

# Geography 485L/585L - Internet Mapping

Karl Benedict

Spring 2018

## Week 11 - Module 2b - OpenLayers Javascript Framework

### Overview

- More detailed Map Object Options
- More detailed Layer Object Options
- Additional Map Layer Types - With Examples

### Map Object Options

- Map Object Options [API Reference](#)
- View Object Options [API Reference](#)
- Layer Object Options
  - `ol.layer.Tile` [API Reference](#)
  - `ol.layer.Image` [API Reference](#)
  - `ol.layer.Vector` [API Reference](#)
  - `ol.layer.VectorTile` [API Reference](#)
  - `ol.layer.Heatmap` [API Reference](#)

A variety of strategies for constructing a new `OpenLayers.Map` object

```
1 // create a map with minimum required elements and default
2 // options in an element with the id "map1"
3 var myMap = new ol.Map({
4     target: 'map1',
5     // a map without layers can be defined and in that case a map with no layers
6     // will be rendered
7     layers: [
8         new ol.layer.Tile({
9             source: new ol.source.OSM()
10        })
11    ],
12    view: new ol.View({
13        center: ol.proj.fromLonLat([-106.624083, 35.08427]),
14        zoom: 18
15    })
16 });
```

```

17
18
19 // create a map with options specified in a separate 'options' variable and
20 // included by reference in the code to create the new map object
21 var options = {
22     // required options
23     target:'map2',
24     layers: ...,
25     view: ...,
26
27     // optional options - only include those that you need
28     controls: ...,
29     pixelRatio: ...,
30     interactions: ...,
31     keyboardEventTarget: ...,
32     loadTilesWhileAnimating: ...,
33     loadTilesWhileInteracting: ...,
34     logo: ...,
35     overlays: ...,
36     renderer: ...
37 };
38 var map = new ol.Map(options);
39
40 // map with non-default options - same as above but with a single argument
41 var map = new ol.Map({
42     // required options
43     target:'map2',
44     layers: ...,
45     view: ...,
46
47     // optional options - only include those that you need
48     controls: ...,
49     pixelRatio: ...,
50     interactions: ...,
51     keyboardEventTarget: ...,
52     loadTilesWhileAnimating: ...,
53     loadTilesWhileInteracting: ...,
54     logo: ...,
55     overlays: ...,
56     renderer: ...
57 });
58
59 // the following commands can be executed to add, set or remove the layers in a map
60 // after a map object has been created
61
62 map.addLayer(layer)
63 map.removeLayer(layer)
64 map.setLayerGroup(layerGroup)
65
66 // the view of a layer can be created or modified after the map object has been
67 // created by using the following command
68
69 map.setView()
70

```

```

71 // the target DOM object for the map object can be set or changed using
72 // the following command
73
74 map.setTarget()

```

## Layer Object Options

Layer Types and a subset of sources for each type

- `ol.layer.Image` - a single map image is rendered for this layer type
  - `ol.source.ImageMapGuide` - [API](#) source is a [MapGuide](#) server hosting data of interest.
  - `ol.source.ImageStatic` - [API](#) source renders a specified static image file within a specified extent within the map.
  - `ol.source.ImageWMS` - [API](#) source retrieves a single map image from the specified OGC Web Map Service (WMS).
  - `ol.source.ImageArcGISRest` - [API](#) source retrieves a single map image from the specified ArcGIS REST service.
  - `ol.source.ImageCanvas` - [API](#) source places the result of a `canvasFunction` within the defined layer element.
  - `ol.source.Raster` - [API](#) source places the result of a raster operation within the defined layer element.
- `ol.layer.Tile` - map images in a tiled grid are rendered for this layer type
  - `ol.source.BingMaps` - [API](#) source is Bing map or image service
  - `ol.source.Stamen` - [API](#) source is one of the layer types supported by the Stamen image service
  - `ol.source.CartoDB` - [API](#) source is the [CartoDB](#) API
  - `ol.source.TileArcGISRest` - [API](#) source is an ArcGIS REST map or image service
  - `ol.source.TileWMS` - [API](#) source is an OGC Web Map Service (WMS)
  - `ol.source.TileJSON` - [API](#) source is the TileJSON format
  - `ol.source.WMTS` - [API](#) source is an OGC Web Map Tile Service ([WMTS](#))
  - `ol.source.XYZ` - [API](#) source is a collection of tiles that are referenced by URLs that follow a template for specifying x-, y-, and z-locations.
  - `ol.source.Zoomify` - [API](#) source is a collection of tiles in the [Zoomify](#) format.
- `ol.layer.VectorTile` - map content is delivered vector data that has been divided into a tile grid and cannot be edited
  - `ol.source.VectorTile` - [API](#) source delivers vector data tiles for rendering in the client
- `ol.layer.Vector` - map content is delivered as vector data that is rendered by the client and may be edited within the client
  - `ol.source.Vector` - [API](#) the source for vector feature(s) that constitute a vector layer. The individual features are [ol.Feature](#) objects that consist of at least one [geometry](#), or a [collection](#) of geometries and any additional attributes that are associated with each feature.

