HERE Maps API - Advanced Routing

This article explains how to request routing information using the HERE Maps API and display routing details on screen. The structure of a route response is discussed in depth. Internationalization of the route instructions is also discussed.

Introduction

Note: Use of Nokia's routing service is subject to terms and conditions, specifically it is **prohibited** to provide real time turn-by-turn navigation services, however the routing service may be used in most applications, provided that the routing itself is not the primary functionality. The relevant section of the terms and conditions is repeated below:
4 (i) You will not: use or incorporate, without Nokia's prior written permission, the Service, Location API Developer Package or any part thereof, in connection with any Application or other service (a) which has the primary functionality of providing turn-by-turn navigation services, real time navigation or route guidance; or (b) where such Application's functionality is substantially similar to the HERE Maps or navigation/location-based products distributed by Nokia or its affiliates; or (c) which has the primary purpose of capturing or collecting end user data;

The full terms and conditions for the use of the location APIs can be found here

Although a simple point-to-point routing example may be found on the API explorer, it does not go further than displaying a route on screen. Further development is required in order to full utilize the power of the routing service and obtain the most from the API. Given that it is usually easier to modify an existing example rather than build one from scratch, this example attempts to fill a gap by providing a series of library functions to do the following:

- Calculate a route based on two or more **waypoints** as defined by parameters in the query string.
- Alternatively, calculate a route based on two or more **addresses** defined in the query string.
- Display the **shortest route** found on a map on screen.
- Interrogate the details of the route response and display **natural language** turn-by-turn instructions for each of the maneuvers required.
- Offer alternative information for an international audience (e.g displaying miles rather than kilometers, turn-by-turn instructions in different languages.)

Tip: Since **April 2012**, the HERE Maps website has now started supporting routing deep links, e.g. to find the route from Madrid to Marseille via Barcelona, use the following URL: http://maps.nokia.com/drive/Madrid/Barcelona/Marseille . Creating these URLs would allow you to swiftly incorporate a desktop and mobile friendly solution, without the need to build everything from scratch provided you don't need to keep a user on your website. The basic commentary for this Worked Example makes the assumption that you wish provide **your own** routing solution from scratch in order to keep the user on **your** website. The library functions provided allow you to create, style and place the generated routing instructions where you see fit.

Preliminary Preparation

Prior to making a routing request, we need at least two waypoints, a start point and a destination. These are supplied by the query string parameters.

Parsing Query Parameters

The reading in of query parameters]]uses a standard Library function called `getParameterByName()` has been created for splitting a query string into parts. This can used to extract addresses and/or geocoordinates from the request which can then be used as the input waypoints to calculate the route. For extended character sets (e.g. Chinese) it has been assumed that the addresses have been appropriately URI encoded, and therefore need to be *decoded* prior to use.

```
function getParameterByName(name) {
 name = name.replace(/[\[]/, "\\\[").replace(/[\]]/, "\\\]");
 var regex = new RegExp("[\\?&]" + name + "=([^&#]*)"),
 results = regex.exec(location.search);
 return results == null ? "" :
  decodeURIComponent(results[1].replace(/\+/g, " "));
}
```

It remains to decide the format of the query string to be parsed. Given that people tend to think in terms of addresses, it would be useful to accept a series of addresses in the format below:

```
http://www.example.com/?addr1=...&add2=...&addr3=...
```

However geocoding addresses (that is transforming them into latitude/longitude) is an inexact science, and no consideration has been given to the fact that the address may be misspelt or the address format may be wrong. It would be more accurate if geolocations could be specified by latitude and longitude directly, so the following format is also supported.

```
http://www.example.com/?lat1=...&long1=...&lat2=...&long2=...&lat3=...&long3=...
```

In order to support both of these waypoint formats, the first 10 parameters `addr0...addr9` and `lat/long0...lat/long9` are parsed in the function below.

```
addresses = new Array();
this.managersFinished = 0;
this.geoCodedWaypoints = new Array();
this.addressContainer = new nokia.maps.map.Container();
var expectedWaypoints = 0;

// Loop thorugh the first 10 parameters and see if the are addresses or geo-coordinates.
// Add them to the appropriate bucket.
for (var i = 0; i < 10; i++){
 var lat = getParameterByName('lat' + i);
 var lng = getParameterByName('long' + i);
 var addr = getParameterByName('addr' + i);
 if (addr !=""){
  addresses.push(getParameterByName('addr' + i));
  expectedWaypoints++;
 } else if (lat != "" && lng != ""){
  geoCodedWaypoints.push(new nokia.maps.geo.Coordinate(
   parseFloat(lat), parseFloat(lng)));
  expectedWaypoints++;
 }
}
```

