xleaflet

Johan Mabille, Sylvain Corlay, Wolf Vollprecht and Martin Renou

Nov 27, 2020

INSTALLATION

The C++ backend for ipyleaflet.

`xleaflet` is a C++ backend for the ipyleaflet maps visualization library.

`xleaflet` and its dependencies require a modern C++ compiler supporting C++14. The following C++ compilers are supported:

- On Windows platforms, Visual C++ 2015 Update 2, or more recent
- On Unix platforms, gcc 4.9 or a recent version of Clang

CHAPTER

ONE

LICENSING

We use a shared copyright model that enables all contributors to maintain the copyright on their contributions.

This software is licensed under the BSD-3-Clause license. See the LICENSE file for details.

1.1 Installation

xleaflet is a header-only library but depends on some traditional libraries that need to be installed. On Linux, installation of the dependencies can be done through the package manager, anaconda or manual compilation.

1.1.1 Using the conda package

A package for xleaflet is available on the conda package manager. The package will also pull all the dependencies.

```
conda install xleaflet -c conda-forge
```

1.1.2 From source with cmake

You can also install `xleaflet` from source with cmake. On Unix platforms, from the source directory: However, you need to make sure to have the required libraries available on your machine.

```
mkdir build
cd build
cmake -DCMAKE_INSTALL_PREFIX=/path/to/prefix ..
make install
```

On Windows platforms, from the source directory:

```
mkdir build
cd build
cmake -G "NMake Makefiles" -DCMAKE_INSTALL_PREFIX=/path/to/prefix ..
nmake
nmake install
```

1.2 Introduction

`xleaflet` is the C++ backend for the leaflet maps visualization library. The Python reference implementation is available in the ipyleaflet project.

`xleaflet` depends on xwidgets, each object that you can create in `xleaflet` is an xwidget instance which is synchronized with one or more views on the frontend. See the `xwidgets` documentation for the usage of widgets.

1.3 Basic usage

1.3.1 Default map

The default map can be displayed using Jupyter's display framework.

```
xlf::map map;
map.display();
```

Changing widget attributes can be done from the model:

```
map.zoom = 15;

// latitude 52.204793, longitude 360.121558
map.center = std::array<double, 2>({52.204793, 360.121558});
```

Or by interacting directly with the view.

The map widget works with a list of layers. Layers are instances of `tile_layer`, `marker`, `popup`, `wms_layer`, `image_overlay`, `video_overlay`, `polygon`, `rectangle`, `circle_marker`, `circle`, `marker_cluster`, `layer_group` or `geo_json`.

```
#include "xleaflet/xmap.hpp"

xlf::map map;

map.add_layer(marker);
map.add_layer(circle);
map.add_layer(layer_group);

map.remove_layer(circle);

map.clear_layers();
```

It is also possible to have a list of controls on the map. Controls are instances of `layers_control`, `split_map_control` or `draw_control`.

```
#include "xleaflet/xmap.hpp"

xlf::map map;

map.add_control(control1);
map.add_control(control2);

map.remove_control(control1);

map.clear_controls();
```

1.4 Generator classes

Widgets such as `map` may have a large number of attributes that can be set by the user, such as `center`, `zoom`, `min_zoom`, `max_zoom`, `scroll_wheel_zoom`, `bounce_at_zoom_limits`, `inertia`.

Providing a constructor for `map` with a large number of such attributes would make the use of `xleaflet` very cumbersome, because users would need to know all the positional arguments to modify only one value. Instead, we mimick a keyword argument initialization with a method-chaining mechanism.

```
#include "xleaflet/xmap.hpp"

auto map = xlf::map::initialize()
    .center({52.204793, 360.121558})
    .zoom(15)
    .scroll_wheel_zoom(true)
    .inertia(false)
    .finalize();
```

1.5 Special Events

One could want to react on special map events like `mousemove`, this can be achieved by using the `on_interaction` method of map:

```
#include <iostream>

#include "xleaflet/xmap.hpp"

void print_mouse_position(xeus::xjson event)
{
    if (event["type"] == "mousemove")
    {
        std::cout << "Mouse position: " << event["coordinates"].dump() << std::endl;
    }

    if (event["type"] == "mouseout")
    {
        std::cout << "Mouse out" << std::endl;;
    }
}

xlf::map map;

map.on_interaction(print_mouse_position);
map.display();
```

