)</td>
<td>anchor point of icon</td>
</tr>
<tr>
<td>shadow_anchor</td>
<td>(0, 0)</td>
<td>anchor point of shadow</td>
</tr>
<tr>
<td>popup_anchor</td>
<td>(0, 0)</td>
<td>anchor point of popup</td>
</tr>
</tbody>
</table>
Font-Awesome icons for markers, see https://fontawesome.com/v4.7.0/icons for available icons.

12.1 Example

```python
from ipyleaflet import AwesomeIcon, Marker, Map

center = (38.91342738235981, -77.03912909142674)

icon1 = AwesomeIcon(
    name='bus',
    marker_color='red',
    icon_color='black',
    spin=False
)

marker1 = Marker(icon=icon1, location=(center[0], center[1] - 0.05))

icon2 = AwesomeIcon(
    name='gear',
    marker_color='green',
    icon_color='darkgreen',
    spin=True
)

marker2 = Marker(icon=icon2, location=(center[0], center[1] + 0.05))

m = Map(center=center, zoom=13)

m.add_layer(marker1)
m.add_layer(marker2)
m
```
12.2 Interactions

Unlike other widgets in ipyleaflet, the `AwesomeIcon` widget is not dynamic. If you want to dynamically update the marker icon, you need to reassign the `Marker.icon` property with a new icon.

```python
marker1.icon = AwesomeIcon(
    name='home',
    marker_color='blue',
    icon_color='black'
)
```

12.3 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
<th>Doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>'home'</td>
<td>Name of the Font-Awesome icon</td>
</tr>
<tr>
<td>marker_color</td>
<td>'blue'</td>
<td>Marker background color</td>
</tr>
<tr>
<td>icon_color</td>
<td>'white'</td>
<td>Icon color</td>
</tr>
<tr>
<td>spin</td>
<td>False</td>
<td>Whether the icon is spinning or not</td>
</tr>
</tbody>
</table>
13.1 Example

```python
from ipywidgets import HTML
from ipyleaflet import Map, Marker, Popup

center = (52.204793, 360.121558)

m = Map(center=center, zoom=9, close_popup_on_click=False)

marker = Marker(location=(52.1, 359.9))
m.add_layer(marker)

message1 = HTML()
message2 = HTML()
message1.value = "Try clicking the marker!"
message2.value = "Hello <b>World</b>"
message2.placeholder = "Some HTML"
message2.description = "Some HTML"

# Popup with a given location on the map:
popup = Popup(
    location=center,
    child=message1,
    close_button=False,
    auto_close=False,
    close_on_escape_key=False
)
m.add_layer(popup)

# Popup associated to a layer
marker.popup = message2
```

(continues on next page)
13.2 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
<th>Doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>(0.0, 0.0)</td>
<td>Content of the popup</td>
</tr>
<tr>
<td>child</td>
<td></td>
<td>Max width of the popup, in pixels</td>
</tr>
<tr>
<td>max_width</td>
<td>300</td>
<td>Min width of the popup, in pixels</td>
</tr>
<tr>
<td>min_width</td>
<td>50</td>
<td>If set, creates a scrollable container of the given height inside a popup if its content exceeds it</td>
</tr>
<tr>
<td>max_height</td>
<td></td>
<td></td>
</tr>
<tr>
<td>auto_pan</td>
<td>True</td>
<td>Set it to False if you don’t want the map to do panning animation to fit the opened popup</td>
</tr>
<tr>
<td>auto_pan_padding</td>
<td>(5, 5)</td>
<td></td>
</tr>
<tr>
<td>keep_in_view</td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>close_button</td>
<td>True</td>
<td>Controls the presence of a close button in the popup</td>
</tr>
<tr>
<td>close_on_escape_key</td>
<td>True</td>
<td>Set it to False if you want to override the default behavior of the ESC key for closing of the popup</td>
</tr>
<tr>
<td>class_name</td>
<td>&quot;&quot;</td>
<td>A custom CSS class name to assign to the popup</td>
</tr>
</tbody>
</table>
WMS Layer

14.1 Example

```python
from ipyleaflet import Map, WMSLayer, basemaps

wms = WMSLayer(
    url='http://mesonet.agron.iastate.edu/cgi-bin/wms/nexrad/n0r.cgi',
    layers='nexrad-n0r-900913',
    format='image/png',
    transparent=True,
    attribution='Weather data © 2012 IEM Nexrad'
)

m = Map(basemap=basemaps.CartoDB.Positron, center=(38.491, -95.712), zoom=4)

m.add_layer(wms)

m
```

14.2 Advanced usage

By default, options like `layers`, `format`, `transparent` are passed in the request URL. If your tiles provider needs any extra parameter, you can define your own `WMSLayer` class which adds new parameters. For example, the following code adds a `time` parameter to the request by defining a custom `TimeWMSLayer`:

```python
from traitlets import Unicode
class TimeWMSLayer(WMSLayer):
    time = Unicode('').tag(sync=True, o=True)
```

(continues on next page)
Because it is a widget, you can dynamically update WMS parameters from Python manually:

```
# This will redraw the layer dynamically
time_wms.time = '2005-08-29T14:00'
```

Or from another widget like a slider: (Note that this example will not work in the documentation as there is no live Python kernel, but it will work in a Jupyter Notebook)

```python
from ipywidgets import SelectionSlider

time_options = ['13:00', '13:30', '14:00', '14:30', '15:00', '15:30', '16:00', '16:30']

slider = SelectionSlider(description='Time:', options=time_options)

def update_wms(change):
    time_wms.time = '2005-08-29T{}'.format(slider.value)

slider.observe(update_wms, 'value')
```
## 14.3 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
<th>Doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>“https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png”</td>
<td></td>
</tr>
<tr>
<td>min_zoom</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>max_zoom</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>tile_size</td>
<td>256</td>
<td></td>
</tr>
<tr>
<td>attribution</td>
<td>“Map data (c) &lt;a href=’<a href="https://openstreetmap.org%E2%80%99%3EOpenStreetMap">https://openstreetmap.org’&gt;OpenStreetMap</a>&lt;/a&gt;”</td>
<td></td>
</tr>
<tr>
<td>detect_retina</td>
<td>False</td>
<td></td>
</tr>
<tr>
<td>opacity</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>visible</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>layers</td>
<td>“”</td>
<td>Comma-separated list of WMS layers to show</td>
</tr>
<tr>
<td>styles</td>
<td>“”</td>
<td>Comma-separated list of WMS styles</td>
</tr>
<tr>
<td>format</td>
<td>“image/jpeg”</td>
<td>WMS image format (use ‘image/png’ for layers with transparency)</td>
</tr>
<tr>
<td>transparent</td>
<td>False</td>
<td>If True, the WMS service will return images with transparency</td>
</tr>
<tr>
<td>crs</td>
<td>ipyleaflet.projections.EPSG3857</td>
<td>Projection used for this service.</td>
</tr>
</tbody>
</table>
CHAPTER 15

