

## Making a Clickable Map Display of CAP Alerts

presented 2 November 2018  
by Eliot Christian <eliot.j.christian@gmail.com>  
at the Filtered Alert Hub Workshop  
in Hong Kong, China

This is a presentation about Making a Clickable Map Display of CAP Alerts. It describes freeware that presents a Web page in HTML with embedded Javascript.

That clickable map and the information presented can be customized as much you like, including to display alerts from any CAP news feed.



## The Javascript Code

---

Use a Web browser to go to the Filtered Alert Hub site at <http://alert-hub.org>

Next, "Click [here](#) for recent alerts" (the link is <https://s3-eu-west-1.amazonaws.com/alert-hub/cap-alerts.html> ) and explore the page by clicking on different points

Hit "CTRL-u" to view the source code, and then you can save a copy as your own file

2 November 2018

Making a Clickable Map Display

2

The code for the Web page can be fetched easily with a Web browser, from the Filtered Alert Hub site at <http://alert-hub.org>.

Once you have the page, try clicking on different points in the map so you will see the generated pop-ups.

Later in this Seminar, we will look at how the map is displayed and user interaction is enabled. For right now, it is enough to know that the only user interaction enabled on the map is the "onClick" event.

Now, to view the HTML and Javascript code, hit "CTRL-u" .

You will likely want to save a copy of this page as your own file that you can customize, as I will explain.



## Presentation Outline

---

- 1 JQuery and JSON
- 2 controller() function and "promise chain"
- 3 compileAlerts() then show each alert
- 4 Open Street Map and Leaflet
- 5 Handling polygons and circles

2 November 2018

Making a Clickable Map Display

3

Now that you have the code, here are the subtopics based on that code that we will explore.

But, before getting down to the code, I need to be sure we are all familiar with a couple of utilities that are in very common use in Javascript programming.

These are JQuery and JSON.



## JQuery AJAX

- JavaScript library to simplify common tasks

```
<script src="https://ajax [...] /jquery/3.1.1/jquery.min.js"
```

```
<!--[if lte IE 9]>
```

```
<script src="//cdnjs.cloudflare.com/ajax/libs/jquery [...] "
```

- AJAX is "Asynchronous JavaScript and XML"

```
28 function showSources() {
29     $.ajax({
30         'type': 'GET',
31         'url': 'https://s3-eu-west-1.amazonaws.com/alert-hub-sources/json',
32         'dataType': 'json',
33         'success': function(json) {
```

The "aynschronouos" function is the "callback" part of the \$.ajax function

2 November 2018

Making a Clickable Map Display

4

jQuery is a JavaScript library intended to simplify some of the most common tasks in Javascript programming. You can see where the library is included in our code. There is also a retro-fit library to provide JQuery on Internet Explore browsers prior to version 9.

We are focusing here on the part of JQuery called Ajax. AJAX stands for "Asynchronous JavaScript and XML". Asynchronous processing is critically important to the Filtered Alert Hub, because speed is crucial for near-real-time event processing.

Ajax provides a fairly simple programming interface for fetching files. But, the particular attraction in our case is that our file fetching requires "cross-domain requests" and JQuery is highly recommended for that.

Here we see our code using the ajax function to fetch a file. Note that the "aynschronouos" function is the "callback" part of the \$.ajax function. Nowadays, a "callback function" would be implemented with a "promise function" so that the code is much easier to understand and maintain.

That asynchronous function does not have its own name in this case, but we do see the name of the data structure that the function returns: a data structure named "json".



## JSON (JavaScript Object Notation)

- JSON is a text notation for data interchange
- Its syntax borrows from JavaScript, making it easier for use by Javascript programmers
- Data in JSON is packaged as name/value pairs, where the colon separates name from value and name/value pairs are separated by the comma
- Objects are indicated by curly braces { } and arrays are indicated by square brackets [ ]
- Although simpler and less verbose than XML, JSON does not have schemas and cannot substitute for XML
- Free JSON tool: <http://codebeautify.org/jsonviewer>

2 November 2018

Making a Clickable Map Display

5

JSON is a text notation for data interchange. Its syntax borrows from JavaScript, which make it relatively easy for Javascript programmers to learn and use.