1.6 XProperty Events

`xleaflet` relies on the xproperty library, so that one could use the `XOBSERVE` function to react on model changes:

```
#include <iostream>

#include "xleaflet/xmap.hpp"

void print_lat_lng(xlf::map& map)
{
    std::string lat = std::to_string(map.center().front());
    std::string lng = std::to_string(map.center().back());

    std::cout << "latitude: " << lat << ", longitude: " << lng << std::endl;
}

xlf::map map;

XOBSERVE(map, center, print_lat_lng);
map.display();
```

1.7 Map

1.7.1 Example

```
#include "xleaflet/xmap.hpp"
#include "xleaflet/xbasemaps.hpp"

auto map = xlf::map::initialize()
    .layers({xlf::basemap({"NASAGIBS", "ModisTerraTrueColorCR"}, "2017-04-08")})
    .center({52.204793, 360.121558})
    .zoom(4)
    .finalize();
map
```

1.7.2 Attributes

Attribute	Type	Default Value	Doc
layers	`std::vector<xlf::Layer>`	{default_layer}	Vector of layers
controls	`std::vector<xlf::Control>`	{}	Vector of controls
center	`std::array<double, 2>`	{0.0, 0.0}	Initial geographic center of the map
zoom	`int`	12	Initial map zoom level
max_zoom	`int`	18	
min_zoom	`int`	1	
dragging	`bool`	true	Whether the map be draggable with mouse/touch or not
touch_zoom	`bool`	true	Whether the map can be zoomed by touch-dragging with two fingers on mobile
scroll_wheel_zoom	`bool`	false	Whether the map can be zoomed by using the mouse wheel
double_click_zoom	`bool`	true	Whether the map can be zoomed in by double clicking on it and zoomed out by double clicking while holding shift
box_zoom	`bool`	true	Whether the map can be zoomed to a rectangular area specified by dragging the mouse while pressing the shift key
tap	`bool`	true	Enables mobile hacks for supporting instant taps
tap_tolerance	`int`	15	The max number of pixels a user can shift his finger during touch for it to be considered a valid tap
world_copy_jump	`bool`	false	With this option enabled, the map tracks when you pan to another "copy" of the world and seamlessly jumps to
close_popup_on_click	`bool`	true	Set it to false if you don't want popups to close when user clicks the map
bounce_at_zoom_limits	`bool`	true	Set it to false if you don't want the map to zoom beyond min/max zoom and then bounce back when pinch-zooming
keyboard	`bool`	true	Makes the map focusable and allows users to navigate the map with keyboard arrows and +/- keys
keyboard_pan_offset	`int`	80	
keyboard_zoom_offset	`int`	1	
inertia	`bool`	true	If enabled, panning of the map will have an inertia effect
inertia_deceleration	`int`	3000	The rate with which the inertial movement slows down, in pixels/second^2
inertia_max_speed	`int`	1500	Max speed of the inertial movement, in pixels/second
zoom_control	`bool`	true	
attribution_control	`bool`	true	
zoom_animation_threshold	`int`	4	

1.7.3 Methods

Method	Return type	Arguments	Doc
add_layer	`void`	`xlf::layer`	Add a new layer to the map
remove_layer	`void`	`xlf::layer`	Remove a layer from the map
clear_layers	`void`		Remove all layers from the map
add_control	`void`	`xlf::control`	Add a new control to the map
remove_control	`void`	`xlf::control`	Remove a control from the map
clear_controls	`void`		Remove all controls from the map
on_interaction	`void`	`std::function<void(xeus::xjson)>`	Add a callback on interaction

1.8 Tile layer

1.8.1 Example

```
#include "xleaflet/xmap.hpp"
#include "xleaflet/xbasemaps.hpp"

auto map = xlf::map::initialize()
    .center({52.204793, 360.121558})
    .zoom(9)
    .finalize();

auto dark_matter_layer = xlf::basemap({"CartoDB", "DarkMatter"});
map.add_layer(dark_matter_layer);
map
```

1.8.2 Usage

Creating a `tile_layer` is straightforward, a list of basic tile layers is provided. This list of layers can be accessed using the `basemaps` function:

```
#include <iostream>

#include "xleaflet/xbasemaps.hpp"

std::cout << xlf::basemaps().dump(6) << std::endl;
```

A `tile_layer` instance can be created using the `basemap` function, specifying the wanted map (e.g. `{"CartoDB", "DarkMatter"}`, `{"Strava", "Winter"}`, `{"NASAGIBS", "ModisTerraTrueColorCR"}`, ...).