Image overlay and Video overlay

15.1 Example ImageOverlay

```python
from ipyleaflet import Map, ImageOverlay

m = Map(center=(25, -115), zoom=4)

image = ImageOverlay(
    url="https://i.imgur.com/06Q1fSz.png",
    # url='../06Q1fSz.png",
    bounds=((13, -130), (32, -100))
)

m.add_layer(image);

m
```

15.2 Example VideoOverlay

```python
from ipyleaflet import Map, VideoOverlay

m = Map(center=(25, -115), zoom=4)

video = VideoOverlay(
    url="https://www.mapbox.com/bites/00188/patricia_nasa.webm",
    bounds=((13, -130), (32, -100))
)

m.add_layer(video);

m
```
## 15.3 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
<th>Doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>&quot;&quot;</td>
<td>An http url to the footage or a relative path to a local file (image/video). Note that absolute local paths are not supported.</td>
</tr>
<tr>
<td>bounds</td>
<td>((0.0, 0.0), (0.0, 0.0))</td>
<td>SW and NE corners of the image</td>
</tr>
</tbody>
</table>
16.1 Example

```python
from ipyleaflet import Map, AntPath

m = Map(center=(51.332, 6.853), zoom=10)

ant_path = AntPath(
    locations=[
        [51.185, 6.773], [51.182, 6.752], [51.185, 6.733], [51.194, 6.729],
        [51.205, 6.732], [51.219, 6.723], [51.224, 6.723], [51.227, 6.728],
        [51.228, 6.734], [51.226, 6.742], [51.221, 6.752], [51.221, 6.758],
        [51.224, 6.765], [51.230, 6.768], [51.239, 6.765], [51.246, 6.758],
        [51.252, 6.745], [51.257, 6.724], [51.262, 6.711], [51.271, 6.701],
        [51.276, 6.702], [51.283, 6.710], [51.297, 6.725], [51.304, 6.732],
        [51.312, 6.735], [51.320, 6.734], [51.326, 6.726], [51.334, 6.713],
        [51.340, 6.696], [51.344, 6.678], [51.349, 6.662], [51.354, 6.655],
        [51.360, 6.655], [51.366, 6.662], [51.369, 6.675], [51.373, 6.704],
        [51.376, 6.715], [51.385, 6.732], [51.394, 6.741], [51.402, 6.743],
        [51.411, 6.742], [51.420, 6.733], [51.429, 6.718], [51.439, 6.711],
        [51.448, 6.716], [51.456, 6.724], [51.466, 6.719], [51.469, 6.713],
        [51.470, 6.701], [51.473, 6.686], [51.479, 6.680], [51.484, 6.680],
        [51.489, 6.685], [51.493, 6.700], [51.497, 6.714]
    ],
    dash_array=[1, 10],
    delay=1000,
    color='#7590ba',
    pulse_color='#3f6fba'
)

m.add_layer(ant_path)

m
```
16.2 Interactions

Like most widgets in ipyleaflet, the AntPath can be dynamically updated from Python.

```python
# Update the color
ant_path.color = 'red'

# Update the path
ant_path.locations = [[51.185, 6.773], [51.326, 6.726], [51.497, 6.714]]
```

16.3 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
<th>Doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>locations</td>
<td>[]</td>
<td>List of path points as (lat, lng) couples</td>
</tr>
<tr>
<td>color</td>
<td>&quot;#0000FF&quot;</td>
<td>Background color for the path</td>
</tr>
<tr>
<td>pulse_color</td>
<td>&quot;#FFFFFF&quot;</td>
<td>Color of the moving ants on the path</td>
</tr>
<tr>
<td>paused</td>
<td>True</td>
<td>Whether the ants are moving or not</td>
</tr>
<tr>
<td>reverse</td>
<td>False</td>
<td>Whether the ants are moving in reverse or not</td>
</tr>
<tr>
<td>use</td>
<td>'polyline'</td>
<td>Which shape is drawn, possible values are 'polyline', 'polygon', 'rectangle' and 'circle'</td>
</tr>
<tr>
<td>dash_array</td>
<td>[10, 20]</td>
<td>Dash pattern for lines as a list of non-negative numbers</td>
</tr>
<tr>
<td>weight</td>
<td>5</td>
<td>Lines weight</td>
</tr>
<tr>
<td>delay</td>
<td>400</td>
<td>Ants speed</td>
</tr>
<tr>
<td>radius</td>
<td>10</td>
<td>Radius of the circle, if use is set to 'circle'</td>
</tr>
</tbody>
</table>
17.1 Example Polyline

```python
from ipyleaflet import Map, Polyline

line = Polyline(
    locations=[
        [45.51, -122.68],
        [37.77, -122.43],
        [34.04, -118.22]
    ],
    color="green",
    fill=False
)

m = Map(center = (42.5, -41), zoom =2)
m.add_layer(line)
m
```

17.2 Example MultiPolyline

```python
from ipyleaflet import Map, Polyline

lines = Polyline(
    locations=[
        [[45.51, -122.68],
        [37.77, -122.43],
        [34.04, -118.2]],
        [[40.78, -73.91],
        [41.83, -87.62],
        [32.76, -96.72]]
    ]
)
```

(continues on next page)
m = Map(center = (42.5, -41), zoom = 2)
m.add_layer(lines)
m

### 17.3 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
<th>Doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>locations</td>
<td>[[]]</td>
<td>List of list of points of the polygon</td>
</tr>
<tr>
<td>stroke</td>
<td>True</td>
<td>Set it to <code>False</code> to disable borders</td>
</tr>
<tr>
<td>color</td>
<td>&quot;#0033FF&quot;</td>
<td>Stroke color</td>
</tr>
<tr>
<td>opacity</td>
<td>1.0</td>
<td>Stroke opacity</td>
</tr>
<tr>
<td>weight</td>
<td>5</td>
<td>Stroke width in pixels</td>
</tr>
<tr>
<td>fill</td>
<td>True</td>
<td>Whether to fill the polyline or not</td>
</tr>
<tr>
<td>fill_color</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>fill_opacity</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>dash_array</td>
<td></td>
<td></td>
</tr>
<tr>
<td>line_cap</td>
<td>&quot;round&quot;</td>
<td></td>
</tr>
<tr>
<td>line_join</td>
<td>&quot;round&quot;</td>
<td></td>
</tr>
</tbody>
</table>
18.1 Polygon

