

# 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](#)

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 map = 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.MapQuest({layer: 'osm'})
10         })
11     ], // end layers
12     view: new ol.View({
13         center: [0, 0],
14         zoom: 1
15     }), //end view
16 });
17
18 // create a map with options specified in a separate 'options' variable and
19 // included by reference in the code to create the new map object
20 var options = {
21     // required options
22     target:'map2',
23     layers: ...,
24     view: ...,
25
26     // optional options - only include those that you need
27     controls: ...,
28     pixelRatio: ...,
29     interactions: ...,
30     keyboardEventTarget: ...,
31     loadTilesWhileAnimating: ...,
```

```

32     loadTilesWhileInteracting: ...,
33     logo: ...,
34     overlays: ...,
35     renderer: ...
36 };
37 var map = new ol.Map(options);
38
39 // map with non-default options - same as above but with a single argument
40 var map = new ol.Map({
41     // required options
42     target: 'map2',
43     layers: ...,
44     view: ...,
45
46     // optional options - only include those that you need
47     controls: ...,
48     pixelRatio: ...,
49     interactions: ...,
50     keyboardEventTarget: ...,
51     loadTilesWhileAnimating: ...,
52     loadTilesWhileInteracting: ...,
53     logo: ...,
54     overlays: ...,
55     renderer: ...
56 });
57
58 // the following commands can be executed to add, set or remove the layers in a map
59 // after a map object has been created
60
61 map.addLayer(layer)
62 map.removeLayer(layer)
63 map.setLayerGroup(layerGroup)
64
65 // the view of a layer can be created or modified after the map object has been
66 // created by using the following command
67
68 map.setView()
69
70 // the target DOM object for the map object can be set or changed using
71 // the following command
72
73 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.layer.Tile` - map images in a tiled grid are rendered for this layer type
  - `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.WMTS` - [API](#) source is an OGC Web Map Tile Service ([WMTS](#))
- `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 ([example](#))
- `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 var basemap_single = new ol.layer.Image({
2   source: new ol.source.ImageWMS({
3     attributions: new ol.Attribution({
4       html: 'Blue Marble Next Generation:' +
5         'R. Stockli, E. Vermote, N. Saleous, R. Simmon and D. Herring (2005). The Blue Marble Next G
6     }),
7     params: {'LAYERS': 'global:BMNG_west'},
8     url: 'http://mapper.karlbenedict.com:8080/geoserver/global/wms?',
9     serverType: 'geoserver'
10  })
11 })
12
13 var states_single = new ol.layer.Image({
14   source: new ol.source.ImageWMS({
15     attributions: new ol.Attribution({
16       html: 'State Boundary Restructured - USGS, National Atlas Release 5-14-12'
17     }),
18     params: {'LAYERS': 'global:statep010'},
19     url: 'http://mapper.karlbenedict.com:8080/geoserver/global/wms?',
20     serverType: 'geoserver'
21  })
22 })
23
24 var singleMap = new ol.Map({
25   target: 'map_image',
26   layers: [basemap_single, states_single],
27   view: new ol.View({
28     center: ol.proj.fromLonLat([-98.58, 39.83]), // the approximate geographic center of the continen
29     zoom: 3,
30     projection: 'EPSG:3857'
31   })
32 });
33
34
```

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](#), [GPX](#), [GeoJSON](#), [JSON](#), [KML](#), [WFS](#), [WKT](#), [Open Streetmap](#)
- Directly encoded [geometries](#): Circle, Geometry, GeometryCollection, LinearRing, LineString, Multi-LineString, 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(classroomCoord);
5  var officePoint = new ol.geom.Point(officeCoord);

```

Sample KML Layer Object creation with style

```

1  var blocks_kml = new ol.layer.Vector({
2      source: new ol.source.Vector({
3          url: 'https://s3.amazonaws.com/kkb-web/data/tl_2010_35001_tabbblock10.kml',
4          projection: projection,
5          format: new ol.format.KML()
6      })
7  })
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_color,
15         width: 1
16     }),
17 });
18
19 var counties_kml_styled = new ol.layer.Vector({
20     source: new ol.source.Vector({
21         url: 'https://s3.amazonaws.com/kkb-web/data/2007fe_35_county00.kml',
22         projection: projection,
23         format: new ol.format.KML({'extractStyles':false}),
24         style: county_style
25     })
26 })
27

```

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

---

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