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

```
auto nasa_layer = xlf::basemap({"NASAGIBS", "ModisTerraTrueColorCR"}, "2018-04-08");
map.add_layer(nasa_layer);
```

1.8.3 Attributes

Attribute	Type	Default Value
url	`std::string`	"https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"
min_zoom	`int`	0
max_zoom	`int`	18
tile_size	`int`	256
attribution	`std::string`	"Map data (c) OpenStreetMap contributors"
detect_retina	`bool`	false
opacity	`float`	1.0
visible	`bool`	true

1.9 Marker

1.9.1 Example

```
#include "xleaflet/xmap.hpp"
#include "xleaflet/xmarker.hpp"

std::array<double, 2> center = {52.204793, 360.121558};

auto map = xlf::map::initialize()
    .center(center)
    .zoom(15)
    .finalize();

auto marker = xlf::marker::initialize()
    .location(center)
    .draggable(false)
    .finalize();
map.add_layer(marker);

map
```

1.9.2 Attributes

Attribute	Type	Default Value	Doc
location	`std::array<double, 2>`	{0.0, 0.0}	
z_index_offset	`int`	0	
draggable	`bool`	true	Whether the marker is draggable with mouse/touch or not
keyboard	`bool`	true	Whether the marker can be tabbed to with a keyboard and clicked by pressing enter
title	`std::string`	""	Text for the browser tooltip that appear on marker hover (no tooltip by default)
alt	`std::string`	""	Text for the *alt* attribute of the icon image (useful for accessibility)
rise_on_hover	`bool`	false	The z-index offset used for the *rise_on_hover* feature
opacity	`float`	1.0	
visible	`bool`	true	
icon	`xlf::icon`		The icon for the marker
rotation_angle	`int`	0	The rotation angle of the marker in degrees
rotation_origin	`std::string`	""	The rotation origin of the marker
rise_offset	`int`	250	The z-index offset used for the *rise_on_hover* feature

1.9.3 Methods

Method	Return type	Arguments	Doc
on_move	`void`	`std::function<void(xeus::xjson)>`	Adds a callback on move event

1.10 Icon

1.10.1 Example

```
#include <array>

#include "xleaflet/xmap.hpp"
#include "xleaflet/xmarker.hpp"
#include "xleaflet/xicon.hpp"

using size_type = std::array<int, 2>;

std::array<double, 2> center({52.204793, 360.121558});

auto map = xlf::map::initialize()
    .center(center)
    .zoom(10)
    .finalize();

auto icon = xlf::icon::initialize()
```

(continues on next page)

(continued from previous page)

```
    .icon_url("https://leafletjs.com/examples/custom-icons/leaf-red.png")
    .icon_size(size_type({38, 95}))
    .icon_anchor(size_type({22, 94}))
    .finalize();

auto marker = xlf::marker::initialize()
    .location(center)
    .icon(icon)
    .rotation_angle(0)
    .rotation_origin("22px 94px")
    .finalize();

map.add_layer(marker);

map
```

1.10.2 Attributes

Attribute	Type	Default Value	Doc
icon_url	`std::string`	“”	url for icon image
shadow_url	`std::string`	“”	url for icon shadow image
icon_size	`std::array<int, 2>`	{10, 10}	size of the icon, in pixels
shadow_size	`std::array<int, 2>`	{10, 10}	size of the icon shadow, in pixels
icon_anchor	`std::array<int, 2>`	{0, 0}	anchor point for the icon
shadow_anchor	`std::array<int, 2>`	{0, 0}	anchor point for the icon shadow
popup_anchor	`std::array<int, 2>`	{0, 0}	anchor point for the popup

1.11 Popup

1.11.1 Example

```
#include "xwidgets/xhtml.hpp"

#include "xleaflet/xmap.hpp"
#include "xleaflet/xbasemaps.hpp"
#include "xleaflet/xmarker.hpp"
#include "xleaflet/xpopup.hpp"

std::array<double, 2> center = {52.204793, 360.121558};

auto map = xlf::map::initialize()
    .center(center)
    .zoom(9)
    .close_popup_on_click(false)
    .finalize();
map.display();

auto marker = xlf::marker::initialize()
    .location({52.1, 359.9})
    .finalize();
```

(continues on next page)

