**OpenLayers 3**

In OpenLayers, a map is a collection of layers and various interactions and controls for dealing with user interaction. A map is generated with three basic ingredients: [markup](http://workshops.boundlessgeo.com/openlayers3/basics/dissect.html" \l "config-dissect-markup), [style declarations](http://workshops.boundlessgeo.com/openlayers3/basics/dissect.html" \l "config-dissect-style), and [initialization code](http://workshops.boundlessgeo.com/openlayers3/basics/dissect.html" \l "config-dissect-code).

<!doctype html>

**<html** lang="en"**>**

**<head>**

**<link** rel="stylesheet" href="ol3/ol.css" type="text/css"**>**

**<style>**

#map {

**height**: 256px;

**width**: 512px;

}

**</style>**

**<title>**OpenLayers 3 example**</title>**

**<script** src="ol3/ol.js" type="text/javascript"**></script>**

**</head>**

**<body>**

**<h1>**My Map**</h1>**

**<div** id="map"**></div>**

**<script** type="text/javascript"**>**

**var** map = **new** ol.Map({ // Map Initialization

target: 'map',

layers: [ // Layer Creation

**new** ol.layer.Tile({

title: "Global Imagery",

source: **new** ol.source.TileWMS({

url: 'http://maps.opengeo.org/geowebcache/service/wms',

params: {LAYERS: 'bluemarble', VERSION: '1.1.1'}

})

}),

**new** ol.layer.Tile({

source: **new** ol.source.BingMaps({ // Proprietary Raster layer [Bing]

imagerySet: 'Road',

key: 'Ak-dzM4wZjSqTlzveKz5u0d4IQ4bRzVI309GxmkgSVr1ewS6iPSrOvOKhA-CJlm3'

})

}),  
 **new** ol.layer.Vector({ // Adding a Vector Layer

title: 'Earthquakes',

source: **new** ol.source.GeoJSON({ // Layer requesting a set of features stored in GeoJSON

url: 'data/layers/7day-M2.5.json'

}),

style: **new** ol.style.Style({

image: **new** ol.style.Circle({

radius: 3,

fill: **new** ol.style.Fill({color: 'white'})

})

})

})

],

view: **new** ol.View({ // Defining View

projection: 'EPSG:4326', // Defining Projection

center: [0, 0],

zoom: 0,

maxResolution: 0.703125

}),

controls: ol.control.defaults().extend([

**new** ol.control.ScaleLine() // Adding a ScaleLine

]),

interactions: ol.interaction.defaults().extend([ // Advantages of serving vector

data is that user can interact with data.

ol.interaction.Interaction is responsible for handling user

interaction.

**new** ol.interaction.Select({

style: **new** ol.style.Style({

image: **new** ol.style.Circle({

radius: 5,

fill: **new** ol.style.Fill({

color: '#FF0000'

}),

stroke: **new** ol.style.Stroke({

color: '#000000'

})

})

})

})

]),

});

**</script>**

**</body>**

**</html>**

<!DOCTYPE html>

<html>

<head>

<meta http-equiv="content-type" content="text/html; charset=UTF-8">

<meta name="robots" content="noindex, nofollow">

<meta name="googlebot" content="noindex, nofollow">

<script type="text/javascript" src="//code.jquery.com/jquery-1.11.0.js"></script>

<link rel="stylesheet" type="text/css" href="/css/result-light.css">

<link rel="stylesheet" type="text/css" href="https://cdnjs.cloudflare.com/ajax/libs/ol3/3.6.0/ol.css">

<script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/ol3/3.6.0/ol.js"></script>

<script type="text/javascript" src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.4/js/bootstrap.min.js"></script>

<link rel="stylesheet" type="text/css" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.4/css/bootstrap.min.css">

<style type="text/css">

</style>

<title>Naveen</title>

<script type='text/javascript'>//<![CDATA[

$(window).load(function(){

var inputvalue = 'default description';

// listen to changes on the input form via jquery

$('#input').change(function () {

newinputvalue = $('#input').val();

inputvalue = newinputvalue;

return inputvalue;

});

var raster = new ol.layer.Tile({

source: new ol.source.MapQuest({

layer: 'sat'

})

});

function personalmarker(text) {

var fill = new ol.style.Fill({

color: 'rgba(255, 255, 255, 0.5)'