Common Pattern of [Layer Object](#) Creation (varies some depending upon the specific layer type)

```

1 var layer = new ol.layer.***({
2   source: new ol.source.***({
3     ...
4   }),
5   other options ...
6 })

```

## Additional Map and Layer Object Functions & Events

Both Map and Layer Objects have a number of associated functions as well

- Retrieving object properties programmatically with **Get** functions.
- Modifying existing object properties with **Set** functions
- Map destruction, and reconfiguration
- Linkage of object events with Javascript functions

## WMS Layer Configuration

Some key issues to be aware of when using the two WMS supporting layers (`ol.layer.Tile`, and `ol.layer.Image`) and their associated WMS sources (`ol.source.TileWMS` and `ol.source.ImageWMS` respectively) include:

- The *projection* of the map object must be supported by the included WMS service (review the WMS GetCapabilities response to see what projections are supported by the service). If you don't specify a **projection** parameter as part of the map object's **view** property a default *Web Mercator* (EPSG:3857) projection is used for the map. Information about how to define and set map projections in OpenLayers is found [here](#)
- The *layers* parameter as part of the **params** option must be provided as part of the server-related property list (the layer names may also be found in the GetCapabilities response)
- Other WMS parameters (again as part of the **params** option) may be provided as well to “adjust” the request automatically generated by OpenLayers
- Use of a tiled WMS may produce unwanted repetition of labels included in the WMS. If that is the case you can use a single-image `ol.layer.Image` layer type to allow the WMS server to handle the distribution of layers across the entire map image instead of including them in each individual map image.

### Sample WMS Layer Object Creation

```
1 // OpenLayers_03_wms.js
2
3 // OpenLayers_03_wms.js
4
5 ///////////////////////////////////////////////////////////////////
6 // define layer objects
7
8 var basemap_tiled = new ol.layer.Tile({
9     source: new ol.source.TileWMS({
10         url: 'http://internetmapping.net:8080/geoserver/wms?',
11         params: {
12             LAYERS: 'kbene:BMNG_west',
13             FORMAT: 'image/png',
14             TRANSPARENT: true
15         },
16         attributions: [
17             new ol.Attribution({
18                 html: 'Data provided by the GEOG x85 GeoServer.'
19             })
20         ]
21     })
22 })
```

```

22  })
23
24  var basemap_single = new ol.layer.Image({
25      source: new ol.source.ImageWMS({
26          url: 'http://internetmapping.net:8080/geoserver/wms?',
27          params: {
28              LAYERS: 'kbene:BMNG_west',
29              FORMAT: 'image/png',
30              TRANSPARENT: true
31          },
32          attributions: [
33              new ol.Attribution({
34                  html: 'Data provided by the GEOG x85 GeoServer.'
35              })
36          ]
37      })
38  })
39
40  var states_single = new ol.layer.Image({
41      source: new ol.source.ImageWMS({
42          attributions: new ol.Attribution({
43              html: 'Data provided by the GEOG x85 GeoServer.'
44          }),
45          params: {'LAYERS': 'kbene:statep010'},
46          url: 'http://internetmapping.net:8080/geoserver/wms?',
47          serverType: 'geoserver'
48      })
49  })
50
51  var states_tiled = new ol.layer.Tile({
52      source: new ol.source.TileWMS({
53          attributions: new ol.Attribution({
54              html: 'State Boundary Restructured - USGS, National Atlas Release 5-14-12'
55          }),
56          params: {'LAYERS': 'kbene:statep010'},
57          url: 'http://internetmapping.net:8080/geoserver/wms?',
58          serverType: 'geoserver'
59      })
60  })
61
62
63
64  //////////////////////////////////////
65  // create our base map objects
66  var singleMap = new ol.Map({
67      target: 'map_image',
68      layers: [basemap_single, states_single], //[basemap_single, states_single]
69      view: new ol.View({
70          center: ol.proj.fromLonLat([-98.58, 39.83]), // the approximate geographic center of the continent
71          zoom: 3,
72          projection: 'EPSG:3857'
73      })
74  });
75

