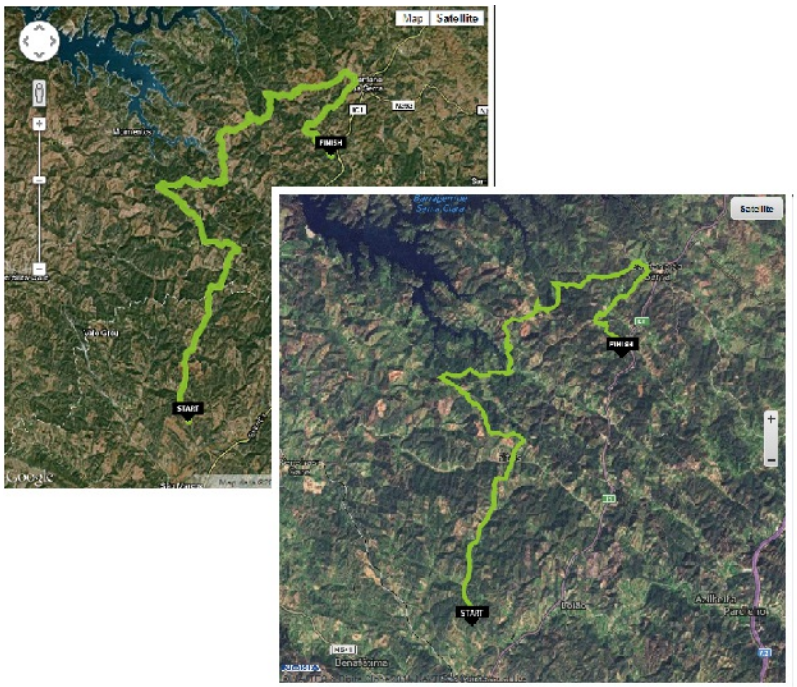
Converting Google Maps for JavaScript to HERE Maps

This article explains how to port a typical web page to use the HERE Maps API for JavaScript instead of Google Maps for JavaScript, and describes a simple metric to measure the change in performance.

01 Jan 2012

Introduction

The latest version of the HERE Maps API has a faster start-up time, lowest number of bytes received and lowest number of data requests per map across a full range of web browsers, compared to its rivals. As such it arguably offers a better user experience. Since the most popular alternative on the web at the moment is Google Maps, this article aims to show how to convert a typical web page from Google Maps to HERE Maps and measure the benefit gained.



Screenshot showing the route of the Vodafone Rally de Portugal using both Google Maps (left) and the equivalent with HERE Maps (right)

Initialisation of the Map

Initialisation occurs in two parts: firstly, the relevant JavaScript library needs to be loaded, and secondly, a Map needs to be initialized within a DOM element on the page.

Header

Google Map	HERE Map

`<script type="text/javascript"`
 	
`src="http://maps.google.com/maps/api/js?sensor=false">`
`</script>`
	`<script type="text/javascript"`
	
	`src="http://js.api.here.com/se/2.5.3/jsl.js" charset="utf-8">`
	`</script>`

Since both the Google Maps API and the HERE Maps API are JavaScript libraries, they both need to be loaded in the header of the page. This is achieved by adding a `<script>` tag. If allowed by the relevant browser, the JavaScript libraries will be cached after loading to reduce start-up on subsequent page loads.

Note: By default, only the mapping functionality of the API is loaded on start up - see Packages and Detection. If additional functionality such as routing is required it can be added by altering the code to load the library as follows:

```
http://js.api.here.com/se/2.5.3/jsl.js?routing=auto"
```

IE6 Support

If IE6 needs to be supported, both libraries need an additional line in the meta data in the form shown below:

(C) Copyright Nokia 2014. All rights reserved.

```
<meta http-equiv="X-UA-Compatible" content="IE=7; IE=EmulateIE9" />
```

Body

Google Map	HERE Map

`var latlng=new google.maps.LatLng(37.44406,-8.35167);`
`var myOptions={`
`  zoom:11, // Zoom level for the map`
`  center:latlng, // Center coordinates`
`  mapTypeId: google.maps.MapTypeId.HYBRID};`
`map=new`
`google.maps.Map(document.getElementById('gmapcanvas'),myOptions);`
	`var map = new nokia.maps.map.Display(`
	
	`document.getElementById('gmapcanvas'),`
	`{`
	`  'zoomLevel': 11, // Zoom level for the map`
	`  'center': [37.44406,-8.35167] // Center coordinates`
	`});`
	`map.set("baseMapType", map.SATELLITE);`

Both the Google Map and the HERE Map accept a zoom and center point in the map constructor, along with a DOM element (typically a `<div>`) to display the map. The difference lies in how the two APIs decide which style of map to display. The Google Map requires a map type in its constructor, whereas the the HERE Map will default to a street map. If another type of map is desired, the type of the HERE Map should be altered programatically as shown.