});

var stroke = new ol.style.Stroke({

color: '#ffcc33',

width: 2

});

var image = new ol.style.Circle({

radius: 7,

fill: new ol.style.Fill({

color: '#ffcc33'

})

});

var textStroke = new ol.style.Stroke({

color: '#fff',

width: 5

});

var textFill = new ol.style.Fill({

color: '#111'

});

var higlighttext = new ol.style.Text({

font: '10px Calibri,sans-serif',

text: text,

fill: textFill,

stroke: textStroke

});

return new ol.style.Style({

text: higlighttext,

fill: fill,

stroke: stroke,

image: image

});

}

var source = new ol.source.Vector({

wrapX: false

});

var vector = new ol.layer.Vector({

source: source,

style: personalmarker(inputvalue)

});

var map = new ol.Map({

layers: [raster, vector],

target: 'map',

view: new ol.View({

center: [-11000000, 4600000],

zoom: 4

})

});

var typeSelect = document.getElementById('type');

var draw; // global so we can remove it later

function addInteraction() {

var value = typeSelect.value;

if (value !== 'None') {

var geometryFunction, maxPoints;

if (value === 'Square') {

value = 'Circle';

geometryFunction = ol.interaction.Draw.createRegularPolygon(4);

} else if (value === 'Box') {

value = 'LineString';

maxPoints = 2;

geometryFunction = function (coordinates, geometry) {

if (!geometry) {

geometry = new ol.geom.Polygon(null);

}

var start = coordinates[0];

var end = coordinates[1];

geometry.setCoordinates([

[start, [start[0], end[1]], end, [end[0], start[1]], start]

]);

return geometry;

};

}

draw = new ol.interaction.Draw({

source: source,

type: /\*\* @type {ol.geom.GeometryType} \*/

(value),

geometryFunction: geometryFunction,

maxPoints: maxPoints

});

map.addInteraction(draw);

**// Retrieving the coordinates of features geometry when finished drawing using '*drawend* ' operation.**

draw.on('drawend', function (e2) {

var feature = e2.feature;

var geometry = feature.getGeometry();

console.log(geometry);

//depending on the type of geometry drawn you may get first and last

//coordinate. From your description I guess you draw a linestring

//you may clarify that using geometry.getType()

//so for ol.geom.LineString do as follows. According to the

//documentation this should work for any type of geometries

var startCoord = geometry.getFirstCoordinate();

var endCoord = geometry.getLastCoordinate();

//If you are not sure what the type is, or if you face any problems

//with getFirstCoordinate, getLastCoordinate

//you may go for a more general technique

var coordinates = geometry.getCoordinates();

//and then parse the coordinates object to get first and last

var startCoord = coordinates[0];

var endCoord = coordinates[coordinates.length-1];

console.log(coordinates);

});

}

}

/\*\*

\* Let user change the geometry type.

\* @param {Event} e Change event.

\*/

typeSelect.onchange = function (e) {

map.removeInteraction(draw);

addInteraction();

};

addInteraction();

});//]]>

</script>

</head>

<body>

<div class="row-fluid">

<div class="span12">

<div id="map" class="map"></div>

<div id="userinput">

<input type="text" id="input" />

</div>

<form class="form-inline">

<label>Geometry type &nbsp;</label>

<select id="type">

<option value="None">None</option>

<option value="Point">Point</option>

<option value="LineString">LineString</option>

<option value="Polygon">Polygon</option>

<option value="Circle">Circle</option>

<option value="Square">Square</option>

<option value="Box">Box</option>

</select>

</form>

</div>

</div>

<script>

// tell the embed parent frame the height of the content

if (window.parent && window.parent.parent){

window.parent.parent.postMessage(["resultsFrame", {

height: document.body.getBoundingClientRect().height,

slug: "None"

}], "\*")

}

</script>

</body>

</html>

**<!doctype html>**

**<html** lang="en"**>**

**<head>**

**<link** rel="stylesheet" href="ol3/ol.css" type="text/css"**>**

**<style>**

#map {

**height**: 256px;

**width**: 512px;

}

**</style>**

**<script** src="ol3/ol.js" type="text/javascript"**></script>**

**<title>**OpenLayers 3 example**</title>**

**</head>**

**<body>**

**<h1>**My Map**</h1>**

**<div** id="map"**></div>**

**<script** type="text/javascript"**>**

**var** source = **new** ol.source.GeoJSON({

url: 'data/layers/7day-M2.5.json'