Data in JSON is packaged as name/value pairs, where the colon character separates name from value, and name/value pairs are separated by the comma character.

Objects in JSON are indicated by a matched pair of curly braces { } and arrays are indicated by a matched pair of square brackets [ ]

As you will see, the Filtered Alert Hub technology uses both JSON and XML. Although JSON is simpler and less verbose than XML, JSON does not have a sophisticated object model with schema validation. That lack greatly complicates the necessary validation of alert data received from external sources.


Since alert data can be life-critical, the lack of validation alone means that JSON will never be a complete substitute for XML in the context of alerting.

So, we use XML where needed, such as parsing or creating CAP alerts and internet news feeds. We use the simpler JSON structures where it is adequate and validation is not of interest.

Here I provide a link to a handy tool for dealing with JSON.



## Presentation Outline

- 1 JQuery and JSON
-  2 controller() function and "promise chain"
- 3 compileAlerts() then show each alert
- 4 Open Street Map and Leaflet
- 5 Handling polygons and circles

2 November 2018

Making a Clickable Map Display

6

Within the code itself, we will start by looking at the controller function.

This function is initiated by the HTML body onload event. The controller function uses a chain of "promise" operations. As I mentioned, a promise function is essentially a simplified way to handle asynchronous callbacks.

The main functions in the Javascript code for the Filtered Alert Hub alerts display are intended to first compile all alerts and then show each of those alerts. On completion of the promise chain, the map display will be ready for user interaction.

I will explain briefly about dependencies in the code other than JQuery and JSON. Those dependencies are: Open Street Map, Leaflet, and Leaflet-PIP (Point in Polygon).

I will also highlight some of the Javascript routines that handle polygons and circles.

So, now let's look at some of the Javascript code.



## controller() function and "promise chain"

The controller() function sets up four "Promise" functions

The first three are executed by a "promise chain" ( **.then** )

The last promise is actually an array of promise functions, one for each CAP alert displayed

```
function controller() {
  var getSubscriptionParms = function() {
    return new Promise(function(resolve, reject){
      [...]
    })
  }
  var compileAlerts = function() {
    return new Promise(function(resolve, reject){
      [...]
    })
  }
  var initMap = function() {
    return new Promise(function(resolve, reject){
      [...]
    })
  }
  getSubscriptionParms()
    .then(compileAlerts)
    .then(initMap)
    .then(function() {
      var promises = [];
      for(var i = 0; i < countAlerts; i++) {
        var thisAlertPromise = new Promise(
          function(resolve, reject){
            [...]
          }
        );
        promises.push(thisAlertPromise);
      }
    })
  }
}
```

2 November 2018

Making a Clickable Map Display

7

Notice that the controller() function sets up four "Promise" functions. The first three Promise functions are named:

getSubscriptionParms  
compileAlerts  
initMap

The first three functions are executed by a "promise chain". Because there is a **.then** connecting the promise functions, each of the functions must complete before the next function executes.

The last promise is actually an array of promise functions, one for each CAP alert being displayed. Here too, all of the promise functions in the array must complete before moving on.

In the code, you will see there is a **.catch** function. This function handles any error set by any one of the promise functions. The setting of an error in a promise is accomplished with **"reject(error)"**.



## getSubscriptionParms

Using JQuery AJAX, "getSubscriptionParms" promise function does an HTTPS GET of the JSON file:  
<https://alert-hub-subscriptions.s3.amazonaws.com/json>

Find the JSON item matching the given subscriptionId, extract the subscriptionName, feedItemsLimit, and polygonCoordinates for that subscription

Use the polygonCoordinates to set values for the map center latitude, longitude, and zoom

2 November 2018

Making a Clickable Map Display

8

Now we are exploring the first promise function, which is named "getSubscriptionParms".