You can easily create a Polygon providing the list of vertex locations (in lat/lng).

```python
from ipyleaflet import Map, Polygon

polygon = Polygon(
    locations=[(42, -49), (43, -49), (43, -48)],
    color="green",
    fill_color="green"
)

m = Map(center=(42.5531, -48.6914), zoom=6)
m.add_layer(polygon);

m
```

Because the Polygon an interactive widget, you can dynamically update the locations/color from Python, and you will see updated on the Map.

18.2 Polygon with holes

You can define holes in your Polygon by using nested lists of vertex locations.

```python
from ipyleaflet import Map, Polygon

polygon = Polygon(
    locations=[
        [(37, -109.05), (41, -109.03), (41, -102.05), (37, -102.04)],
        [(37.29, -108.58), (40.71, -108.58), (40.71, -102.50), (37.29, -102.50)]
    ],
```
18.3 MultiPolygon

```python
from ipyleaflet import Map, Polygon

multipolygon = Polygon(
    locations=
    [
        [(42, -49), (43, -49), (43, -48)],
        [(44,-49), (43, -50), (44,-50)]
    ],
    color="green",
    fill_color="green"
)

m = Map(center=(42.5531, -48.6914), zoom=6)
m.add_layer(multipolygon);
m
```

18.4 Editable Polygon

If `transform` is set to `True`, you can dynamically edit the polygon with the mouse.

```python
from ipyleaflet import Map, Polygon

polygon = Polygon(
    locations=[[42, -49], (43, -49), (43, -48)],
    color="green",
    fill_color="green",
    transform=True
)

m = Map(center=(42.5531, -48.6914), zoom=6)
m.add_layer(polygon);
m
```
## 18.5 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
<th>Doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>locations</td>
<td>[]</td>
<td>List of points of the polygon</td>
</tr>
<tr>
<td>stroke</td>
<td>True</td>
<td>Set it to False to disable borders</td>
</tr>
<tr>
<td>color</td>
<td>“#0033FF”</td>
<td>Stroke color</td>
</tr>
<tr>
<td>opacity</td>
<td>1.0</td>
<td>Stroke opacity</td>
</tr>
<tr>
<td>weight</td>
<td>5</td>
<td>Stroke width in pixels</td>
</tr>
<tr>
<td>fill</td>
<td>True</td>
<td>Whether to fill the polygon or not</td>
</tr>
<tr>
<td>fill_color</td>
<td>None</td>
<td>If None, it will be the same as the color value</td>
</tr>
<tr>
<td>fill_opacity</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>dash_array</td>
<td></td>
<td></td>
</tr>
<tr>
<td>line_cap</td>
<td>“round”</td>
<td></td>
</tr>
<tr>
<td>line_join</td>
<td>“round”</td>
<td></td>
</tr>
<tr>
<td>transform</td>
<td>False</td>
<td>Whether the polygon is editable with the mouse or not</td>
</tr>
<tr>
<td>scaling</td>
<td>True</td>
<td>Whether the polygon scale is editable or not, needs transform set to True</td>
</tr>
<tr>
<td>rotation</td>
<td>True</td>
<td>Whether the polygon rotation is editable or not, needs transform set to True</td>
</tr>
<tr>
<td>uniform_scaling</td>
<td>False</td>
<td>Whether to keep the scale ratio when editing the scale, needs transform set to True</td>
</tr>
</tbody>
</table>
19.1 Example

```python
from ipyleaflet import Map, basemaps, basemap_to_tiles, Rectangle

watercolor = basemap_to_tiles(basemaps.Stamen.Watercolor)

m = Map(layers=(watercolor, ), center=(53, 354), zoom=5)

rectangle = Rectangle(bounds=((52, 354), (53, 360)))

m.add_layer(rectangle)

m
```

19.2 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
<th>Doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>bounds</td>
<td>()</td>
<td>SW and NE corners of the rectangle</td>
</tr>
<tr>
<td>stroke</td>
<td>True</td>
<td>Set it to False to disable borders</td>
</tr>
<tr>
<td>color</td>
<td>&quot;#0033FF&quot;</td>
<td>Stroke color</td>
</tr>
<tr>
<td>opacity</td>
<td>1.0</td>
<td>Stroke opacity</td>
</tr>
<tr>
<td>weight</td>
<td>5</td>
<td>Stroke width in pixels</td>
</tr>
<tr>
<td>fill</td>
<td>True</td>
<td>Whether to fill the polygon or not</td>
</tr>
<tr>
<td>fill_color</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>fill_opacity</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>dash_array</td>
<td></td>
<td></td>
</tr>
<tr>
<td>line_cap</td>
<td>&quot;round&quot;</td>
<td></td>
</tr>
<tr>
<td>line_join</td>
<td>&quot;round&quot;</td>
<td></td>
</tr>
</tbody>
</table>
20.1 Example

```python
from ipyleaflet import Map, basemaps, basemap_to_tiles, Circle

watercolor = basemap_to_tiles(basemaps.Stamen.Watercolor)

m = Map(layers=(watercolor, ), center=(53, 354), zoom=5)

circle = Circle()
circle.location = (50, 354)
circle.radius = 50000
circle.color = "green"
circle.fill_color = "green"

m.add_layer(circle)

m
```
## 20.2 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
<th>Doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>(0.0, 0.0)</td>
<td>Circle location</td>
</tr>
<tr>
<td>radius</td>
<td>10</td>
<td>Circle radius in meters</td>
</tr>
<tr>
<td>stroke</td>
<td>True</td>
<td>Set it to <code>false</code> to disable borders</td>
</tr>
<tr>
<td>color</td>
<td>&quot;#0033FF&quot;</td>
<td>Stroke color</td>
</tr>
<tr>
<td>opacity</td>
<td>1.0</td>
<td>Stroke opacity</td>
</tr>
<tr>
<td>weight</td>
<td>5</td>
<td>Stroke width in pixels</td>
</tr>
<tr>
<td>fill</td>
<td>True</td>
<td>Whether to fill the circle or not</td>
</tr>
<tr>
<td>fill_color</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>fill_opacity</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>dash_array</td>
<td></td>
<td></td>
</tr>
<tr>
<td>line_cap</td>
<td>&quot;round&quot;</td>
<td></td>
</tr>
<tr>
<td>line_join</td>
<td>&quot;round&quot;</td>
<td></td>
</tr>
</tbody>
</table>
21.1 Example

```python
from ipyleaflet import Map, basemaps, basemap_to_tiles, CircleMarker

watercolor = basemap_to_tiles(basemaps.Stamen.Watercolor)

m = Map(layers=(watercolor,), center=(53, 354), zoom=5)

circle_marker = CircleMarker()
circle_marker.location = (55, 360)
circle_marker.radius = 50
circle_marker.color = "red"
circle_marker.fill_color = "red"

m.add_layer(circle_marker)

m
```
## 21.2 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
<th>Doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>(0.0, 0.0)</td>
<td>Circle location</td>
</tr>
<tr>
<td>radius</td>
<td>10</td>
<td>Circle radius in pixels</td>
</tr>
<tr>
<td>stroke</td>
<td>True</td>
<td>Set it to <code>false</code> to disable borders</td>
</tr>
<tr>
<td>color</td>
<td>&quot;#0033FF&quot;</td>
<td>Stroke color</td>
</tr>
<tr>
<td>opacity</td>
<td>1.0</td>
<td>Stroke opacity</td>
</tr>
<tr>
<td>weight</td>
<td>5</td>
<td>Stroke width in pixels</td>
</tr>
<tr>
<td>fill</td>
<td>True</td>
<td>Whether to fill the circle or not</td>
</tr>
<tr>
<td>fill_color</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>fill_opacity</td>
<td>0.2</td>
<td></td>
</tr>
<tr>
<td>dash_array</td>
<td></td>
<td></td>
</tr>
<tr>
<td>line_cap</td>
<td>&quot;round&quot;</td>
<td></td>
</tr>
<tr>
<td>line_join</td>
<td>&quot;round&quot;</td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 22

