HERE Maps API - How to create a custom overlay

This article explains how to create a custom overlay and add data from your own tile server to the map.

Introduction

For extremely large data sets, it no longer makes sense to add each data point as an individual `marker`, `image` or `polyline`. In situations such as displaying weather or traffic conditions over a map, it would be less processor intensive to retrieve pre-built images from a tiler server rather than re-calculating and adding a large number of `mapObjects` on to the map `display`. The cost of making a few extra round trips to retrieve data from a server will be much smaller than the equivalent client side processing of adding hundreds of thousands of objects onto the map.

Image Tile Provider

In the HERE Maps API for Javascript, the basic overlay provider is an `ImgTileProvider`, this retrieves data from a tile server and displays small 256x256 Pixel "tiles" over the regular map as shown:

Adding an Overlay to the map

This example adds an Overlay which retrieves Map Tiles from a tile server run by the National Library of Scotland
The mapping is based on out-of-copyright Ordnance Survey maps, dating from the 1920s to the 1940s.

Opacity

The example above calls to an arbitrary tile server which generates tiles in the Normalised Mercator projection, which is the commonest format for online maps. It also handles the necessary attribution of the overlay images. The code example retrieves Map Tiles from a tile server run by the National Library of Scotland . The mapping is based on out-of-copyright Ordnance Survey maps, dating from the 1920s to the 1940s.

A working example can be found at:

Historic Scotland Example

Defining the tile source

To create an overlay a `ImgTileProvider` requires an input function which defines the tile source. Fortunately in our case the necessary definition already exists, a function called `NLSTileUrlOS()` has been defined in the following library:

```
<script type="text/javascript" charset="UTF-8"
src="http://nls.tileserver.com/api.js"></script>
```

To hook up the tile provider, just create a method as shown. Note that within the HERE Maps API the `ImgTileProvider` passes the

(C) Copyright Nokia 2014. All rights reserved.

zoom, row and column in a different order to the `NLSTileUrlOS()`, so an adjustment to the mapping needs to be made:

```
function getTileUrl(zoom, row, column) {
 return NLSTileUrlOS( column, row, zoom );
}
```

Adding Copyright Attribution

Unless you own your own map tile source, you will have to comply with the licensing agreement of the map tile provider. In many cases this will mean giving some **attribution** to the original owner, even if the tiles are being offered without cost. Here is the licensing agreement from the National Library of Scotland used in the example:

"You can embed the map in your own website, display your own markers or mapping data on top of it, use it for research purposes, or create derivative work from it. The only condition is that you must display an attribution to the National Library of Scotland, together with a link to the National Library of Scotland website http://www.nls.uk/ whenever our map is used. If you create derivative work, the documentation of your work must contain this attribution"

A second function must be defined to show the copyright(s) if necessary. In this case, we will only display maps from zoom 5-20, so a check is made to only display the attribution if the overlay is present.

```
function showCopyright(area, zoom) {
 if (zoom < 5 || zoom > 20) {
  return [];
 }

 return [
  {
   label: "Historical maps from <a href='http://geo.nls.uk/maps/api/'>NLS Maps
API<\/a>",
   alt: "Historical maps from the National Library of Scotland"
  }
 ];
}
```

Adding the Overlay

The `getTileUrl()` and `showCopyright()` functions are passed in as `tileProviderOptions` along with other parameters such as the definition of the maximum and minimum zoom level and level of opacity (if required).

```
function addTileOverlayToMap(map){

 tileProviderOptions = {
  getUrl: getTileUrl,
  max:20,
  min:5,
  opacity: 0.5,
  alpha:true,
  getCopyrights: showCopyright
 };
 this.nlsOverlay = new nokia.maps.map.provider.ImgTileProvider(tileProviderOptions);
 map.overlays.add(nlsOverlay);
}
```

How Map Providers and Overlays Work

(C) Copyright Nokia 2014. All rights reserved.

For the base **map providers**, the globe is divided into individual map tiles. These are retrieved online from Nokia's map tile service.

- At minimum zoom, the world is contained in a single 256x256 pixel tile.
- At the next zoom level up, the world is 2 tiles wide and 2 tiles high. Then 4x4, 8x8, 16x16 etc.
- Each zoom level splits the tiles in two, doubling the number of tiles across the width and height
- Each Map is based on the Normalized Mercator projection.