We can see that the promise starts with use of JQuery AJAX to execute an HTTPS GET of the JSON file named **<https://alert-hub-subscriptions.s3.amazonaws.com/json>**

That file is the master list of all Filtered Alert Hub subscriptions. As of today, there are more than 2400 subscriptions.

Once the file is retrieved by JQuery, control passes to 'success' if the retrieval request had no errors from the server, and there were no JSON errors in the data.

The file contents is then searched to find the JSON item matching the value in the **subscriptionId** variable. That identifies the subscription feed to be displayed.

NOTE: To make this clickable map code use a different subscription feed, or any other CAP news feed in RSS format, just change the subscription file name parameter.


Now, having found the JSON item that matches the subscription sought, the code extracts subscriptionName, feedItemsLimit, and polygonCoordinates. The polygonCoordinates are then used to set values for the map center latitude, center longitude, and map zoom.





## Presentation Outline

---

- 1 JQuery and JSON
- 2 controller() function and "promise chain"
-  3 compileAlerts() then show each alert
- 4 Open Street Map and Leaflet
- 5 Handling polygons and circles

2 November 2018

Making a Clickable Map Display

9

Now we turn to functions in the Javascript code for the Filtered Alert Hub that actually deal with alerts.

First we'll look at compileAlerts() and then we'll look at how the code shows each of the alerts.



## compileAlerts

```
var compileAlerts = function() {
  return new Promise(function(resolve, reject){
    $.ajax({
      type: "GET",
      url: subscriptionUrl,
      dataType: "xml",
      success: function(rssXml) {
```

The Javascript "compileAlerts" promise performs an HTTPS GET of the JSON file at **subscriptionUrl**, e.g.:

<https://s3-eu-west-1.amazonaws.com/alert-feeds/unfiltered/rss.xml>

2 November 2018

Making a Clickable Map Display

10

We can see that the Javascript "compileAlerts" function uses JQuery AJAX to execute an HTTPS GET of the JSON file named in the **subscriptionUrl** variable.

Here, we are displaying all alerts from the Filtered Alert Hub. Therefore, the **subscriptionId** we want is "**unfiltered**". The full subscriptionURL is: <https://s3-eu-west-1.amazonaws.com/alert-feeds/unfiltered/rss.xml>

## compileAlerts

Once compileAlerts retrieves the subscription feed file, it pushes to a table each CAP alert with its values of title, link, and pubDate

CAP alerts are compiled only up to feedItemsLimit

Once compilation is done, the table is reversed

Then, an array of promises reads the alerts table and processes each CAP alert

```

success: function(rssXml) {
  var title = $(this).find('title').text();
  var link = $(this).find('link').text();
  var pubDate = $(this).find('pubDate').text();
  var alertRef = {
    'title': title,
    'link': link,
    'pubDate': pubDate
  }
  fileRefs.alerts.push(alertRef);
  countAlerts = countAlerts + 1;
  if (countAlerts > feedItemsLimit) {
    return false;
  }
  }); // end of items each loop
  resolve("sucess");
}, // end of success function
error: function(xhr, status, error) {
  reject("Error: "+xhr.responseText+" on attempt "+
    "to get subscription URL: "+subscriptionUrl);
}
}); // end of ajax function GET xml
}); // end of Promise function
}; // end of compileAlerts

```

2 November 2018

Making a Clickable Map Display

11

Here we see more of the actual Javascript code.

Once compileAlerts retrieves the subscription feed file, it reads each CAP alert and pushes to a table the values of **title**, **link**, and **pubDate** from that alert. For processing we need just the link, but we bring also the title and pubDate for use in an error message in case the fetch from that link fails.

CAP alerts are compiled from the subscription feed only up to the value of the feedItemsLimit specified for that subscription feed.

When the compilation is done, an array of promises will be used to process each item in the table of CAP alerts and to map it.