});

**var** style = **new** ol.style.Style({

image: **new** ol.style.Circle({

radius: 7,

fill: **new** ol.style.Fill({

color: [0, 153, 255, 1]

}),

stroke: **new** ol.style.Stroke({

color: [255, 255, 255, 0.75],

width: 1.5

})

}),

zIndex: 100000

});

**var** select = **new** ol.interaction.Select({style: style});

**var** modify = **new** ol.interaction.Modify({

features: select.getFeatures()

});

**var** map = **new** ol.Map({

interactions: ol.interaction.defaults().extend([select, modify]),

target: 'map',

layers: [

**new** ol.layer.Tile({

title: "Global Imagery",

source: **new** ol.source.TileWMS({

url: 'http://maps.opengeo.org/geowebcache/service/wms',

params: {LAYERS: 'bluemarble', VERSION: '1.1.1'}

})

}),

**new** ol.layer.Vector({

title: 'Earthquakes',

source: source,

style: **new** ol.style.Style({

image: **new** ol.style.Circle({

radius: 5,

fill: **new** ol.style.Fill({

color: '#0000FF'

}),

stroke: **new** ol.style.Stroke({

color: '#000000'

})

})

})

})

],

view: **new** ol.View({

projection: 'EPSG:4326',

center: [0, 0],

zoom: 1

})

});

**</script>**

**</body>**

**</html>**

Modifying features works by using an ol.interaction.Select in combination with an ol.interaction.Modify. They share a common collection (ol.Collection) of features. Features selected with the ol.interaction.Select become candidates for modifications with the ol.interaction.Modify.

**Styling based on features attribute**

**Adding Fill and Stroke styling**

style: (**function**() {

**var** defaultStyle = [**new** ol.style.Style({

fill: **new** ol.style.Fill({color: 'navy'}),

stroke: **new** ol.style.Stroke({color: 'black', width: 1})

})];

**var** ruleStyle = [**new** ol.style.Style({

fill: **new** ol.style.Fill({color: 'olive'}),

stroke: **new** ol.style.Stroke({color: 'black', width: 1})

})];

**return** **function**(feature, resolution) {

**if** (feature.get('shape\_area') < 3000) {

**return** ruleStyle;

} **else** {

**return** defaultStyle;

}

};

})()

**Adding text styling**

style: (**function**() {

**var** stroke = **new** ol.style.Stroke({

color: 'black'

});

**var** textStroke = **new** ol.style.Stroke({

color: '#fff',

width: 3

});

**var** textFill = **new** ol.style.Fill({

color: '#000'

});

**return** **function**(feature, resolution) {

**return** [**new** ol.style.Style({

stroke: stroke,

text: **new** ol.style.Text({

font: '12px Calibri,sans-serif',

text: feature.get('key'),

fill: textFill,

stroke: textStroke

})

})];

};

})()

**Geoserver**

The OpenGeo Suite is a complete web-based geospatial software stack. In this package, the applications contained are:

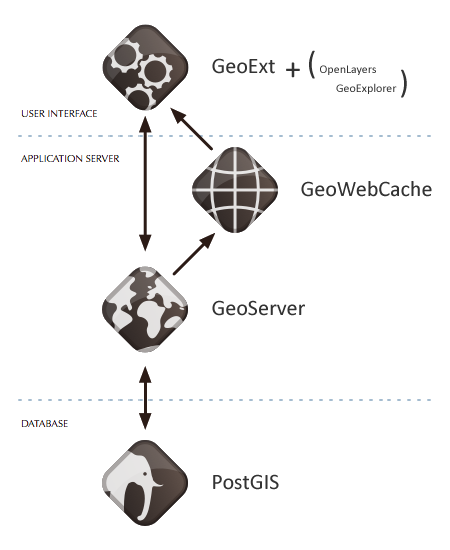
* **PostGIS** - A spatially enabled object-relational database.
* **GeoServer**- A software server for loading ,sharing and editing geospatial data. Designed for

interoperability, it publishes data from any major spatial data source using open

standards.

* **GeoWebCache** - A tile cache server that accelerates the serving of maps (built into GeoServer).
* **GeoExplorer** - A web application for composing, styling, and publishing maps.