Custom **overlays** work in the same fashion - using your own map tile server complex visual data can be displayed at different zoom levels.

- One tile at minimum zoom, then 2x2, 4x4 and so on.
- Must be based on the Normalized Mercator projection.
- Use `.png` tiles to support transparency, so underlying map can be seen beneath your data.

Normalised Mercator Projection

The HERE Maps API offers maps using the *Normalised Mercator Projection*. This covers the globe as a square from 85.0511°N to 85.0511°S and across all longitudes. At the lowest zoom this is contained within an individual map tile. The number of map tiles displayed doubles on each axis as the zoom level increases, therefore it is possible to obtain a map tile image at any location `{x, y}` on the globe (apart from the poles) at any zoom where:

- The `x` axis runs from 0 (left edge is 180°W) to `2^zoom -1` (right edge is 180 °E)
- The `y` axis runs from 0 (top edge is 85.0511°N) to `2^zoom -1` (bottom edge is 85.0511 °S)

The relationship between the longitude λ and latitude φ in **radians** and the map tiles `{x, y}` is defined as follows:

- `{λ, φ} -> [-1, 1] x [-1, 1]`
- `x = λ / π`
- `y = ln(tan(π/4 + φ/2)) / π`

Obviously as the zoom level increase, the number of tiles across the width of the globe also increases:

Zoom Level	Tile Width
1st zoom level	1 tile
2nd zoom level	2 tiles
3rd zoom level	4 tiles
nth zoom level	2^n tiles

Displaying the Tile definition

It is possible to add a series of SVG markers at the points defined at the top left of each tile, and then mimic an overlay to display the zoom, column and row for each tile. An additional library has been added to do this, which again accepts a function which defines each tile:

```
var tileUrl = function printTileDefintion(column, row, zoom) {
 var svgParser = new nokia.maps.gfx.SvgParser();
 var iconSVG = '<svg width="256" height="256" xmlns="http://www.w3.org/2000/svg">' +
  '<rect width="256" height="256"  style="fill:none;stroke:black;stroke-
width:5;"></rect>' +
```


(C) Copyright Nokia 2014. All rights reserved.

```
  '<text x="100" y="123" font-size="10pt" font-family="arial" font-weight="bold" '+
  'text-anchor="left" fill="black" textContent="zoom:__ZOOM__">zoom:__ZOOM__</text>' +
  '<text x="100" y="143" font-size="10pt" font-family="arial" font-weight="bold" '+
  'text-anchor="left" fill="black" textContent="col:__COLUMN__">col:__COLUMN__</text>' +
  '<text x="100" y="163" font-size="10pt" font-family="arial" font-weight="bold" ' +
  'text-anchor="left" fill="black" textContent="row:__ROW__">row:__ROW__</text>' +
  '</svg>';

 var svg = iconSVG
   .replace(/__ZOOM__/g, zoom)
   .replace(/__COLUMN__/g, column)
   .replace(/__ROW__/g, row);
 return new nokia.maps.gfx.GraphicsImage(svgParser.parseSvg(svg));
}
```

The overlay is added as shown:

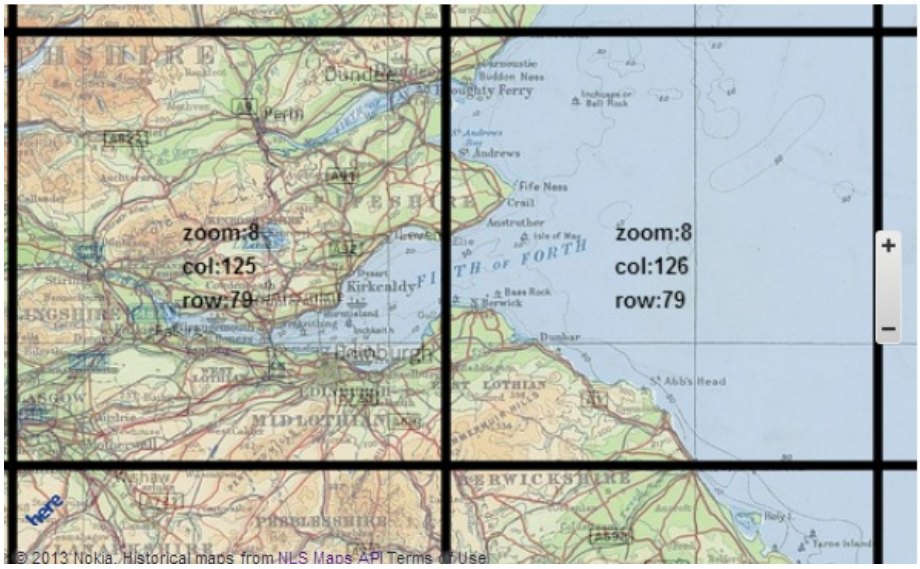
```
tileGrid =   new MarkerGridOverlay(map, printTileDefintion);
tileGrid.add();
```