Marker Cluster

22.1 Example

```python
from ipyleaflet import Map, Marker, MarkerCluster

m = Map(center=(50, 0), zoom=5)

marker1 = Marker(location=(48, -2))
marker2 = Marker(location=(50, 0))
marker3 = Marker(location=(52, 2))

marker_cluster = MarkerCluster(
    markers=(marker1, marker2, marker3)
)

m.add_layer(marker_cluster);

m
```

22.2 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
<th>Doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>markers</td>
<td>()</td>
<td>Tuple of markers</td>
</tr>
</tbody>
</table>
CHAPTER 23

Heatmap

23.1 Example

```python
from ipyleaflet import Map, Heatmap
from random import uniform
m = Map(center=(0, 0), zoom=2)

heatmap = Heatmap(
    locations=[[uniform(-80, 80), uniform(-180, 180), uniform(0, 1000)] for i in range(1000)],
    radius=20
)

m.add_layer(heatmap);

m
```

23.2 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
<th>Doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>locations</td>
<td>[]</td>
<td>List of center locations</td>
</tr>
<tr>
<td>min_opacity</td>
<td>0.05</td>
<td>Minimum opacity the heat will start at</td>
</tr>
<tr>
<td>max_zoom</td>
<td>18</td>
<td>Zoom level where max intensity is reached</td>
</tr>
<tr>
<td>max</td>
<td>1.0</td>
<td>Maximum point intensity</td>
</tr>
<tr>
<td>radius</td>
<td>25.0</td>
<td>Radius of each “point” of the heatmap</td>
</tr>
<tr>
<td>blur</td>
<td>15.0</td>
<td>Amount of blur</td>
</tr>
<tr>
<td>gradient</td>
<td>{0.4: ‘blue’, 0.6: ‘cyan’, 0.7: ‘lime’, 0.8: ‘yellow’, 1.0: ‘red’}</td>
<td>Color gradient config</td>
</tr>
</tbody>
</table>
24.1 Example

```python
from ipyleaflet import Map, TileLayer, basemaps
from ipyleaflet.velocity import Velocity
import xarray as xr
import os

if not os.path.exists('wind-global.nc'):
    url = 'https://github.com/benbovy/xvelmap/raw/master/notebooks/wind-global.nc'
    import requests
    r = requests.get(url)
    wind_data = r.content
    with open('wind-global.nc', 'wb') as f:
        f.write(wind_data)

center = [0, 0]
zoom = 1
m = Map(center=center, zoom=zoom, interpolation='nearest', basemap=basemaps.CartoDB.DarkMatter)

ds = xr.open_dataset('wind-global.nc')
display_options = {
    'velocityType': 'Global Wind',
    'displayPosition': 'bottomleft',
    'displayEmptyString': 'No wind data'
}
wind = Velocity(data=ds,
    zonal_speed='u_wind',
    meridional_speed='v_wind',
    latitude_dimension='lat',
    longitude_dimension='lon',
    velocity_scale=0.01,
    max_velocity=20,
```

(continues on next page)
m.add_layer(wind)

24.2 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
<th>Doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Empty dataset</td>
<td>Underlying dataset</td>
</tr>
<tr>
<td>zonal_speed</td>
<td>''</td>
<td>Variable name in underlying dataset for the zonal speed</td>
</tr>
<tr>
<td>meridional_speed</td>
<td>''</td>
<td>Variable name in underlying dataset for the meridional speed</td>
</tr>
<tr>
<td>latitude_dimension</td>
<td>‘latitude’</td>
<td>Name of the latitude dimension in underlying dataset</td>
</tr>
<tr>
<td>longitude_dimension</td>
<td>‘longitude’</td>
<td>Name of the longitude dimension in underlying dataset</td>
</tr>
<tr>
<td>units</td>
<td>None</td>
<td>Units</td>
</tr>
<tr>
<td>display_values</td>
<td>True</td>
<td>Display velocity data on mouse hover</td>
</tr>
<tr>
<td>display_options</td>
<td>{}</td>
<td>Display options</td>
</tr>
<tr>
<td>min_velocity</td>
<td>0.0</td>
<td>Used to align color scale</td>
</tr>
<tr>
<td>max_velocity</td>
<td>10.0</td>
<td>Used to align color scale</td>
</tr>
<tr>
<td>velocity_scale</td>
<td>0.005</td>
<td>Modifier for particle animations</td>
</tr>
<tr>
<td>color_scale</td>
<td>[]</td>
<td>Array of hex/rgb colors for user-specified color scale.</td>
</tr>
</tbody>
</table>
25.1 Example

```python
from ipyleaflet import (Map, basemaps, basemap_to_tiles, Circle, Marker, Rectangle, LayerGroup)

toner = basemap_to_tiles(basemaps.Stamen.Toner)

m = Map(layers=(toner, ), center=(50, 354), zoom=5)

# Create some layers
marker = Marker(location=(50, 354))
circle = Circle(location=(50, 370), radius=50000, color="yellow", fill_color="yellow")
rectangle = Rectangle(bounds=((54, 354), (55, 360)), color="orange", fill_color="orange")

# Create layer group
layer_group = LayerGroup(layers=(marker, circle))

m.add_layer(layer_group)

layer_group.add_layer(rectangle)

layer_group.remove_layer(circle)

m
```
25.2 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
<th>Doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>layers</td>
<td>()</td>
<td>List of layers</td>
</tr>
</tbody>
</table>