(continued from previous page)

```
map.add_layer(marker);

xw::html message1, message2;
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:
auto popup = xlf::popup::initialize()
    .location(center)
    .child(message1)
    .close_button(false)
    .auto_close(false)
    .close_on_escape_key(false)
    .finalize();
map.add_layer(popup);

// Popup associated to a layer
marker.popup = message2;
```

1.11.2 Attributes

Attribute	Type	Default Value	Doc
location	`std::array<double, 2>`	{0.0, 0.0}	
child	`xwidget`		Content of the popup
max_width	`int`	300	Max width of the popup, in pixels
min_width	`int`	50	Min width of the popup, in pixels
max_height	`int`		If set, creates a scrollable container of the given height inside a popup if its content exceeds it
auto_pan	`bool`	true	Set it to *false* if you don't want the map to do panning animation to fit the opened popup
auto_pan_padding	`std::array<double, 2>`	{5, 5}	
keep_in_view	`bool`	false	Set it to *true* if you want to prevent users from panning the popup off of the screen while it is open
close_button	`bool`	true	Controls the presence of a close button in the popup
close_on_escape_key	`bool`	true	Set it to *false* if you want to override the default behavior of the ESC key for closing of the popup
class_name	`std::string`	""	A custom CSS class name to assign to the popup

1.12 WMS layer

1.12.1 Example

```
#include "xleaflet/xmap.hpp"
#include "xleaflet/xwms_layer.hpp"

auto wms = xlf::wms_layer::initialize()
    .url("https://demo.boundlessgeo.com/geoserver/ows?")
    .layers("nasa:bluemarble")
    .finalize();

auto map = xlf::map::initialize()
    .layers({wms})
    .center({42.5531, -48.6914})
    .zoom(3)
    .finalize();

map
```

1.12.2 Attributes

Attribute	Type	Default Value	Doc
url	std::string	"https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"	
min_zoom	int	0	
max_zoom	int	18	
tile_size	int	256	
attribution	std::string	"Map data (c) OpenStreetMap contributors"	
detect_retina	bool	false	
opacity	float	1.0	
visible	bool	true	
service	std::string	"WMS"	
request	std::string	"GetMap"	
layers	std::string	""	Comma-separated list of WMS layers to show
styles	std::string	""	Comma-separated list of WMS styles
format	std::string	"image/jpeg"	WMS image format (use *'image/png'* for layers with transparency)
transparent	bool	false	If *true*, the WMS service will return images with transparency
version	std::string	"1.1.1"	Version of the WMS service to use
crs	std::string	""	

1.13 Image overlay and Video overlay

1.13.1 Example

```
#include "xleaflet/xmap.hpp"
#include "xleaflet/xvideo_overlay.hpp"

auto map = xlf::map::initialize()
    .center({25, -115})
    .zoom(4)
    .finalize();

auto video = xlf::video_overlay::initialize()
    .url("https://www.mapbox.com/bites/00188/patricia_nasa.webm")
    .bounds({{{13, -130}, {32, -100}}})
    .finalize();

map.add_layer(video);
map
```

1.13.2 Attributes

Attribute	Type	Default Value	Doc
url	`std::string`	""	Url to the footage
bounds	`std::array<std::array<double, 2>, 2>`	{{0.0, 0.0}, {0.0, 0.0}}	SW and NE corners of the image

1.14 Polygon

1.14.1 Example

```
#include "xleaflet/xmap.hpp"
#include "xleaflet/xpolygon.hpp"

auto polygon = xlf::polygon::initialize()
    .locations({{{42, -49}, {43, -49}, {43, -48}}})
    .color("green")
    .fill_color("green")
    .finalize();

auto map = xlf::map::initialize()
    .center({42.5531, -48.6914})
    .zoom(6)
    .finalize();
map.add_layer(polygon);

map
```

1.14.2 Attributes

Attribute	Type	Default Value	Doc
locations	`std::vector<std::array<double, 2>>`	{}	List of points of the polygon
stroke	`bool`	true	Set it to *false* to disable borders
color	`std::string`	"#0033FF"	Color of the stroke
opacity	`float`	1.0	Opacity of the stroke
weight	`int`	5	Width of the stroke in pixels
fill	`bool`	true	Whether to fill the polygon or not
fill_color	`std::string`	"#0033FF"	
fill_opacity	`float`	0.2	
dash_array	`std::string`		
line_cap	`std::string`	"round"	
line_join	`std::string`	"round"	

