# HERE Maps API - How to create a KML data file

This article explains how to create and edit a sample **KML** file using an online editor

**Note:** in order to load a KML file successfully, the generated KML file should be hosted on the same domain as the JavaScript or the results may be unpredictable. Some browsers will automatically prohibit cross-domain access.
For example, if you are hosting at `example.com`, the final line of the JavaScript to load the KML will need to be:

```
kml.parseKML("http://example.com/" + "generated_kml_data_file.kml")
```

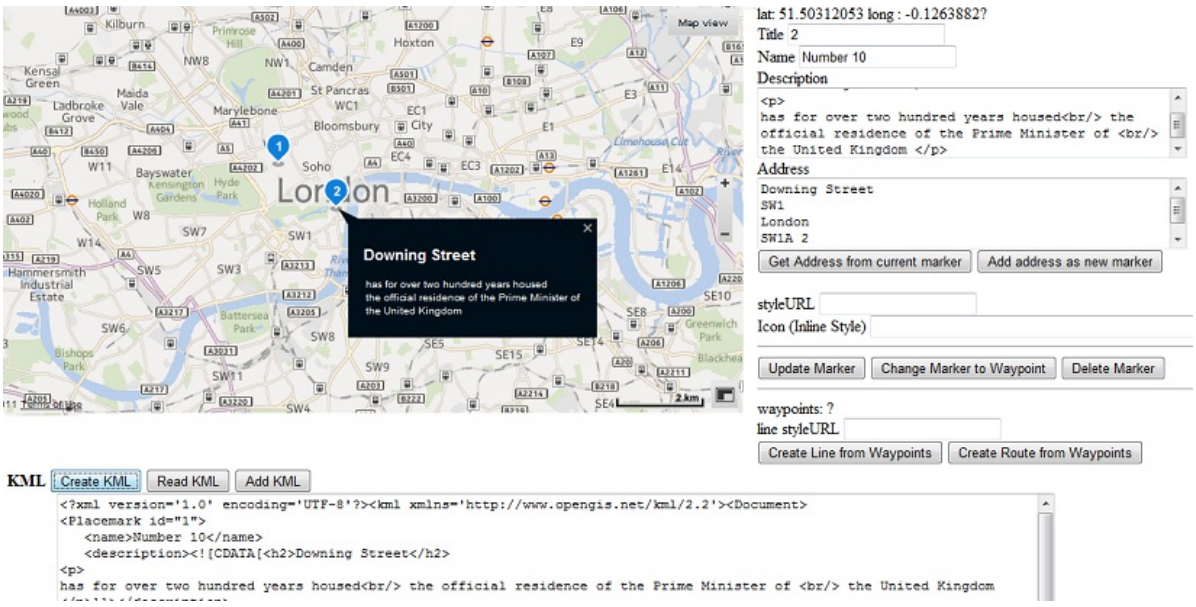
and both the KML loading HTML and the generated_kml_data_file.kml should be placed on `http://example.com/`

**Tip:** The KML reader of the HERE Maps API for JavaScript will ignore elements it is unable to interpret. KML readers vary and are more or less forgiving in the strictness of interpreting the KML specifications. It is recommended that you validate your KML syntax yourself through an online validator such as: http://feedvalidator.org

## Introduction

Keyhole Markup Language (**KML**) is an XML notation for geographic applications. The advantages of using **KML** are numerous, and have been listed in a previous article. The learning curve for **KML** is quite steep, requires a thorough understanding of both XML and mapping technologies, and since the reference documentation is written for 3d rather than 2d maps, much of it is irrelevant for simple mapping cases. All this means that some web developers are put off using **KML**, which is a pity, since **KML** is a standard data format, and the use of **KML** would result in the faster porting of a mapping applications to other platforms.

In order to remedy this, it would be useful to have a WYSIWYG *point and click* **KML** generator, where a map could be edited on screen, and the resultant **KML** file generated. The coded example below, attempts to fill this gap as shown.