You might be wondering: Why do this in two steps: Why walk through the items in the feed to make a table, and then walk through the table?

We do this because a news feed always shows the *most recent* items **first** but when we display alerts on a map we must have the *most recent* alerts **last**. That is necessary since the area specified in a subsequent alert often overlaps the area of one or more prior alerts. Any user of the map expects to see the most recent alerts on the **top** layer rather than the **bottom** layer.

Because we need to walk the alerts table backwards, we apply a

Javascript array "reverse" function to the alerts table.

## showThisAlert

- For each table entry, showThisAlert gets the CAP XML using "link"
- capNS makes regular the XML namespace
- Nested functions then parse the alertXml structure to get all the polygons and circles, plus other CAP values useful when a user clicks on the mapped alert

```
function showThisAlert(title, link, pubDate, resolve, reject) {
  $.ajax({
    type: "GET",
    url: link,
    dataType: "xml",
    success: function(alertXml) {
      capNs = nsPrefix(alertXml, "urn:oasis:names:tc:emergency:cap");
      -----
      $(alertXml).find(capNs+"info").each(function() {
        -----
        thisInfo.find(capNs+"area").each(function () {
          -----
          thisArea.find(capNs+"polygon").each(function() {
            -----
            thisArea.find(capNs+"circle").each(function() {
              -----
            }); // end of this alert/info/area/circle function
          }); // end of this alert/info/area function
        }); // end of this alert/info function
        return resolve();
      }, // end of success function
      -----
    }); // end of ajax function GET json
  });
}
```

2 November 2018

Making a Clickable Map Display

12

Here we see a skeleton view of the code for the **showThisAlert** function.

In showThisAlert, the value of the link variable is a URL, and the code uses that URL to fetch the CAP alert file asynchronously. On success of that "GET", the contents of that file is passed in a variable called "alertXML".

The function "nsPrefix" then makes the CAP namespace prefix regular.

Here it is useful that you already know the structure of a CAP XML file. Some elements are in the root part of the XML, but we also need sub-elements of the "info" element. Because "info" can occur multiple times, we use the "each" method to process each info element in turn.

This pattern holds for the "area" element as well, because it can occur multiple times within a given info element.

Then within a given area element, there can be multiple polygon elements and multiple circle elements.


So, the result of this nested set of functions is that we get each of the polygons and circles in the CAP alert.

In the actual coder but not visible in this skeleton, the code also extracts certain other values we will need when the alert is displayed on a map.



## Presentation Outline

---

- 1 JQuery and JSON
- 2 controller() function and "promise chain"
- 3 compileAlerts() then show each alert
-  4 Open Street Map and Leaflet
- 5 Handling polygons and circles

2 November 2018

Making a Clickable Map Display

13

We have seen how the alert files are retrieved and parsed. Now let's look at how alerts are displayed on an interactive map.

This is where we make use of OpenStreetMap and Leaflet.



## Open Street Map and Leaflet

- Filtered Alert Hub now uses [OpenStreetMap](#) for the mapping base layers; others could be used instead
- Leaflet is an open-source library for interactive maps
- Map is initialized through Leaflet in function `initMap`
- `onEachFeature()` sets an alert area to yellow or red
- `handleClick()` generates popup for alerts at the clicked point, using Leaflet PIP (Point in Polygon)

2 November 2018

Making a Clickable Map Display

14

OpenStreetMap is a digital map of the world. It is free to use under an open license. Although the Filtered Alert Hub uses OpenStreetMap for its mapping base layers; we could have used other sources instead, such as Google Maps or ESRI tools.

The promise function **initMap** is where our alerts display map is initialized. For that, we use Leaflet, an open-source JavaScript library for interactive maps.