1.15 Rectangle

1.15.1 Example

```
#include "xleaflet/xmap.hpp"
#include "xleaflet/xbasemaps.hpp"
#include "xleaflet/xrectangle.hpp"

auto terrain = xlf::basemap({"Stamen", "Watercolor"});

auto map = xlf::map::initialize()
    .layers({terrain})
    .center({53, 354})
    .zoom(5)
    .finalize();

auto rectangle = xlf::rectangle::initialize()
    .bounds({{{52, 354}, {53, 360}}})
    .finalize();

map.add_layer(rectangle);

map
```

1.15.2 Attributes

Attribute	Type	Default Value	Doc
bounds	`std::array<std::array<double, 2>, 2>`	{}	SW and NE corners of the rectangle
stroke	`bool`	true	Set it to *false* to disable borders
color	`std::string`	"#0033FF"	Color of the stroke
opacity	`float`	1.0	Opacity of the stroke
weight	`int`	5	Width of the stroke in pixels
fill	`bool`	true	Whether to fill the polygon or not
fill_color	`std::string`	"#0033FF"	
fill_opacity	`float`	0.2	
dash_array	`std::string`		
line_cap	`std::string`	"round"	
line_join	`std::string`	"round"	

1.16 Circle

1.16.1 Example

```
#include "xleaflet/xmap.hpp"
#include "xleaflet/xbasemaps.hpp"
#include "xleaflet/xcircle.hpp"

auto terrain = xlf::basemap({"Stamen", "Watercolor"});

auto map = xlf::map::initialize()
    .layers({terrain})
    .center({53, 354})
    .zoom(5)
    .finalize();

auto circle = xlf::circle::initialize()
    .location({50, 354})
    .radius(50000)
    .color("green")
    .fill_color("green")
    .finalize();

map.add_layer(circle);

map
```

1.16.2 Attributes

Attribute	Type	Default Value	Doc
location	`std::array<double, 2>`	{0.0, 0.0}	Location of the circle
radius	`int`	1000	Radius of the circle, in meters
stroke	`bool`	true	Set it to *false* to disable borders
color	`std::string`	"#0033FF"	Color of the stroke
opacity	`float`	1.0	Opacity of the stroke
weight	`int`	5	Width of the stroke in pixels
fill	`bool`	true	Whether to fill the circle or not
fill_color	`std::string`	"#0033FF"	
fill_opacity	`float`	0.2	
dash_array	`std::string`		
line_cap	`std::string`	"round"	
line_join	`std::string`	"round"	

1.17 Circle Marker

1.17.1 Example

```
#include "xleaflet/xmap.hpp"
#include "xleaflet/xbasemaps.hpp"
#include "xleaflet/xcircle_marker.hpp"

auto terrain = xlf::basemap({"Stamen", "Watercolor"});

auto map = xlf::map::initialize()
    .layers({terrain})
    .center({53, 354})
    .zoom(5)
    .finalize();

auto circle_marker = xlf::circle_marker::initialize()
    .location({55, 360})
    .radius(50)
    .color("red")
    .fill_color("red")
    .finalize();

map.add_layer(circle_marker);

map
```

1.17.2 Attributes

Attribute	Type	Default Value	Doc
location	`std::array<double, 2>`	{0.0, 0.0}	Location of the circle
radius	`int`	10	Radius of the circle, in pixels
stroke	`bool`	true	Set it to *false* to disable borders
color	`std::string`	"#0033FF"	Color of the stroke
opacity	`float`	1.0	Opacity of the stroke
weight	`int`	5	Width of the stroke in pixels
fill	`bool`	true	Whether to fill the circle or not
fill_color	`std::string`	"#0033FF"	
fill_opacity	`float`	0.2	
dash_array	`std::string`		
line_cap	`std::string`	"round"	
line_join	`std::string`	"round"	

1.18 Marker Cluster

1.18.1 Example

```
#include "xleaflet/xmap.hpp"
#include "xleaflet/xbasemaps.hpp"
#include "xleaflet/xmarker.hpp"
#include "xleaflet/xmarker_cluster.hpp"

auto map = xlf::map::initialize()
    .center({50, 354})
    .zoom(5)
    .finalize();

auto marker1 = xlf::marker::initialize()
    .location({50, 354})
    .finalize();
auto marker2 = xlf::marker::initialize()
    .location({52, 356})
    .finalize();
auto marker3 = xlf::marker::initialize()
    .location({48, 352})
    .finalize();

auto marker_cluster = xlf::marker_cluster::initialize()
    .markers({marker1, marker2, marker3})
    .finalize();

map.add_layer(marker_cluster);

map
```

