Build a Fleet Management App in 5 Steps

Easily integrate Web-based GIS using the ArcWeb Services JavaScript API

By Andy Gup, ESRI

This is the second in a series of articles about the ArcWeb Services JavaScript API. These articles demonstrate how easy it is to help solve real-world problems by integrating Web-based GIS into applications. In just five steps, learn how to build a prototype real-time fleet management application. Completing this project does not require installing any mapping software or data. Simply set up the application on a Web server and start building.

The ArcWeb Services JavaScript API was designed for developers who want to integrate Web-based mapping into their applications without spending a significant amount of time developing code or spatially enabling a database. Designed for the easy deployment of mapcentric, rich Internet applications (RIAs), this API provides access to a wide variety of on-demand mapping functionality and content. Based on AJAX and Adobe Flex, it is a subset of the ArcWeb Services family of APIs that include SOAP, REST, OpenLS, and Java ME—all hosted by ESRI. These APIs can augment an existing GIS installation or be used by themselves over the Internet.

The functional requirements for the prototype fleet management application are

- Define a geographic coverage area for the fleet (also known as a geofence).
- Overlay the geographic coverage area as an opaque polygon on top of the map.
- Enable the polygon to stay the correct size at different zoom levels and move with the map as it is panned.
- Pull vehicle locations from an XML file into the client at regular, defined intervals.
- Display the vehicles as icons on a map along with labels.
- Determine if a vehicle leaves or enters the coverage area and send alerts.
- Move the vehicle icons dynamically on the map at regular intervals without reloading the entire HTML page.
- Display detailed, real-time traffic information on the map.

The JavaScript API not only significantly reduces the amount of scripting a developer must write, it also does not require extensive knowledge of the underlying core mapping technology. For example, the sample provided also includes widgets for pan and zoom as well as turning on or off satellite imagery.

Setting Up ArcWeb Services

Before starting this exercise, some basic information on setting up ArcWeb Services is needed as well as an understanding of the basics of building an ArcWeb Services application. This information is available from a previous *ArcUser* article, "3 Steps in One Hour—Building a business intelligence application with the ArcWeb Services JavaScript API," which is available online at www.esri.com/news/arcuser/0207/files/3steps.pdf. Use an existing ArcWeb Services account or sign up for a free 90-day ArcWeb Services evaluation at www.esri.com/arcwebservices. For ArcWeb Services API documentation, visit www.arcwebservices.com/help. The sample code for this example is available at arcscripts.esri.com/details.asp?dbid=15075.

Step 1

Retrieve and Parse the XML File

A good place to start is reviewing the server-side logic. Since this article is about prototyping, the sample client code simulates updating an XML file containing vehicle locations that are dynamically pulled from GPS-enabled vehicles to the server.

Whatever method is used to create the XML file, it has to reside locally on the Web server so that the client application can access it. Listing 1 approximates the XML file included with the sample code.

```
<?xml version="1.0" ?>
<markers>
  <marker lat="32.768" lon="-117.2" label="Vehicle A" />
  <marker lat="32.767" lon="-117.2" label="Vehicle B" />
  <marker lat="32.765" lon="-117.2" label="Vehicle C" />
  <marker lat="32.765" lon="-117.205" label="Vehicle D" />
</markers>
```

Listing 1: Sample client code simulating vehicle locations.

The XML file is very basic. There are four vehicles in this fleet, and for each vehicle a latitude and a longitude coordinate is provided. A fully functional application would also include other attributes such as time stamps and vehicle speed. Depending on the application, elevation information might be used.

AJAX helps retrieve the XML file from your Web server and transport it to the client application. The ArcWeb Services JavaScript API calls a Prototype-based JavaScript library that provides the AJAX functionality. (This article doesn't cover the differences between Prototype and non-Prototype JavaScript libraries.) The majority of the samples in the API developer documents are Prototype based. A non-Prototype library is available for download from the ArcWeb Services Online Help pages.

The Prototype library is a class-based library. It helps take care of serialization and deserialization of data such as vehicle locations stored in XML files. The APIs do all the work to maintain an exact copy of data when passing it across application domains. The library also contains a built-in JSON parser that works behind the scenes to handle various objects, strings and arrays. JSON is a subset of JavaScript that provides a lightweight data-interchange format.

To retrieve th file, use an Ajax Request() object inside the onCreationComplete() function. This initiates an asynchronous HTTP request without the need to refresh the entire browser. The Ajax.Request Class is a wrapper for the XMLHttpRequest object. This AJAX request should look like the one shown in Listing 2. The basics are fairly straightforward.

```
var ajaxReq = new Ajax.Request("XML/locations.xml",
{method: 'get', onComplete: completeHandler, onFailure:
failureHandler});
```

Listing 2: Creating an AJAX request object

1. Specify the location of the XML file. Specify the method for retrieving this information as GET, which is a Prototype-specific setting.
2. Next, two callbacks are used: onComplete and onFailure. When the request is finished, the completeHandler() function takes the responseXML property and uses it to retrieve the response body from the AJAX request object. The response body is an XML Document Object Model (DOM) object. AJAX attempts to send a binary stream to the server. If the connection is successful, the XML file is returned to the client. If the connection fails, the failureHanlder() function runs. More information on prototype library callbacks is available at www.prototypejs.org/api/ajax/request.
3. The script now needs to loop through the XML file and parse it into an array that separates out the various attributes. The attributes (in this case) are lat (latitude), lon (longitude), and label (vehicle description). Use the addMarker() method in the AWMap Class to push the array of information to the actual map image.

