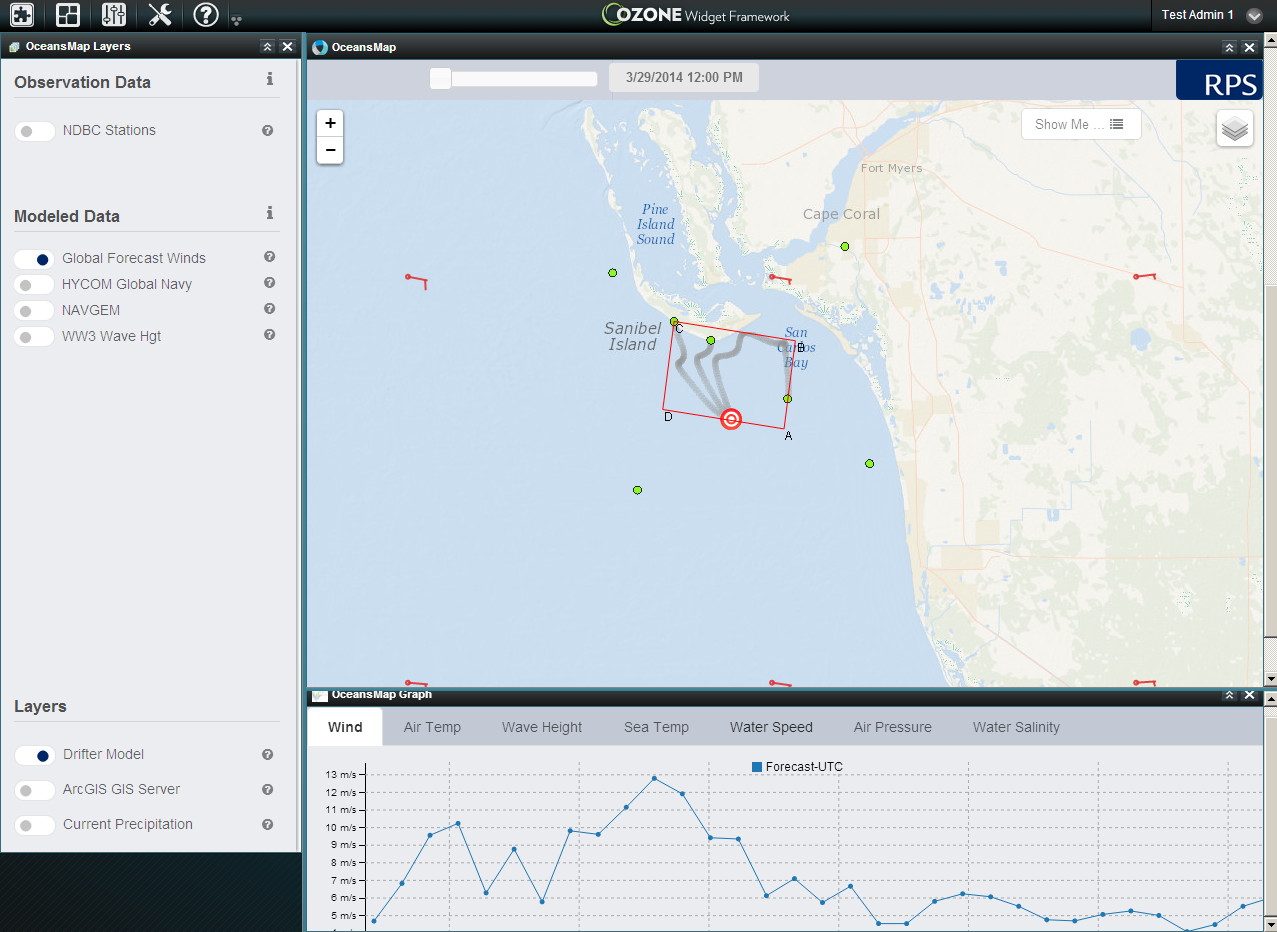
**Ozone Widget Framework OceansMap Dashboard**

**Technical Documentation**

****

****

**April, 2014**

**Application Overview**

The OceansMap application is a simple web application used to visualize oceanographic information. Consisting of a simple map panel, a Table of Contents window and a graphing chart, the Oceansmap website is a very simple Javascript webGIS application. Using the Ozone Widget Framework, the OceansMap pieces were configured to work with the communication and channel architecture to be used individually and collaboratively with other operational tools and widgets.