1.18.2 Attributes

Attribute	Type	Default Value	Doc
markers	`std::vector<xlf::marker>`	{}	Array of markers

1.19 Heatmap

1.19.1 Example

```
#include <random>
#include <array>
#include <vector>

#include "xleaflet/xmap.hpp"
#include "xleaflet/xheatmap.hpp"

auto map = xlf::map::initialize()
    .center({37.58, 261.65})
    .zoom(5)
    .finalize();

std::random_device rd;
std::mt19937 mt(rd());
std::uniform_real_distribution<double> rd_latitude(34.0, 40.0);
std::uniform_real_distribution<double> rd_longitude(255.0, 265.0);
std::uniform_real_distribution<double> rd_intensity(0.0, 1000.0);

std::vector<std::array<double, 3>> heatmap_points;

for (std::size_t i = 0; i < 100; ++i)
{
    heatmap_points.push_back({rd_latitude(mt), rd_longitude(mt), rd_intensity(mt)});
}

auto heatmap = xlf::heatmap::initialize()
    .locations(heatmap_points)
    .finalize();

map.add_layer(heatmap);

map
```

1.19.2 Attributes

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

1.20 Velocity

1.20.1 Example

```
#include <fstream>

#include "nlohmann/json.hpp"

#include "xleaflet/xmap.hpp"
#include "xleaflet/xvelocity.hpp"

auto map = xlf::map::initialize()
    .center({0, 0})
    .zoom(1)
    .finalize();

auto base_layer = xlf::basemap({"CartoDB", "DarkMatter"});
map.add_layer(base_layer);

std::ifstream file("velocity_data.json");
nlohmann::json data;
file >> data;

auto velocity = xlf::velocity::initialize()
    .data(data)
    .velocity_scale(0.01)
    .max_velocity(20)
    .display_options(R"({
        "velocityType": "Global Wind",
        "displayPosition": "bottomleft",
        "displayEmptyString": "No wind data"
    })")
    .finalize();

map.add_layer(velocity);

map
```

1.20.2 Attributes

Attribute	Default Value	Doc
data	Null JSON objectdataset	Underlying dataset
units	None	Units
display_values	True	Display velocity data on mouse hover
display_options	{}	Display options
min_velocity	0.0	Used to align color scale
max_velocity	10.0	Used to align color scale
velocity_scale	0.005	Modifier for particle animations
color_scale	Empty std::vector of html colors	Array of hex/rgb colors for user-specified color scale.

1.21 Layer Group

1.21.1 Example

```
#include "xleaflet/xmap.hpp"
#include "xleaflet/xbasemaps.hpp"
#include "xleaflet/xlayer_group.hpp"
#include "xleaflet/xcircle.hpp"
#include "xleaflet/xmarker.hpp"
#include "xleaflet/xrectangle.hpp"

auto toner = xlf::basemap({"Stamen", "Toner"});

auto map = xlf::map::initialize()
    .layers({toner})
    .center({50, 354})
    .zoom(5)
    .finalize();

// Create some layers
auto marker = xlf::marker::initialize()
    .location({50, 354})
    .finalize();
auto circle = xlf::circle::initialize()
    .location({50, 370})
    .radius(50000)
    .color("yellow")
    .fill_color("yellow")
    .finalize();
auto rectangle = xlf::rectangle::initialize()
    .bounds({{{54, 354}, {55, 360}}})
    .color("orange")
    .fill_color("orange")
    .finalize();

// Create layer group
auto layer_group = xlf::layer_group::initialize()
    .layers({marker, circle})
    .finalize();

map.add_layer(layer_group);
```

(continues on next page)

(continued from previous page)

```
layer_group.add_layer(rectangle);

layer_group.remove_layer(circle);

map
```

1.21.2 Attributes

Attribute	Type	Default Value	Doc
layers	`std::vector<xlf::layer>`	{}	Array of layers

1.21.3 Methods

Method	Return type	Arguments	Doc
add_layer	`void`	`xlf::layer`	Add a new layer to the group
remove_layer	`void`	`xlf::layer`	Remove a layer from the group
clear_layers	`void`		Remove all layers from the group