Step 2

Create the Coverage Area Polygon

With the vehicle locations loaded into the map, the next step is to address the coverage area polygon and the real-time traffic feed requirements. Polygons are defined by creating three or more latitude/longitude coordinates. These coordinates must be listed in a particular order for the polygon to build correctly. An easy way to think about this is to list the points in a clockwise fashion. For the rectangular coverage area, the coordinates are defined by first listing the top left (32.773, -117.21), top right (32.773, -117.19), bottom right (32.755, -117.19), and bottom left (32.755, -117.21) coordinates as shown in Listing 3.

To create a polygon, use the addPolygon() method in the AWPolygon Class. For brevity, the sample uses all the default polygon drawing settings and, if it executes correctly, will create an opaque blue rectangle in the center of the map with the street map visible underneath the polygon along with the icons representing vehicles. To implement something other than the default style, investigate the AWPolygonStyle() Class.

```
//Set Polygon to simulate a coverage area
var myPolygon1 = new AWPolygon(
"polygon1", // Polygon identifier.
[
__new AWLatLon(32.773, -117.21),
__new AWLatLon(32.773, -117.19),
__new AWLatLon(32.755, -117.19),
__new AWLatLon(32.755, -117.21)
]);

myExplorer.addPolygon(myPolygon1);
```

Listing 3: Setting the polygon coverage area

Continued on page 52

Continued from page 51

Step 3

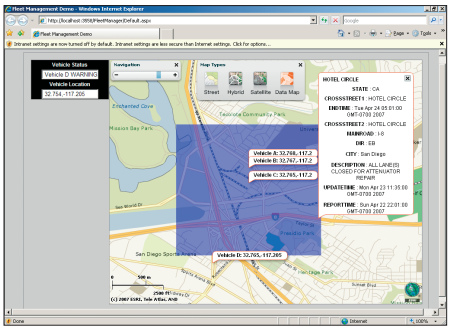
Add Real-Time Traffic

For the real-time traffic requirement, use the queryGroupLayer data source called ArcWeb:TC.Traffic.US. This data source is accessed through the addGroupLayer method in the AWMap Class. With the addGroupLayer method, numerous types of data can be easily added to the map. Simply specify a whereClause that is the same as any SQL WHERE clause, and it will limit the search results. In this example, set the city equal to San Diego, then specify a label field and a field by which to order the labels. ArcWeb Services handles the hard work of determining which incidents are returned for the viewing area and zoom level. The script snippet is shown in Listing 4.

The assignable attributes for the traffic feed include nearest cross street, travel direction, description (with up to 1,000 characters of text), expected end time of incident, severity of incident, and date and time last updated. To find additional information on fine tuning group layer parameters, go to the online documentation under ArcWeb Explorer JavaScript > Parameter Descriptions > Group Layers.

```
myExplorer.addGroupLayer("queryGroupLayer",
"ArcWeb:TC.Traffic.US",
{whereClause:"CITY = 'San Diego'",
labelField:CROSSSTREET1",
orderByField:"CROSSSTREET1",
orderByDecending:false,
count:10,
refreshScope:2
});
```

Listing 4: Using a spatial query to limit search results



The ArcWeb Services JavaScript API can be used to rapidly build applications for tracking vehicles going in and out of a specific geographic region.

Summary

Download the sample fleet management application and test drive it. The ArcWeb Services JavaScript API provides a solid platform for rapidly integrating mapping functionality and content into Web-enabled, fleet management solutions. This sample contains lots of functionality and data. Feel free to experiment with it and, perhaps, combine functionality from other ArcWeb Services APIs or consume ArcWeb Services from a variety of ESRI's ArcGIS products.

Step 4

Dynamically Move Vehicle Markers

There are two client-side methods for dynamically moving markers. The first method involves simply moving specific icons using the setMarkerLatLon() method in the AWMap Class. The second method requires removing all markers on the map using the removeAllMarkers()method, then updating all icons on the existing map using the addMarker() method. When using the addMarker() method, pay attention to browser caching issues related to the caching of GET request via XMLHttpRequest. Using either method should not require refreshing the entire HTML screen.

To limit the amount of information passed between the server and the client, consider having the server-side business logic check for vehicle movement and only send information to clients that have changed since the last update. This method could potentially reduce network traffic between client and server. With a large fleet, the XML file size can get quite large. If the server is sending the entire vehicle list to the client anyway, experiment with client load times using a typical customer machine and browser as it may not be as fast or powerful as a typical development machine.

Step 5

Alert if Vehicle Leaves Coverage Area

Finally, to check if a vehicle is inside or outside a particular area, use the AWMap Class intersects object, which simply passes the polygonID and the markerID and the object returns a Boolean. True means the marker representing the vehicle is within the polygon, and False means the vehicle is outside the polygon.