- see also : Basic Map , Switching Map Types

Note: The DOM element which holds the map - in this case the (`'gmapcanvas'`), needs to be defined before the `<script>` setting up the display is run. This can be achieved by the simple expedient of placing the `<script>` after the `<div>` in the code:

```
<div id='gmapcanvas' style="width:600px; height:600px;">nbsp;</div>
<script>var map = new nokia.maps.map.Display( document.getElementById('gmapcanvas'), {
    'zoomLevel': 11,  'center': [37.44406,-8.35167] });
</script>
```

also check that the map has a defined height and width by adding a style attribute.

Tip: A cleaner separation of HTML and JavaScript can be obtained by calling the initialisation script as a function after the `<body>` has loaded

```
<head>
    <script>
    function goPageOnLoad() {
        var map = new nokia.maps.map.Display( document.getElementById('gmapcanvas'), {
        'zoomLevel': 11,  'center': [37.44406,-8.35167] });
    }
</script>
</head>
<body onload="goPageOnLoad();" >
    <div id='gmapcanvas' style="width:600px; height:600px;" >nbsp;</div>
</body>
```

Adding Markers

The following code adds a custom marker to the map:

Google Map	HERE Map

`var miStart=new google.maps.MarkerImage`
 	
`('http://www.wrc.com//img/mapicon/start.png');`
`var miFinish=new google.maps.MarkerImage`
 	
`('http://www.wrc.com//img/mapicon/finish.png');`
`var markerEnd = new google.maps.Marker({`
`  position:roadCoords[roadCoords.length-1],`
	`var markerStart = new nokia.maps.map.Marker(`
	`  roadCoords[0], {`
	`  icon: "http://www.wrc.com//img/mapicon/start.png",`
	`  anchor: new nokia.maps.util.Point(20, 12)`
	`});`
	`var markerEnd = new nokia.maps.map.Marker(`

(C) Copyright Nokia 2014. All rights reserved.

```
     icon:miFinish,
     map: map,
     title:'End'});
var markerStart = new google.maps.Marker({
     position:roadCoords[0],
     icon:miStart,
     map: map,
     title:'Start'});
```

```
var markerEnd = new nokia.maps.map.Marker(
     roadCoords[roadCoords.length-1],{
     icon:
"http://www.wrc.com//img/mapicon/finish.png",
     anchor: new nokia.maps.util.Point(20,
12)
});
map.objects.addAll([markerStart, markerEnd]);
```

Most maps require some sort of Markers to add focus to the map. For the custom markers illustrated above, the Google map creates a `MarkerImage` and then creates the `Marker` passing in both the image and the map. The HERE Map creates a `Marker` directly and then the `Map` adds the `Marker` rather than the other way round. Google's `position` attribute is implicit in the Nokia `Marker` syntax. The `anchor` attribute decides the offset point of the icon.

- see also : Adding Markers

Adding a Route

The following code adds a route to the map; the route in the coded example is much more complex, and contains many more points:

Google Map

```
var roadCoords=[
     new
google.maps.LatLng(37.385433,-8.375983)
    ,new
google.maps.LatLng(37.385577,-8.376008)
    ,new
google.maps.LatLng(37.385674,-8.376025)
];
rallyroute=new google.maps.Polyline(
     {path:roadCoords,
      strokeColor:'#8dc72d',
      strokeOpacity:0.9,
      strokeWeight:8});
rallyroute.setMap(map);
```

HERE Map

```
var roadCoords=[
     new
nokia.maps.geo.Coordinate(37.385433,-8.375983)
    ,new
nokia.maps.geo.Coordinate(37.385577,-8.376008)
   ,new
nokia.maps.geo.Coordinate(37.385674,-8.376025)
];

map.objects.add(new nokia.maps.map.Polyline(
 roadCoords,
 {pen: {
             strokeColor: "#8dc72d",
      lineWidth: 5
 }
}));
```

Adding a route is another common use case. Both APIs accept a list of `GeoCoordinates` as the basis for the route. Again the Google syntax is to add the `Map` as a function of the `Route`, whereas the HERE Maps API adds the `Route` to the `Map`. In a similar fashion to the `Marker` syntax above, Google's `path` attribute is implicit to the HERE Map constructor for a `Route`.

- see also : Adding a route

Adding Observers

The following code restricts the zoom level of the map to 14 or lower:

Google Map

```
var zoomObserver = function (obj, key, newValue,
oldValue) {
 if (newValue > 14){
   map.set("zoomLevel", oldValue);
 }
};

google.maps.event.addListener(map,'zoom_changed',
zoomObserver );
```

HERE Map

```
var zoomObserver = function
(obj, key, newValue, oldValue) {
 if (newValue > 14){
   map.set("zoomLevel",
oldValue);
 }
};

map.addObserver("zoomLevel",
zoomObserver );
```

Adding an observer allows the developer to write code which fires on a specific map event or change of property. In the example above, this is used to limit the zoom level. As you can see the syntax is **exactly** the same in both cases. The only difference is the name of the property to be observed.

(C) Copyright Nokia 2014. All rights reserved.