```

76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98

```
var tiledMap = new ol.Map({
  target: 'map_tiled',
  layers: [basemap_tiled,states_tiled], //[basemap_tiled,states_tiled]
  view: new ol.View({
    center: ol.proj.fromLonLat([-98.58,39.83]), // the approximate geographic center of the continent
    zoom: 3,
    projection: 'EPSG:3857'
  })
});

var mixedMap = new ol.Map({
  target: 'map_mixed',
  layers: [basemap_tiled,states_single], //[basemap_tiled,states_single]
  view: new ol.View({
    center: ol.proj.fromLonLat([-98.58,39.83]), // the approximate geographic center of the continent
    zoom: 3,
    projection: 'EPSG:3857'
  })
});

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

Example: [HTML](#), [Javascript](#)

## Vector Layer Configuration

Vector layers support

- External Data in a Variety of supported [formats](#) for both *reading* and *writing* (just a sample): [GML 2](#) and [GML 3](#), [GPX](#), [GeoJSON](#), [KML](#), [WFS](#), [WKT](#), [Open Streetmap XML](#)
- Directly encoded [geometries](#): Circle, Geometry, GeometryCollection, LinearRing, LineString, MultiLineString, MultiPoint, MultiPolygon, Point, Polygon, SimpleGeometry
- User created features, including support for interactive editing of features
- [Styling](#) of Vector features

Sample Point Feature Object creation

```
1 var classroomCoord = [-106.624073,35.084280]
2 var officeCoord = [-106.624899,35.084506]
3
4 var classroomPoint = new ol.geom.Point(ol.proj.fromLonLat(classroomCoord, projection));
5 var officePoint = new ol.geom.Point(ol.proj.fromLonLat(officeCoord, projection));
```

Sample KML Layer Object creation with style

```
1 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
2 // define some styles
3
4 var block_color = [0,255,0,.1]
5 var block_line_color = [0,255,0,1]
```

```

6  var county_color = [124,124,255,.25]
7  var county_line_color = [124,124,255,1]
8
9  var county_style = new ol.style.Style({
10     fill: new ol.style.Fill({
11         color: county_color
12     }),
13     stroke: new ol.style.Stroke({
14         color: county_line_color,
15         width: 2
16     }),
17 });
18
19 var block_style = new ol.style.Style({
20     fill: new ol.style.Fill({
21         color: block_color
22     }),
23     stroke: new ol.style.Stroke({
24         color: block_line_color,
25         width: 1
26     }),
27 });
28 ///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
29
30 // unstyled layers
31
32 var blocks_kml = new ol.layer.Vector({
33     source: new ol.source.Vector({
34         url: 'https://s3.amazonaws.com/kkb-web/data/tl_2010_35001_tabbblock10.kml',
35         projection: projection,
36         format: new ol.format.KML()
37     })
38 })
39
40 var counties_kml = new ol.layer.Vector({
41     source: new ol.source.Vector({
42         url: 'https://s3.amazonaws.com/kkb-web/data/2007fe_35_county00.kml',
43         projection: projection,
44         format: new ol.format.KML()
45     })
46 })
47
48
49 // styled layers
50
51 var counties_kml_styled = new ol.layer.Vector({
52     source: new ol.source.Vector({
53         url: 'https://s3.amazonaws.com/kkb-web/data/2007fe_35_county00.kml',
54         projection: projection,
55         format: new ol.format.KML({
56             extractStyles:false
57         })
58     }),
59     style: county_style

```

```
60  })
61
62  var blocks_kml_styled = new ol.layer.Vector({
63    source: new ol.source.Vector({
64      url: 'https://s3.amazonaws.com/kkb-web/data/tl_2010_35001_tabblock10.kml',
65      projection: projection,
66      format: new ol.format.KML({
67        extractStyles:false
68      })
69    }),
70    style: block_style
71  })
72
73
```

Example: [HTML](#), [Javascript](#)

---

This work by Karl Benedict is licensed under a Creative Commons Attribution-ShareAlike 4.0 International License.