GeoExplorer is based on the [**GeoExt**](http://geoext.org/) framework and contains code from [**OpenLayers**](http://openlayers.org/).



## **Starting and stopping services**

OpenGeo Suite has been installed and is now running by default. There are two system services related to OpenGeo Suite:

* **OpenGeo Jetty** - Controls GeoServer and other web applications
* **OpenGeo Postgres** - Controls PostgreSQL/PostGIS database

## **Web mapping servers**

A web mapping server is a specialized subset of web server. Like a web server, requests are sent to the server which are interpreted and responded. But the requests and responses are designed specifically toward the transfer of geographic information.

A web mapping server may use HTTP, but employ specialized protocols, such as **Web Map Service (WMS)**, **Web Feature Service (WFS)**. These protocols are designed for the transferring geographic information to and from the server, whether it be raw feature data, geographic attributes, or map images.

# **Web Map Service (WMS)**

**The Web Map Service (WMS)** is a standard protocol for serving georeferenced map images generated by a map server.

**WMS sample URL**

http://suite.opengeo.org/geoserver/wms?

service=WMS&

version=1.3.0&

request=GetMap&

layers=usa:states&

srs=EPSG:4326&

bbox=24.956,-124.731,49.372,-66.97&

format=image/png&

width=780&

**height=330**

Information requested :

* Server details (a WMS 1.3.0 request)
* Request type (WMS GetMap)
* Layer name (usa:states)
* Projection (EPSG:4326)
* Bounding box (in this case, latitude/longitude coordinates)
* Image properties (780x330 PNG)

# **Web Feature Service (WFS)**

WFS web service communicates geographic feature information which allows features to be queried, updated, created, or deleted by the client.

**WFS sample URL**

http://suite.opengeo.org/geoserver/wfs?

service=wfs&

version=1.1.0&

request=GetFeature&

typename=usa:states&

featureid=states.39

Information requested :

* Server details (WFS 1.1.0 request)
* Request type (GetFeature)
* Layer name (usa:states)
* Feature ID (states.39)

**Web Coverage Service (WCS)**

WCS web service is used to transfer "coverages", ie. objects covering a geographical area. Coverages can be a set of data points; a regular grid of points (or pixels); a set of segmented curves (eg. road paths).

**Web Processing Service (WPS)**

WPS web service is for the publishing of geospatial processes, algorithms, and calculations.

A **workspace** (sometimes referred to as a namespace) is the name for a container for grouping similar data together.

A **store** is the name for a container of geographic data (shapefile, database, or any other data source that GeoServer supports ).

A **layer** (sometimes known as a featuretype) is a collection of geospatial features (points, lines, polygons, raster ) or a coverage.

A **style** is a visualization directive for rendering geographic data .

**SLD (Styled Layer Descriptor)**

GeoServer uses the Styled Layer Descriptor (SLD) markup language to visualize geospatial data. SLD is an XML-based standard created by the Open Geospatial Consortium (OGC).

An SLD file contains the following hierarchical structure:

* Header
  + FeatureTypeStyles
    - Rules
      * Symbolizers

The header of the SLD contains metadata about XML namespaces, and is usually identical among different SLDs.

A **FeatureTypeStyle** is a group of styling rules.

A **Rule** is a single styling directive.

A **Symbolizer** is the actual style instruction. There are five types of symbolizers:

* PointSymbolizer
* LineSymbolizer
* PolygonSymbolizer
* RasterSymbolizer
* TextSymbolizer.

# GeoExplorer

Creating SLD files by hand can be a difficult and time-consuming process. Fortunately, there is a tool called *GeoExplorer* which is a graphical style editor. With GeoExplorer, you can create rules and symbolizers without ever needing to view SLD code.

# Filtering layers

# The cql\_filter parameter is a way to specify a predicate based on attribute values or spatial orientation. Let’s single out only countries that have populations of one hundred million or more.

&cql\_filter=POP\_EST > '100000000'

**Projection**

The method of representing the surface of a sphere on a plane.

**WCS**

Web Coverage Service. An OGC protocol that enables access to coverages, or raster geospatial data.

**WFS**

Web Feature Service. An OGC protocol that enables access to featuretypes, or vector geospatial data.

**WMS**

Web Map Service. An OGC protocol for requesting georeferenced map images.

**WPS**

Web Processing Service. An OGC protocol that enables the publishing of geospatial processes for geospatial analysis.