## WSYIWIG Editor

### Adding Markers

Markers can be added by pressing and holding down the mouse , this is the `longpress` event. An event handler has been added to the map to cope with this. The associated values such as `<description>` and `<name>` come directly from the text boxes associated with the HTML.

```
function addClickListener(map){
 map.addListener("longpress", function (evt){
  var data = getMarkerData();
  data.coords = map.pixelToGeo(evt.targetX , evt.targetY);
  currentMarker = addMarker(data);

  // If a description has been added, display the info bubble.
  if (data.description != ""){
   infoBubbles.addBubble(data.description, currentMarker.coords );
  }
 });
}
```


(C) Copyright Nokia 2013. All rights reserved.

```
function getMarkerData(){
 var markerData = {
  title : $('#title').val(),
  description : $('#description').val(),
  name : $('#name').val(),
  address: $('#address').val()
 };
 if ($('#icon').val() != ""){
  markerData.icon = $('#icon').val();
 } else {
  markerData.styleURL = $('#styleURL').val();
 }
 return markerData;
}
```

An alternative method of adding a marker is by adding an address. This uses the standard geoCoding service, and picks the first address found on the list. If your address is in the "wrong" format, just try making the address more general for now e.g.try the city only. The handler for geo-coding retrieves most of its information from the text boxes on the right hand side of the screen, but the address comes from the callback function.

```
var searchManager = nokia.places.search.manager;
searchManager.geoCode({
                  searchTerm : address.value,
                  onComplete: onSearchComplete});
```

```
  function onSearchComplete (data, requestStatus) {
 // If the search  has finished we can process the results
 if (requestStatus == "OK") {
  var markerData = getMarkerData()
  markerData.address = data.location.address.text;
  markerData.coords = data.location.position;
  // Ensure the address edit box contains the latest address contents.
  $("#address").val(data.location.address.text);
  currentMarker = addMarker(markerData);
  map.setCenter(data.location.position);

 } else if (requestStatus == "ERROR") {
  alert("The search request failed.");
 }
}
```

The `addMarkers()` method actually adds the marker, which itself contains a copy of its own `<description>`, `<address>`, `<name>` & etc. The marker has two events:

- a `click` event to display the data in text boxes for editing. This will also display an infobubble if the `<description>` field is set. A reference is kept on the `currentMarker` i.e. the last one selected.
- a `drag` event to remove the infobubble if a marker is repositioned. All markers are defined as draggable so the user can reposition them as necessary.

```
function addMarker(markerData) {
 var marker;

 if (markerData.title === undefined){
  markerData.title = "";
 }

 // Either an a Standard Marker or an icon, both are left
 // Draggable, so the location can be corrected.
 if (markerData.icon === undefined){
  marker = new nokia.maps.map.StandardMarker(markerData.coords, {
   text:  markerData.title, //small title
   draggable: true,  //the marker is marked  to be draggable
   $data: markerData
  });
 } else {
  marker =  new nokia.maps.map.Marker(markerData.coords,  {
  icon: markerData.icon,
   anchor: new nokia.maps.util.Point(16, 32),
   draggable: true,  //the marker is marked  to be draggable
   $data: markerData
  });
 }

 // When the marker is clicked, the edit boxes must be refreshed.
 marker.addListener("click" ,  function(evt) {
  currentMarker = evt.target;
  setMarkerData(currentMarker);
 });

 // If a marker is dragged, we need to close the infobox.
 marker.addListener("drag" ,  function(evt) {
  removeBubble();
  currentMarker = evt.target;
 });

 map.objects.add(marker);
 return marker;

}
```

## Editing Marker Data

The `editMarker()` method, removes the `currentMarker` and adds another one with the current data from the text fields.

```
function editMarker(addWaypoint) {
 // deletes the current marker
 map.objects.remove(currentMarker);
 removeBubble();
 if (addWaypoint == true){
  currentWaypoints.push( currentMarker.coordinate);
  $('#waypoints').text(JSON.stringify(currentWaypoints));
 } else {
  var data = getMarkerData();
  data.coords = currentMarker.coordinate;
  currentMarker = addMarker(data);
  // If a description has been added, display the info bubble.
```


(C) Copyright Nokia 2013. All rights reserved.

```
  if ( description.value != ""){
   infoBubbles.addBubble(data.description, data.coords );
  }
 }

}
```

### Adding Polylines

It is impossible to create `Polylines` without first having some defined `waypoints`. The `waypoints` are merely `geo.Coordinates`, so a button has been added to take the coordinates of the `currentMarker` and add them to an array. The marker is then removed to "tidy up" the display. When the create line from Waypoints button is pressed, the `Polyline` is created.