- See also : Observing Map Properties

Adding an Infobubble

The following code adds an info bubble to a marker:

Google Map	HERE Map
`var infoWindow = new google.maps.InfoWindow();`	`var infoBubbles = new nokia.maps.map.component.InfoBubbles(); map.addComponent( infoBubbles);`
`markerStart.html = "<div>SOME HTML</div>"; google.maps.event.addListener(markerStart, 'click', function() { infoWindow.setContent(this.html); infoWindow.open(map, this); });`	`markerStart.html = "<div>SOME HTML</div>"; markerStart.addListener("click" , function(evt) { infoBubbles.addBubble(evt.target.html, evt.target.coordinate); }, false);`

An `infobubble` is some sort of pop-up text, usually triggered to obtain additional details when a marker is clicked on. As such it is a typical use of the `Observer` pattern described above. Since not all maps require info bubbles, HERE Maps requires the explicit definition of an infobubble component and the addition of this component to the map prior to use. Meanwhile, the Google `InfoWindow` equivalent is always supplied by the framework on start up regardless of use, and hence only requires the infowindow definition. In both cases some sort of additional `.html` parameter needs to be held in memory along with the marker's `onclick` function

Note: The `.html` attribute of the marker can be given any name, `.html` seems to be the most common.

- see also : Adding an Infobubble to a marker

How to Measure When the Map has finished loading

The most obvious reason for switching from one API to another is if there is an increase in performance. Some sort of metric needs to be established in order to verify that one library is more capable of satisfying a user's needs than another. A simple example of such a metric would be to measure the total start up time from loading a page to when the map is fully rendered.

Header

The following code can be placed in the `<head>` of the HTML file to start a timer on load and display the result in a `<span>` called `'endtime`

```
<script type="text/javascript">
var startTime=new Date();

function StopTime()
  {
    document.getElementById("endTime").innerHTML=String((new Date()-startTime)/1000);
  }
</script>
```

Body

Additionally, the following `<span>` should be placed in the `<body>` of the page

```
<span id="endTime">0.0</span>
```

Adding an Observer

`StopTime()` needs to be called once the Map has rendered: this is done by adding an observer, as in the example above. The events chosen are the `idle` event for the Google Map and the `mapviewchangeend` for the HERE Map. Choosing a different metric will obviously result in slightly different results.

Google Map	HERE Map

(C) Copyright Nokia 2014. All rights reserved.

```
google.maps.event.addListenerOnce(map,
'idle', function(){
    // do something only the first time the
map is
    StopTime();
});
```

```
map.addObserver("mapviewchangeend",
StopTime() );
```

Result of the Generated Metric

Obviously the rate at which map data is generated on screen will depend upon a variety of factors such as the performance of the browser and the Internet traffic at the time of test. For the attached coded example, the following metric was generated for three runs of the maps across three common browsers. In all cases, the cache was emptied for the initial run only.

Note: The data in the tests below reflects the performance of the current live Google Map API and the current live HERE Map API (**2.0**) during **January 2012**. These were the latest API versions available when the tests were run. As of **April 2012**, the Worked examples have been updated to include additional samples for the latest HERE Maps API (**2.2**) and the RESTful Map API but updated test results have not been generated.

Internet Explorer

	Initial Run	Second Run	Third Run
Google Map	1.460 s	0.479 s	0.493 s
Nokia Map 2.0	0.262 s	0.269 s	0.269 s

Firefox

	Initial Run	Second Run	Third Run
Google Map	0.867 s	0.783 s	0.487 s
Nokia Map 2.0	0.368 s	0.262 s	0.350 s

Google Chrome

	Initial Run	Second Run	Third Run
Google Map	1.027 s	0.415 s	0.386 s
Nokia Map 2.0	0.674 s	0.096 s	0.234 s

Alternative Solution

The HERE Maps API for JavaScript typically creates a pannable, zoomable map on screen. If you don't need this functionality, an even quicker loading solution for displaying a **static map** would be to use the RESTful Map Image API where the location of the map is encoded into the URL of the image. For example, the following URL will display a map of the general location of the Vodafone Rally de Portugal (without the route) in approximately 0.01 seconds:

```
<img id="gmapcanvas" src="http://m.nok.it/?c=37.44406,-
8.351673&w=600&h=600&z=11&t=3&nord" alt="Vodafone Rally de Portugal" />
```

A route can be added by sampling coordinates along the Polyline. Obviously the accuracy of the route will depend upon the number of points selected, but a complex route will take around 0.15 seconds. An example of this use of the Map Image API has been added to the Worked examples

Summary

For most use cases of a JavaScript Mapping API, conversion from one API to an alternative can be achieved swiftly and painlessly. A metric can easily be generated to compare the performance of alternative APIs and decide which API should be used. Ultimately the developer's decision should be made based on the ultimate benefit to the end user working with the map.

(C) Copyright Nokia 2014. All rights reserved.

(C) Copyright Nokia 2014. All rights reserved.