25.3 Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Arguments</th>
<th>Doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_layer</td>
<td>Layer instance</td>
<td>Add a new layer to the group</td>
</tr>
<tr>
<td>remove_layer</td>
<td>Layer instance</td>
<td>Remove a layer from the group</td>
</tr>
<tr>
<td>clear_layers</td>
<td></td>
<td>Remove all layers from the group</td>
</tr>
</tbody>
</table>
CHAPTER 26

GeoJSON

26.1 Example

import os
import json
import random
import requests

from ipyleaflet import Map, GeoJSON

if not os.path.exists('europe_110.geo.json'):
    url = 'https://github.com/jupyter-widgets/ipyleaflet/raw/master/examples/europe_→110.geo.json'
    r = requests.get(url)
    with open('europe_110.geo.json', 'w') as f:
        f.write(r.content.decode("utf-8"))

with open('europe_110.geo.json', 'r') as f:
    data = json.load(f)

def random_color(feature):
    return {
        'color': 'black',
        'fillColor': random.choice(['red', 'yellow', 'green', 'orange']),
    }

m = Map(center=(50.6252978589571, 0.34580993652344), zoom=3)

geo_json = GeoJSON(
    data=data,
    style={
        'opacity': 1, 'dashArray': '9', 'fillOpacity': 0.1, 'weight': 1
    },
    hover_style={

{'color': 'white', 'dashArray': '0', 'fillOpacity': 0.5}
},
    style_callback=random_color
)m.add_layer(geo_json)

26.2 Usage

The GeoJSON layer is a widget, which means that you can update the data or any other attribute from Python and it will dynamically update the map:

geo_json.data = new_data
geo_json.hover_style = new_hover_style

26.3 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Data dictionary</td>
</tr>
<tr>
<td>style</td>
<td>Style dictionary</td>
</tr>
<tr>
<td>hover_style</td>
<td>Hover style dictionary</td>
</tr>
<tr>
<td>style_callback</td>
<td>Styling function that is called for each feature, and should return the feature style. This styling function takes the feature as argument.</td>
</tr>
</tbody>
</table>

26.4 Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Arguments</th>
<th>Doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>on_click</td>
<td>Callable object</td>
<td>Adds a callback on click event</td>
</tr>
<tr>
<td>on_hover</td>
<td>Callable object</td>
<td>Adds a callback on hover event</td>
</tr>
</tbody>
</table>
GeoData is an ipyleaflet class that allows you to visualize a GeoDataFrame on the Map.

27.1 Example

```python
from ipyleaflet import Map, GeoData, basemaps, LayersControl
import geopandas
def json

countries = geopandas.read_file(geopandas.datasets.get_path('naturalearth_lowres'))
→naturalearthdata.com/download/10m/physical/ne_10m_rivers_lake_centerlines.zip")
m = Map(center=(52.3,8.0), zoom = 3, basemap= basemaps.Esri.WorldTopoMap)

geo_data = GeoData(geo_dataframe = countries,
                  style={'color': 'black', 'fillColor': '#3366cc', 'opacity':0.05,
                  'weight':1.9, 'dashArray':"2", 'fillOpacity':0.6},
                  hover_style={'fillColor': 'red', 'fillOpacity': 0.2},
                  name = 'Countries')

rivers_data = GeoData(geo_dataframe = rivers,
                      style={'color': 'purple', 'opacity':3, 'weight':1.9, 'dashArray':"2
                      ", 'fillOpacity':0.6},
                      hover_style={'fillColor': 'red', 'fillOpacity': 0.2},
                      name = 'Rivers')

m.add_layer(rivers_data)
m.add_layer(geo_data)
m.add_control(LayersControl())
m
```
### Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Doc</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geo_data</td>
<td>Data dictionary</td>
<td>GeoDataFrame</td>
</tr>
<tr>
<td>style</td>
<td>Style dictionary</td>
<td></td>
</tr>
<tr>
<td>hover_style</td>
<td>Hover style dictionary</td>
<td></td>
</tr>
</tbody>
</table>
28.1 Example

```python
import ipyleaflet
import json
import pandas as pd
import os
import requests
from ipywidgets import link, FloatSlider
from branca.colormap import linear

def load_data(url, filename, file_type):
    r = requests.get(url)
    with open(filename, 'w') as f:
        f.write(r.content.decode("utf-8"))
    with open(filename, 'r') as f:
        return file_type(f)


unemployment = dict(zip(unemployment['State'].tolist(), unemployment['Unemployment'].tolist()))

layer = ipyleaflet.Choropleth(
```
geo_data=geo_json_data,
choro_data=unemployment,
colormap=linear.YlOrRd_04,
border_color='black',
style={'fillOpacity': 0.8, 'dashArray': '5, 5'})

m = ipyleaflet.Map(center = (43,-100), zoom = 4)
m.add_layer(layer)
m

### 28.2 Usage

The Choropleth takes geo_data and choro_data as arguments.