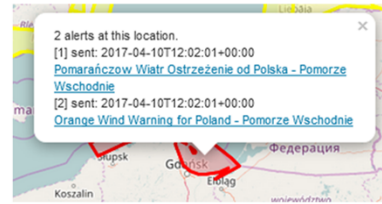
The function **onEachFeature** sets an alert area to yellow or red. This is done using "setStyle" based on the feature property named "priority". Priority is set to either "lower" (yellow) or "highest" (red) based on the values of three CAP alert elements: urgency, severity, and certainty.

The function `handleClick` is called when a click event occurs on the map. Its purpose is to generate the popup which will show the map user which alert areas overlap the clicked point. That is accomplished using Leaflet PIP (Point in Polygon), another open source Javascript library.



## Making a geojson Feature

- Clicking on a geojson feature results in a popup that lists all alerts at the clicked point
- The popup content is part of the geojson feature code



```
var popupContent = " sent: "+sent+"<br/>"+
  "<a href='"+link+"'>"+headlineOrEvent+"</a>";
var geojsonFeature = {
  "type": "Feature",
  "name": headlineOrEvent,
  "properties": {
    "priority": priority,
    "popupContent": popupContent
  },
  "geometry": {
    "type": "Polygon",
    "coordinates": [ polygonCoords ]
  }
};
return geojsonFeature;
```

2 November 2018

Making a Clickable Map Display

15

When the user clicks on a feature on the map, a popup is shown that lists all alerts at the clicked point.


The popup content is part of the geojson feature code, shown here.





## Presentation Outline

---

- 1 JQuery and JSON
- 2 controller() function and "promise chain"
- 3 compileAlerts() then show each alert
- 4 Open Street Map and Leaflet
-  5 Handling polygons and circles

2 November 2018

Making a Clickable Map Display

16

As our last subtopic, I have a few things to say about handling polygons and circles.



## Handling polygons and circles

- `constructPolygon()` makes `geojsonPolygon` from `capPolygon`
- add feature to map: `"alerts.addData(geojsonPolygon);"`
- `constructCircle()` makes `geojsonPolygonCircle` from `CapCircle`
- add feature to map: `"alerts.addData(geojsonPolygonCircle);"`
- Circle special handling:
  - If the circle has zero radius, change radius to 1 (kilometer)
  - Convert each circle to a 20-sided geoJSON polygon so the Leaflet PIP facility can handle it

2 November 2018

Making a Clickable Map Display

17

I mentioned earlier that the CAP alert can have multiple area elements within any given info element. For each CAP area element, we extract each polygon and each circle.

For each polygon, we must convert the `capPolygon` to a map feature named `"geojsonPolygon"`. That is accomplished with the function named `constructPolygon`. Once that is done, the map feature is added at the line coded with: `"alerts.addData(geojsonPolygon)"`

For each circle, we must convert the `CapCircle` to a map feature named `"geojsonPolygonCircle"`. That is accomplished with the function named `constructCircle`. Once that is done, the map feature is added at the line coded with: `"alerts.addData(geojsonPolygonCircle);"`

There are a couple peculiarities in handling a CAP alert circle. One is that some CAP sources provide a circle with zero as the radius. But, a circle with radius zero is actually a point and has no extent. For the purpose of mapping, we go ahead and coerce the radius to one, so we have instead a circle with a radius of one kilometer.

Also, the Leaflet Point in Polygon facility does not handles circles. So we need to convert the circle to a many-sided geoJSON polygon. The function `polygonFromCircle` performs that conversion. It makes a polygon with 20 sides, which seems an adequate approximation.



## Further Reading and Exercises

---

### Further Reading

- [Leaflet Quick Start Guide](#)

### Exercises

- Change the subscriptionId to select a different feed
- Changing the format and content of the Legend
- Create and test code to change what information is displayed when the user clicks on a map feature

2 November 2018

Making a Clickable Map Display

18

I provide a pointer to the Leaflet Quick Start Guide. This is a very good way to gain an understanding of Leaflet.

For exercises, I suggest that you first change the subscriptionId to select a different feed. Next, try changing the format and content of the Legend on the map.

You could also play with changing what information is displayed when the user clicks on a map feature.