**Warning:** There is currently no method to delete `waypoints` or `Polylines`, this must be done by editing the **KML** directly.

```
var currentWaypoints = new Array();
```

```
map.objects.remove(currentMarker);
currentWaypoints.push( currentMarker.coordinate);
```

```
function createLineFromWaypoints(waypoints) {
 var lineData = new Object();
 lineData.lineNo = map.objects.getLength() + 1;
 lineData.waypoints = waypoints ;
 if( lineStyleURL.value != ""){
  lineData.styleURL =  lineStyleURL.value;
 }
 addLine(lineData);

 // Clear the current waypoint list and the info <span>
 currentWaypoints = new Array();
 $('#waypoints').text("");
}
```

## KML Generator

The `kmlGenerator.generateKML();` method has been taken from the converting from JavaScript to KML article. All the map Objects currently on screen are iterated through and added as **KML** equivalents. In order to discourage the use of inline styles, a `<styleURL>` element has been added. This should refer to a `<Style>` somewhere else in the document, but obviously the `<Style>` has not been created, and must be added manually. An example of how to add a `<styleURL>` and an associated `<Style>` is shown below:

```
<Style id='smallIcon'>
    <IconStyle>
       <Icon>
           <href>http:www.example.com/Icon-Small.png</href>
        </Icon>
    </IconStyle>
</Style>
<Placemark>
     <Point><coordinates>-0.12759880162775517,51.50320703163743,0</coordinates></Point>
     <styleUrl>#smallIcon</styleUrl>
</Placemark>
```

Since **KML** is just a form of XML, the generated file can be read using jQuery and applying the technique found in the Create map markers from XML data, the **KML** is in a known format, it is merely an exercise to extract the name from the `<name>` element, the address from the `<address>` element and so on. Two problems do need to be solved however. Firstly the geocoordinates are held as comma separated text, rather than individual `<longitude>` and `<latittude>` elements and secondly the description in the `<description>` element can contain formatted HTML and is held in a `CDATA` section. The first problem is dealt with by splitting the text using the `search()` function:

```
var lng = coord.substring( 0 , coord.search(","));
coord = coord.substring(coord.search(",")+ 1);
var lat = coord.substring( 0 , coord.search(","));
markerData.coords = new nokia.maps.geo.Coordinate(parseFloat(lat), parseFloat(lng));
```

The `CDATA` section is dealt with by stripping out the marker characters, and safely encoding the HTML data within it. The HTML is re-inflated using the `escape()` function when it is read.

```
while ( kml.search ("<!\\[CDATA\\[")  > - 1){
          var from = kml.search ("<!\\[CDATA\\[");
            var to= kml.search ("\\]\\]>");
            kml = kml.substring(0, from) +  escape(kml.substring(from + 9, to)) +  kml.substring(to+ 3);
  }
```

```
markerData.description =  unescape($(this).find('description').text());
```

## Summary

It should be easy to create a simple **KML** file using the technique described above. Editing can be done either directly on the source **KML** or by moving elements around on the map. The generated file is valid **KML**, and can be displayed using the standard technique described in the How to display KML file data on the map. The generated **KML** could also be added to an existing file (provided that the header and `<Document>` lines are not included.


(C) Copyright Nokia 2013. All rights reserved.

A full working example can be found at:

http://rawgithub.com/heremaps/examples/master/maps_api_for_javascript/demos/generate-kml/generate-kml-file-from-map.html


(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.