The geo_data is a GeoJSON dictionary, for instance:

```json
{
   "type": "FeatureCollection",
   "features": [{
      "type": "Feature",
      "id": "AL",
      "properties": {"name": "Alabama"},
      "geometry": {
         "type": "Polygon",
         "coordinates": [[[-87.359296, 35.00118]]] ...
      }
   }
}
```

The choro_data is a dictionary that maps an key to a float value, in order to build the colormap:

```json
{'AL': 7.1, 'AK': 6.8}
```

The Choropleth layer is then created specifying on which key the colormap is applied:

```python
Choropleth(
   geo_data=geo_data,
   choro_data=choro_data,
   key_on='id'
)
```
# 28.3 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default</th>
<th>Doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>geo_data</td>
<td>{}</td>
<td>Data dictionary</td>
</tr>
<tr>
<td>choro_data</td>
<td>{}</td>
<td>Mapping key -&gt; float data for constructing the colormap</td>
</tr>
<tr>
<td>key_on</td>
<td>‘id’</td>
<td>Key used for the colormap construction</td>
</tr>
<tr>
<td>value_min</td>
<td></td>
<td>Color scale minimum value</td>
</tr>
<tr>
<td>value_max</td>
<td></td>
<td>Color scale maximum value</td>
</tr>
<tr>
<td>colormap</td>
<td>OrRd</td>
<td>Map of color from branca</td>
</tr>
<tr>
<td>style</td>
<td></td>
<td>Style dictionary</td>
</tr>
<tr>
<td>hover_style</td>
<td></td>
<td>Hover style dictionary</td>
</tr>
<tr>
<td>style_callback</td>
<td></td>
<td>Styling function that is called for each feature, and should return the feature style.</td>
</tr>
</tbody>
</table>

This styling function takes the feature, the colormap function and the key data as arguments.
CHAPTER 29

Vector Tile Layer

29.1 Example

```python
from ipyleaflet import Map, VectorTileLayer

from traitlets import Unicode, Dict

# This is a custom VectorTileLayer subclass, allowing to pass our api key to the url
class CustomVectorTileLayer(VectorTileLayer):
    api_key = Unicode('gCZXZglvRQa6sB2z7JzL1w').tag(sync=True, o=True)

water_style = dict(
    fill="true",
    weight=1,
    fillColor="#06cccc",
    color="#06cccc",
    fillOpacity=0.2,
    opacity=0.4,
)

waterway_style = dict(
    weight=1, fillColor="#2375e0", color="#2375e0",
    fillOpacity=0.2, opacity=0.4
)

admin_style = dict(
    weight=1, fillColor="pink", color="pink",
    fillOpacity=0.2, opacity=0.4
)

landcover_style = dict(
    fill="true",
    weight=1,
    fillColor="#53e033",
    color="#53e033",
    fillOpacity=0.2,
)
```

(continues on next page)
opacity=0.4,
}

landuse_style = dict(
    fill="true",
    weight=1,
    fillColor="#e5b404",
    color="#e5b404",
    fillOpacity=0.2,
    opacity=0.4,
)

park_style = dict(
    fill="true",
    weight=1,
    fillColor="#84ea5b",
    color="#84ea5b",
    fillOpacity=0.2,
    opacity=0.4,
)

boundary_style = dict(
    weight=1, fillColor="#c545d3", color="#c545d3", fillOpacity=0.2, opacity=0.4
)

aeroway = dict(
    weight=1, fillColor="#51aeb5", color="#51aeb5", fillOpacity=0.2, opacity=0.4
)

road = dict(
    weight=1, fillColor="#f2b648", color="#f2b648", fillOpacity=0.2, opacity=0.4
)

transit = dict(
    weight=0.5, fillColor="#f2b648", color="#f2b648", fillOpacity=0.2, opacity=0.4
)

buildings = dict(
    fill="true",
    weight=1,
    fillColor="#2b2b2b",
    color="#2b2b2b",
    fillOpacity=0.2,
    opacity=0.4,
)

water_name = dict(
    weight=1, fillColor="#022c5b", color="#022c5b", fillOpacity=0.2, opacity=0.4
)

transportation_name = dict(
    weight=1, fillColor="#bc6b38", color="#bc6b38", fillOpacity=0.2, opacity=0.4
)

place = dict(
    weight=1, fillColor="#f20e93", color="#f20e93", fillOpacity=0.2, opacity=0.4
)
housenumber = dict(
    weight=1, fillColor="#ef4c8b", color="#ef4c8b", fillOpacity=0.2, opacity=0.4
)

poi = dict(weight=1, fillColor="#3bb50a", color="#3bb50a", fillOpacity=0.2, opacity=0.4)

earth = dict(
    fill="true",
    weight=1,
    fillColor="#c0c0c0",
    color="#c0c0c0",
    fillOpacity=0.2,
    opacity=0.4,
)

url = 'https://tile.nextzen.org/tilezen/vector/v1/512/all/{z}/{x}/{y}.mvt?api_key=

vector_tile_layer_styles = dict(
    water=water_style,
    waterway=waterway_style,
    admin=admin_style,
    andcover=landcover_style,
    landuse=landuse_style,
    park=park_style,
    boundaries=boundary_style,
    aeroway=aeroway,
    roads=road,
    transit=transit,
    buildings=buildings,
    water_name=water_name,
    transportation_name=transportation_name,
    places=place,
    housenumber=housenumber,
    pois=poi,
    earth=earth
)

m = Map(center=(52.204793, 360.121558), zoom=9)
v1 = CustomVectorTileLayer(url=url, vector_tile_layer_styles=vector_tile_layer_styles)
m.add_layer(v1)

m

## 29.2 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
<th>Doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>'</td>
<td>Url for the source protobuf data.</td>
</tr>
<tr>
<td>attribution</td>
<td>'Map data (c) &lt;a href=&quot;https://openstreetmap.org&quot;&gt;OpenStreetMap&lt;/a&gt; contributors'</td>
<td>Attribution for the map.</td>
</tr>
<tr>
<td>vector_tile_layer_styles</td>
<td></td>
<td>Styles for the various data layer of protobuf layers.</td>
</tr>
</tbody>
</table>
CHAPTER 30

Zoom Control

30.1 Example

```python
from ipyleaflet import Map, ZoomControl

m = Map(zoom=5, center=[51.64, -76.52], zoom_control=False)  # Do not automatically create a ZoomControl
m.add_control(ZoomControl(position='topright'))
m
```