1.22 GeoJSON

1.22.1 Example

```
#include <fstream>

#include "xleaflet/xmap.hpp"
#include "xleaflet/xbasemaps.hpp"
#include "xleaflet/xgeo_json.hpp"

auto black_and_white = xlf::basemap({"OpenStreetMap", "BlackAndWhite"});

auto map = xlf::map::initialize()
    .layers({black_and_white})
    .center({34.6252978589571, -77.34580993652344})
    .zoom(10)
    .finalize();

// Load a local file
std::ifstream file("geo.json");
xeus::xjson geo_data;
file >> geo_data;

auto geo_json = xlf::geo_json::initialize()
    .data(geo_data)
    .finalize();
map.add_layer(geo_json);

map
```

1.22.2 Attributes

Attribute	Type	Default Value	Doc
data	`xeus::xjson`		Data dictionary
style	`xeus::xjson`		Style dictionary
hover_style	`xeus::xjson`		Hover style dictionary

1.22.3 Methods

Method	Return type	Arguments	Doc
on_click	`void`	`std::function<void(xeus::xjson)>`	Adds a callback on click event
on_hover	`void`	`std::function<void(xeus::xjson)>`	Adds a callback on hover event

1.23 Fullscreen Control

The `fullscreen_control` allows one to display a selector on the top left of the map in order to display the map in fullscreen.

```
#include "xleaflet/xmap.hpp"
#include "xleaflet/xfullscreen_control.hpp"

auto map = xlf::map::initialize()
    .center({51.64, -76.52})
    .zoom(5)
    .finalize();

map.add_control(xlf::fullscreen_control());

map
```

1.24 Layers Control

The `layers_control` allows one to display a selector on the top right of the map in order to select which tile layer to display on the map.

```
#include "xleaflet/xmap.hpp"
#include "xleaflet/xbasemaps.hpp"
#include "xleaflet/xtile_layer.hpp"
#include "xleaflet/xwms_layer.hpp"
#include "xleaflet/xlayers_control.hpp"

auto map = xlf::map::initialize()
    .center({50, 354})
    .zoom(4)
    .finalize();

auto nasa_layer = xlf::basemap({"NASAGIBS", "ModisTerraTrueColorCR"}, "2018-03-30");
map.add_layer(nasa_layer);
```

(continues on next page)

(continued from previous page)

```
auto wms = xlf::wms_layer::initialize()
    .url("https://demo.boundlessgeo.com/geoserver/ows?")
    .layers("nasa:bluemarble")
    .name("nasa:bluemarble")
    .finalize();
map.add_layer(wms);

map.add_control(xlf::layers_control());

map
```

1.25 Split Map Control

1.25.1 Example

```
#include "xleaflet/xmap.hpp"
#include "xleaflet/xbasemaps.hpp"
#include "xleaflet/xsplit_map_control.hpp"

auto map = xlf::map::initialize()
    .center({42.6824, 365.581})
    .zoom(5)
    .finalize();

auto right_layer = xlf::basemap({"NASAGIBS", "ModisTerraTrueColorCR"}, "2017-11-11");
auto left_layer = xlf::basemap({"NASAGIBS", "ModisAquaBands721CR"}, "2017-11-11");

auto control = xlf::split_map_control::initialize()
    .left_layer(left_layer)
    .right_layer(right_layer)
    .finalize();
map.add_control(control);

map
```

1.25.2 Attributes

Attribute	Type	Default Value	Doc
left_layer	`xlf::layer`		Left layer
right_layer	`xlf::layer`		Right layer

1.26 Measure Control

1.26.1 Example

```
#include <iostream>
#include <string>

#include "xleaflet/xmap.hpp"
#include "xleaflet/xmeasure_control.hpp"
#include "xleaflet/xbasemaps.hpp"

auto water_color = xlf::basemap({"Stamen", "Watercolor"});

auto map = xlf::map::initialize()
    .layers({water_color})
    .center({50, 354})
    .zoom(5)
    .finalize();

auto measure_control = xlf::measure_control::initialize()
    .finalize();

map.add_control(measure_control);

map
```

1.26.2 Attributes

Attribute	Default Value	Doc
position	"topright"	Position of the control on the Map, possible values are topleft, topright, bottomleft or bottomright
primary_length_unit	"feet"	Primary length unit, possible values are feet, meters, miles, kilometers or any user defined length unit
secondary_length_unit	None	Secondary length unit, possible values are None, feet, meters, miles, kilometers or any user defined length unit
primary_area_unit	"acres"	Primary area unit, possible values are acres, hectares, sqfeet, sqmeters, sqmiles or any user defined area unit
secondary_area_unit	None	Secondary area unit, possible values are None, acres, hectares, sqfeet, sqmeters, sqmiles or any user defined area unit
active_color	"#ABE67E"	Color of the currently drawn area
completed_color	"#C8F2BE"	Color of the completed areas
popup_options	{"className": "leaflet-measure-resultpopup", "autoPanPadding": [10, 10]}	
capture_z_index	10000	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.