Adding an Grid the zoom, column and row of a tile overlay source

This example adds an Overlay which retrieves Map Tiles from a tile server run by the National Library of Scotland
The mapping is based on out-of-copyright Ordnance Survey maps, dating from the 1920s to the 1940s.

A second overlay displays the column, row and zoom of each tile based on the normalised mercator projection.
This can be used to align and debug tile sources.

Opacity



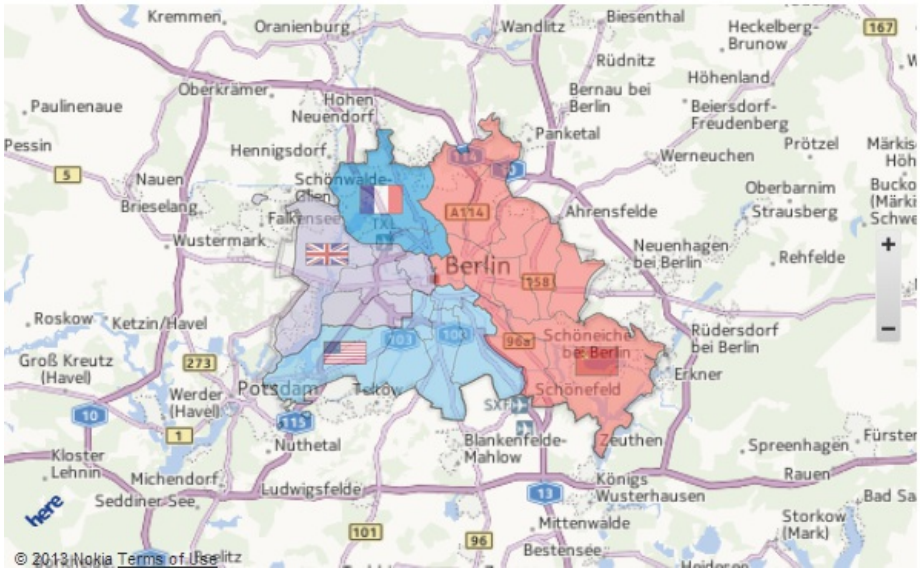
This tool could prove useful when aligning your overlay to abase map. A working example showing this can be found at:

Grid Overlay Example

Ground Overlay

An alternative form of overlay is a **ground overlay**. This is effectively a single image which is moved and re-sized to cover the same area of the map as the map is panned or zoomed.

(C) Copyright Nokia 2014. All rights reserved.



Since the ground overlay is not a standard component, an additional library is needed to add it to the map:

```
<script type="text/javascript" charset="UTF-8" src="libs/groundoverlay.js"></script>
```

The component is defined and extended as shown:

```
function extend(B, A) {
 function I() {}
 I.prototype = A.prototype;
 B.prototype = new I();
 B.prototype.constructor = B;
}
```

```
extend(GroundOverlay, nokia.maps.map.component.Component);
```

The ground overlay takes a single image url, and a `BoundingBox` when added to the map:

```
function addGroundOverlay(map){
 var berlinBounds = new nokia.maps.geo.BoundingBox(
  new nokia.maps.geo.Coordinate(52.687,13.06),
  new nokia.maps.geo.Coordinate(52.328,13.78)
 );

 map.zoomTo(berlinBounds, false);
 var dividedBerlin = new GroundOverlay(
  './img/berlin-sector.png',
  berlinBounds);

 map.components.add(dividedBerlin);
 dividedBerlin.setOpacity(0.5);
}
```

A working example showing this can be found at:

Ground Overlay Example

Summary

(C) Copyright Nokia 2014. All rights reserved.

It is possible to overlay map tile data over a HERE Map. If the images are returned in the `.png` format, transparency will be respected and the HERE Map base tile will show through underneath. - feel free to adapt the code to your own use.

(C) Copyright Nokia 2014. All rights reserved.