30.2 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
<th>Doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>‘topleft’</td>
<td>Position of the control, can be ‘bottomleft’, ‘bottomright’, ‘topleft’, or ‘topright’</td>
</tr>
<tr>
<td>zoom_in_text</td>
<td>‘+’</td>
<td>Text to show in the “zoom in” button</td>
</tr>
<tr>
<td>zoom_in_title</td>
<td>‘Zoom in’</td>
<td>Text to show on mouse hover on the “zoom in” button</td>
</tr>
<tr>
<td>zoom_out_text</td>
<td>‘-’</td>
<td>Text to show in the “zoom out” button</td>
</tr>
<tr>
<td>zoom_out_title</td>
<td>‘Zoom out’</td>
<td>Text to show on mouse hover on the “zoom out” button</td>
</tr>
</tbody>
</table>
Chapter 30. Zoom Control
CHAPTER 31

Scale Control

31.1 Example

```python
from ipyleaflet import Map, ScaleControl
m = Map(zoom=5, center=[51.64, -76.52])
m.add_control(ScaleControl(position='bottomleft'))
m
```

31.2 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
<th>Doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>'topleft'</td>
<td>Position of the control, can be ‘bottomleft’, ‘bottomright’, ‘topleft’, or ‘topright’</td>
</tr>
<tr>
<td>max_width</td>
<td>100</td>
<td>Maximum width of the control in pixels. The width is set dynamically to show round values (e.g. 100, 200, 500).</td>
</tr>
<tr>
<td>metric</td>
<td>True</td>
<td>Whether to show the metric scale line (m/km).</td>
</tr>
<tr>
<td>imperial</td>
<td>True</td>
<td>Whether to show the imperial scale line (mi/ft).</td>
</tr>
<tr>
<td>update_when_idle</td>
<td>False</td>
<td>If true, the control is updated only after ending dragging the Map, otherwise it’s always up-to-date (updated during move).</td>
</tr>
</tbody>
</table>
The LayersControl allows one to display a layer selector on the map in order to select which layers to display on the map.

All layers have a name attribute which is displayed in the selector and can be changed by the user.

```python
from ipyleaflet import Map, Marker, LayersControl

m = Map(center=(50, 0), zoom=5)

marker1 = Marker(name='marker1', location=(48, -2))
marker2 = Marker(name='marker2', location=(50, 0))
marker3 = Marker(name='marker3', location=(52, 2))
m.add_layer(marker1)
m.add_layer(marker2)
m.add_layer(marker3)

control = LayersControl(position='topright')
m.add_control(control)

m
```
33.1 Example

```python
from ipyleaflet import Map, FullScreenControl

m = Map(zoom=5, center=[51.64, -76.52])
m.add_control(FullScreenControl())
m
```

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
<th>Doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>‘topleft’</td>
<td>Position of the control, can be ‘bottomleft’, ‘bottomright’, ‘topleft’, or ‘topright’</td>
</tr>
</tbody>
</table>
CHAPTER 34

Measure Control

34.1 Example

```python
from ipyleaflet import Map, MeasureControl, basemaps

m = Map(center=(43.0327, 6.0232), zoom=9, basemap=basemaps.Hydda.Full)

measure = MeasureControl(
    position='bottomleft',
    active_color='orange',
    primary_length_unit='kilometers'
)

m.add_control(measure)

measure.completed_color = 'red'

measure.add_length_unit('yards', 1.09361, 4)
measure.secondary_length_unit = 'yards'

measure.add_area_unit('sqyards', 1.19599, 4)
measure.secondary_area_unit = 'sqyards'

m
```
34.2 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
<th>Doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>“topright”</td>
<td>Position of the control on the Map, possible values are topleft, topright, bottomleft or bottomright</td>
</tr>
<tr>
<td>primary_length_unit</td>
<td>“feet”</td>
<td>Primary length unit, possible values are feet, meters, miles, kilometers or any user defined length unit</td>
</tr>
<tr>
<td>secondary_length_unit</td>
<td>None</td>
<td>Secondary length unit, possible values are None, feet, meters, miles, kilometers or any user defined length unit</td>
</tr>
<tr>
<td>primary_area_unit</td>
<td>“acres”</td>
<td>Primary area unit, possible values are acres, hectares, sqfeet, sqmeters, sqmiles or any user defined area unit</td>
</tr>
<tr>
<td>secondary_area_unit</td>
<td>None</td>
<td>Secondary area unit, possible values are None, acres, hectares, sqfeet, sqmeters, sqmiles or any user defined area unit</td>
</tr>
<tr>
<td>active_color</td>
<td>“#ABE67E”</td>
<td>Color of the currently drawn area</td>
</tr>
<tr>
<td>completed_color</td>
<td>“#C8F2BE”</td>
<td>Color of the completed areas</td>
</tr>
<tr>
<td>popup_options</td>
<td></td>
<td></td>
</tr>
<tr>
<td>capture_z_index</td>
<td>10000</td>
<td>Z-index of the marker used to capture measure clicks. Set this value higher than the z-index of all other map layers to disable click events on other layers while a measurement is active.</td>
</tr>
</tbody>
</table>

34.3 Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Arguments</th>
<th>Doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_length_unit</td>
<td>name, factor, decimals=0</td>
<td>Adds a length unit with a name, a factor (factor to apply when converting to this unit. Length in meters will be multiplied by this factor), and an optional number of displayed decimals</td>
</tr>
<tr>
<td>add_area_unit</td>
<td>name, factor, decimals=0</td>
<td>Adds a area unit with a name, a factor (factor to apply when converting to this unit. Area in sqmeters will be multiplied by this factor), and an optional number of displayed decimals</td>
</tr>
</tbody>
</table>
35.1 Example

```python
from ipyleaflet import Map, basemaps, basemap_to_tiles, SplitMapControl

m = Map(center=(42.6824, 365.581), zoom=5)

right_layer = basemap_to_tiles(basemaps.NASAGIBS.ModisTerraTrueColorCR, "2017-11-11")
left_layer = basemap_to_tiles(basemaps.NASAGIBS.ModisAquaBands721CR, "2017-11-11")

control = SplitMapControl(left_layer=left_layer, right_layer=right_layer)
m.add_control(control)

m
```

35.2 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Default Value</th>
<th>Doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>left_layer</td>
<td>Layer instance</td>
<td></td>
<td>Left layer</td>
</tr>
<tr>
<td>right_layer</td>
<td>Layer instance</td>
<td></td>
<td>Right layer</td>
</tr>
</tbody>
</table>
CHAPTER 36