1.27 Widget control

1.27.1 Example

```
#include "xleaflet/xmap.hpp"
#include "xleaflet/xbasemaps.hpp"
#include "xwidgets/xslider.hpp"
#include "xwidgets/xnumeral.hpp"
#include "xwidgets/xlink.hpp"
#include "xleaflet/xwidget_control.hpp"

std::array<double, 2> center = {52.204793, 360.121558};

auto map = xlf::map::initialize()
    .center(center)
    .zoom(4)
    .close_popup_on_click(false)
    .finalize();
map.display();

auto button1 = xw::slider<double>::initialize()
    .min(1.0)
    .max(9.0)
    .value(4.0)
    .orientation("horizontal")
    .finalize();

auto popup2 = xlf::widget_control::initialize()
    .widget(button1)
    .position("bottomright")
    .finalize();
map.add_control(popup2);

xw::numeral<double> numeral;
auto popup4 = xlf::widget_control::initialize()
    .widget(numeral)
    .position("topright")
    .finalize();
map.add_control(popup4);


auto l = xw::link(numeral, "value", map, "zoom");
auto m = xw::link(button1, "value", map, "zoom");
<div style ="height:30px;"> </div>
```

1.27.2 Attributes

Attribute	Type	Default Value	Doc
widget	`xwidget`		Content of the widget
position	`string`		Position of the widget

1.28 Draw Control

The `draw_control` allows one to draw shapes on the map such as `rectangle` `circle` or lines.

```
#include "xleaflet/xmap.hpp"
#include "xleaflet/xdraw_control.hpp"
#include "xleaflet/xbasemaps.hpp"

auto water_color = xlf::basemap({"Stamen", "Watercolor"});

auto map = xlf::map::initialize()
    .layers({water_color})
    .center({50, 354})
    .zoom(5)
    .finalize();

    xeus::xjson polyline_options = {
    {"shapeOptions", {
        {"color", "#6bc2e5"},
        {"weight", 8},
        {"opacity", 1.0}
    }}
};

// Set some options for draw control
xeus::xjson polygon_options = {
    {"shapeOptions", {
        {"fillColor", "#6be5c3"},
        {"color", "#6be5c3"},
        {"fillOpacity", 1.0}
    }},
    {"drawError", {
        {"color", "#dd253b"},
        {"message", "Oups!"}
    }},
    {"allowIntersection", false}
};

xeus::xjson circle_options = {
    {"shapeOptions", {
        {"fillColor", "#efed69"},
        {"fillOpacity", 1.0},
        {"color", "#efed69"}
    }}
};

xeus::xjson rectangle_options = {
    {"shapeOptions", {
        {"fillColor", "#fca45d"},
        {"fillOpacity", 1.0},
        {"color", "#fca45d"}
    }}
};

auto draw_control = xlf::draw_control::initialize()
    .polyline(polyline_options)
    .polygon(polygon_options)
```

(continues on next page)

(continued from previous page)

```
    .circle(circle_options)
    .rectangle(rectangle_options)
    .finalize();
map.add_control(draw_control);

map
```

1.29 Releasing xleaflet

1.29.1 Releasing a new version

From the master branch of xleaflet

- Make sure that you are in sync with the master branch of the upstream remote.
- In file `xleaflet_config.hpp`, set the macros for `XLEAFLET_VERSION_MAJOR`, `XLEAFLET_VERSION_MINOR` and `XLEAFLET_VERSION_PATCH` to the desired values.
- Update the readme file w.r.t. dependencies on xleaflet
- Stage the changes (`git add`), commit the changes (`git commit`) and add a tag of the form `Major.minor.patch`. It is important to not add any other content to the tag name.
- Push the new commit and tag to the main repository. (`git push`, and `git push --tags`)
- Release the new version on conda
- Update the stable branch to point to the new tag