For this proof-of-concept example, the three window panels were abstracted and divided into three separate widgets:

* The **MAP**
* The **GRAPH**
* The Table of Contents (**TOC**)

Using the custom Dashboard tool within OWF, a three panelled dashboard is created with this current version. This is currently a raw example of what can be done with this framework. Please provide bugs or issues in the GitHub branch below.

**Setup and Access**

This is currently hosted on RPS ASA AGS01 server. Address and login credentials are here:

<https://gis.asascience.com:8443/owf>

**User**: testAdmin1 P**assword**: password

The application *sourcecode* is forked from <http://ozoneplatform.org/> and is located here:

<https://github.com/ssontag55/owf>

The application uses the following JS libraries:

<http://semantic-ui.com/> - for overall look and feel

<http://leafletjs.com> – for all Mapping related classes

<http://backbonejs.org/> - for some models and bindings

<http://nodejs.org/> - this really isn’t being used but can be leveraged for backend processing if needed.

The application uses the publishing and subscribing architecture from OWF. The widgets can be added to additional users but currently only added to testAdmin1. To edit or use individual widgets, they are located in the following code folder:

[project\_directory]\owf\OWF-bundle-7.0.1-GA\apache-tomcat-7.0.21\webapps\owf\examples\walkthrough\widgets

**Channels and Communication**

The following are a list of channels that are being broadcasted with each individual widget and what formats are being pushed to the OWF. As said before, these applications build off the included JS libraries so using the selected classes with the Leaflet Library specifically is recommended. Be sure to include the /apache-tomcat-7.0.21/webapps/owf/js-min/owf-widget-debug.js script within your widget.

* Channels are published or broadcasted as follows:

OWF.Eventing.publish("eventChannel", returnedObject);

* Channels are subscribed or listened to as follows:

OWF.Eventing.subscribe("graphData", function (sender, msg, channel) {

Do something here });

The OceansMap **MAP** Panel publishes an event on every Map Click:

Channel: leafletMapClick

Object returned:

Latlng, layerpoint in pixels coordinates. More info here:

<http://leafletjs.com/reference.html#mouse-event>

The OceansMap **MAP** Panel publishes/subscribes an event on when TimeSlider is changed/moved:

Channel: mapTimeChange

Object returned:

"3/30/2014 12:00:00 AM"

The OceansMap **MAP** Panel publishes this event when Feature/Marker is clicked:

The **TOC** Panel subscribes to this event when Feature/Marker is clicked:

Channel: leafletmarkerClick

Object returned:

Marker info with Lat/Long and attributes. More info here:

<http://leafletjs.com/reference.html#marker>

The **GRAPH** window subscribes to this event to clear the graph data:

Channel: clearGraphData

Object returned: None

The **GRAPH** window subscribes to this event to start progress bar while data is being process/queried:

Channel: graphQueryStart

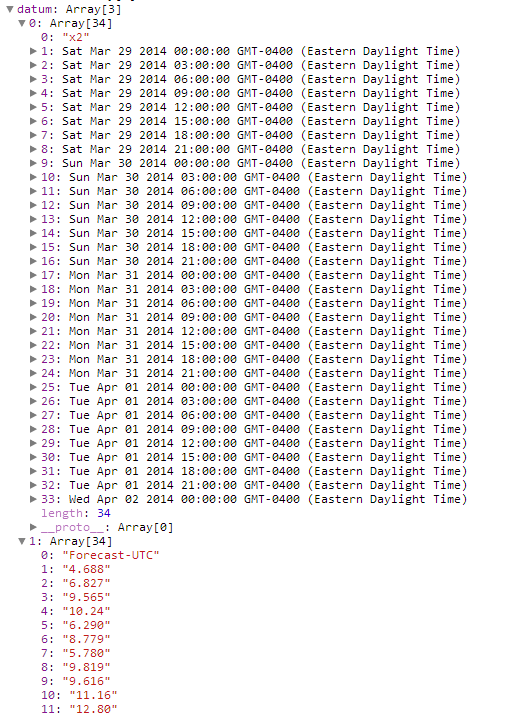
Object returned: None

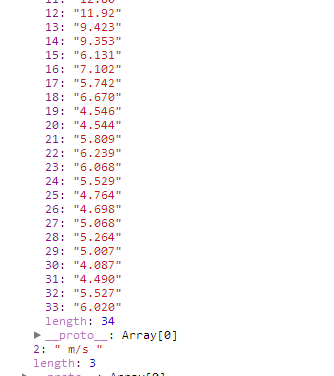
The **MAP** panelpublishes this event to graph Forecasted Data from EDS/WMS GetFeatureInfo:

The **GRAPH** window subscribes to this event and charts the JSON response as forecasted data:

Channel: graphData

Object returned: JSON Array of data to be graphed. Example below:





The **TOC** panelpublishes this event to graph RealTime Data from SOS Service. This service is semi custom to the NDBC SOS service. We are using a modified SOS library from Open Layers:

The **GRAPH** window subscribes to this event and charts the JSON response as real-time data:

Channel: graphSOSData

Object returned: JSON Array of data to be graphed. Example similar to the previous page

The **TOC** panelpublishes this event and when Layers are turned on:

The **MAP** windowsubscribes to this event and makes the layer visible on the map:

Channel: add2Map

Object returned: JSON Array object of the specific layers. Layers that are accepted are listed and explained below

The **TOC** panelpublishes this event and when Layers are turned off:

The **MAP** windowsubscribes to this event and removes this specific layer from the map:

Channel: removeFromMap

Object returned: JSON Array object of the specific layers

**Layer Options:**

TMS (Tile Map Service):

L.tileLayer('http://{s}.tile.openweathermap.org/map/precipitation/{z}/{x}/{y}.png', {options});

Regular WMS: (setup PROXY)

L.tileLayer.betterWms(‘PROXY?http://coastmap.com/ecop/wms.aspx?', {

layers: [overlays.name],format: 'image/png', zIndex: integer });

ArcGIS Server Dynamic WMS:

L.esri.dynamicMapLayer('http://serverlayer/MapServer',{zIndex:4000});

SARWMS: (setup PROXY) Custom layer for SAR output with required extra parameters

L.tileLayer.sarWms(PROXY?http://172.16.16.124/oilmapwebservice20/DrawModel.aspx?', {

layers: 'model',

scenario\_id:126,

OM\_SPILLETS:true,

OM\_CONTOUR:false,

lang:0,

transparent:true,

OM\_SWEPT:true,

OM\_TRACKLINE:false,

SEARCHAREA:true,

format:'image/png',

session\_id:'15\_alzm10677AL106',

PROBABILITYGRID:false,

DAYNIGHTICON:false,

SUMMARYTABLE:false,

user\_id:15,

time:'2014-03-30T16:40:00',

zIndex:3000 });

GeoJSON Features:

There are a number of options for sending features to the map. This currently based on the Leaflet GeoJSON and marker geometry settings. Examples are found through the Leaflet JS API in the documents below. A featureGroup is a collection of different features and can be created together and sent as a group list to the map. This is tricky with how to interact with the map. That is why there is a marker clicked event that would then allow users to control what happens when a graphic is clicked. To be continued.

<http://leafletjs.com/reference.html#featuregroup>

<http://leafletjs.com/examples/geojson.html>

**Deployment**

You need JAVA JDK and JRE installed. To run *locally*, run startup.bat Tomcat using the script here: \owf\OWF-bundle-7.0.1-GA\apache-tomcat-7.0.21\bin

Access will be seen here: <https://localhost:8443/owf> username: testAdmin1 password: password

To run on *remote* server takes about an hour to configure. Required is OpenSSL and correct Keytool (Java) settings.

Use these steps to configure OWF for remote server access. User names will have to be added for key access. This is very important to ad certificates and user access!

<https://github.com/ozone-development/ozp-docs/blob/master/owf/src/main/resources/OWF7/OWF-7-Allowing-Remote-Access-to-OWF.md>

If setup for production, tomcat will have to be setup as a service to run/startup automatically.