Draw Control

The DrawControl allows one to draw shapes on the map such as Rectangle Circle or lines.

```python
from ipyleaflet import Map, basemaps, basemap_to_tiles, DrawControl

watercolor = basemap_to_tiles(basemaps.Stamen.Watercolor)

m = Map(layers=(watercolor, ), center=(50, 354), zoom=5)

draw_control = DrawControl()

draw_control.polyline = {
    "shapeOptions": {
        "color": "#6bc2e5",
        "weight": 8,
        "opacity": 1.0
    }
}

draw_control.polygon = {
    "shapeOptions": {
        "fillColor": "#6be5c3",
        "color": "#6be5c3",
        "fillOpacity": 1.0
    }
}

draw_control.circle = {
    "shapeOptions": {
        "fillColor": "#efed69",
        "color": "#efed69",
        "fillOpacity": 1.0
    }
}
```

(continues on next page)
draw_control.rectangle = {
    "shapeOptions": {
        "fillColor": "#fca45d",
        "color": "#fca45d",
        "fillOpacity": 1.0
    }
}

m.add_control(draw_control)
CHAPTER 37

Widget Control

37.1 Example

```python
from ipyleaflet import Map, basemaps, WidgetControl
from ipywidgets import IntSlider, ColorPicker, jslink

m = Map(center=(46.01, 6.16), zoom=12, basemap=basemaps.Stamen.Terrain)
zoom_slider = IntSlider(description='Zoom level:', min=0, max=15, value=7)
jslink((zoom_slider, 'value'), (m, 'zoom'))
widget_control1 = WidgetControl(widget=zoom_slider, position='topright')
m.add_control(widget_control1)

color_picker = ColorPicker(description='Pick a color:')
widget_control2 = WidgetControl(widget=color_picker, position='bottomright')
m.add_control(widget_control2)
m
```

37.2 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
<th>Doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>'topleft'</td>
<td>Position of the control, can be 'bottomleft', 'bottomright', 'topleft', or 'topright'</td>
</tr>
<tr>
<td>widget</td>
<td>None</td>
<td>Widget content</td>
</tr>
<tr>
<td>min_width</td>
<td>None</td>
<td>Min width of the widget (in pixels), if None it will respect the content size</td>
</tr>
<tr>
<td>max_width</td>
<td>None</td>
<td>Max width of the widget (in pixels), if None it will respect the content size</td>
</tr>
<tr>
<td>min_height</td>
<td>None</td>
<td>Min height of the widget (in pixels), if None it will respect the content size</td>
</tr>
<tr>
<td>max_height</td>
<td>None</td>
<td>Max height of the widget (in pixels), if None it will respect the content size</td>
</tr>
</tbody>
</table>
38.1 Example

```python
from ipyleaflet import Map, LegendControl

m = Map(center=(-10,-45), zoom=4)

legend = LegendControl({"low":"#FAA", "medium":"#A55", "High":"#500"}, name="Legend", position="bottomright")
m.add_control(legend)

# Manipulate the legend

# Set/Get legend title
legend.name = "Risk" # Set name
legend.name # Get name

# Set/Get legend content
legend.legends = {"el1":"#FAA", "el2":"#A55", "el3":"#500"} # Set content
legend.legends # Get content

legend.add_legend_element("el5","#000") # Add a legend element
legend.remove_legend_element("el5") # Remove a legend element

# legend position
legend.position = "topright" # Set position
legend.position # Get current position

'mtopright'
```
## 38.2 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
<th>Doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>'topleft'</td>
<td>Position of the control, can be ‘bottomleft’, ‘bottomright’, ‘topleft’, or ‘topright’</td>
</tr>
<tr>
<td>legend</td>
<td>None</td>
<td>A dictionary containing the name-&gt;color mapping that represents the legend</td>
</tr>
<tr>
<td>title</td>
<td>‘Legend’</td>
<td>Legend name</td>
</tr>
</tbody>
</table>
CHAPTER 39

Search Control

39.1 Example

```python
from ipyleaflet import Map, SearchControl, Marker, AwesomeIcon

m = Map(zoom=3, center=[19.1646, 72.8493])

marker = Marker(icon=AwesomeIcon(name="check", marker_color='green', icon_color='darkgreen'))

m.add_control(SearchControl(
    position="topleft",
    url='https://nominatim.openstreetmap.org/search?format=json&q={s}',
    zoom=5,
    marker=marker
))

m
```

You can also search features from GeoJSON layers.

```python
import json
import os
import requests

from ipyleaflet import AwesomeIcon, GeoJSON, Map, Marker, LayerGroup, SearchControl

m = Map(zoom=3, center=[19.1646, 72.8493])

if not os.path.exists('countries.geo.json'):
    url = 'https://raw.githubusercontent.com/jupyter-widgets/ipyleaflet/master/examples/countries.geo.json'
    r = requests.get(url)
    with open('countries.geo.json', 'w') as f:
        ...  # write response content to file
```

(continues on next page)
f.write(r.content.decode("utf-8"))

with open("countries.geo.json") as f:
    data = json.load(f)

countries = GeoJSON(data=data)

layer_group = LayerGroup(layers=(countries,))
marker = Marker(icon=AwesomeIcon(name="check", marker_color='green', icon_color='darkred'))

m.add_control(SearchControl(
    position="topleft",
    layer=layer_group,
    zoom=4,
    property_name='name',
    marker=marker
))

m

### 39.2 Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Default Value</th>
<th>Doc</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>‘topleft’</td>
<td>Position of the control, can be ‘bottomleft’, ‘bottomright’, ‘topleft’, or ‘topright’</td>
</tr>
<tr>
<td>url</td>
<td>‘’</td>
<td>The url used for the search queries.</td>
</tr>
<tr>
<td>layer</td>
<td>None</td>
<td>The LayerGroup used for search queries.</td>
</tr>
<tr>
<td>zoom</td>
<td>10</td>
<td>Default zoom level for move to location</td>
</tr>
<tr>
<td>marker</td>
<td>Marker()</td>
<td>The marker used by the control.</td>
</tr>
<tr>
<td>found_style</td>
<td>{'fillColor': '#3f0', 'color': '#0f0'}</td>
<td>Style for searched feature when searching in LayerGroup.</td>
</tr>
</tbody>
</table>
Here is a list of existing open source projects that build functionality upon ipyleaflet.

- **geemap**: a Python package for interactive mapping with Google Earth Engine, ipyleaflet, and ipywidgets.
- **xarray-leaflet**: an xarray extension for tiled map plotting, based on ipyleaflet.
- **ipygee**: another Python package for interactive mapping with Google Earth Engine, ipyleaflet, and ipywidgets.