Geocoding addresses

Tip: Use the latitude/longitude format where possible, as this reduces user error. Usually you will be able to supply at least one of the desired coordinates either by geocoding your destination or using geolocation to obtain the user's start point

If the user has supplied the waypoints as free text addresses, it is necessary to convert these into geo-coordinates before calculating the route. In order to reduce processing time, these calculations can be made concurrently. The details of how to make concurrent search requests has been discussed in a previous article. Whenever a geocode response has been obtained, an entry is added to the `geoCodedWaypoints` array. Once all the addresses have been processed, the map can be set up, and the calculation of the route invoked.

```
 var i = addresses.length;
while(i--) {
        nokia.places.search.manager.geoCode({
                 searchTerm :addresses[i],
  onComplete:  new searchManager(i).onSearchComplete
 });
}
```

```
function searchManager($index) {
 this.$index = $index;
 this.onSearchComplete = function (data, requestStatus) {
 // If the search  has finished we can process the results
 if (requestStatus == "OK") {
  geoCodedWaypoints [$index] =  data.location.position;
  addressContainer.objects.add(
   new nokia.maps.map.StandardMarker(
    geoCodedWaypoints [$index],{text: ($index + 1)}));
  //increment the counter to notify another manager has finished
  managersFinished++;
 } else if(requestStatus === "ERROR") {
  // we'll also increment in case of an error
  managersFinished++;
 }


 // if all managers are finished, we call the final function
 if(managersFinished === managersCount) {
  // hence we get the bounding box of the container
  var bbox = addressContainer.getBoundingBox();
  // if the bounding box is null then there are no objects inside
  // meaning no markers have been added to it
  if (bbox != null) {
   // we have at least one address mapped so we zoomTo it
    map.zoomTo(bbox);
  }
  makeRouteRequest(geoCodedWaypoints);
 }
}}
```

Making the route request and deciphering the result

Having obtained two waypoints, we can now make the route request. This consists of creating a router adding a callback function, making the request and obtaining the result. The code here is basically a repeat of the simple route example in the developer playground, and described in the Simple routing article. The main processing of the route occurs in the `onRouteCalculated()` method and will be discussed in greater detail below. In the example below, we are obtaining the shortest route for a vehicle - obviously the mode and type of transport may be altered as required.

```
var modes = [{
   type: "shortest",
   transportModes: ["car"],
   options: "avoidTollroad",
   trafficMode: "default"
}];
router = new nokia.maps.routing.Manager();
router.addObserver("state", onRouteCalculated);
router.calculateRoute(waypoints, modes);
```

The function `onRouteCalculated()` will be called when a route is calculated. Assuming a route has been returned we can:

- Display a polyline showing the route on the map. (this is the same as the Playground example)

- Display textual instructions of the list of each of the maneuvers which make up the route.
- Add a small marker on each maneuver point which shows the associated text instruction and more visual information (i.e. Junction view and turnpoint).

In order to understand the details of the route processing it is necessary to understand the anatomy of a route:

- A route consists of one or more **legs**. A leg is is the route between two of the specified waypoints. A simple route will only have one leg. A route with intermediate waypoints (i.e go via....) will have multiple legs.
- legs consist of a series of **maneuvers**. A maneuver is a location where the driver must decide on an action (e.g. turn left here). Each maneuver will have an associated location and an associated turn instruction.
- The route's **shape** defines all of the twists and turns that make up the route in order to display the polyline on screen. The geometry is a series of geolocations, but only some of them have an associated maneuver,

since a road may bend round a series of corners, but the driver can only make a decision to turn at a junction.

Taking the first returned `route`, we then need to walk through each `maneuver` in turn to extract the necessary routing information:

```
function onRouteCalculated (observedRouter, key, value){
 if (value == "finished") {
  $("#ticker").text( "");
  var routes = observedRouter.getRoutes();

  //create the default map representation of a route
  var mapRoute =
   new nokia.maps.routing.component.RouteResultSet(
    routes[0]).container; //first option found
  map.objects.add(mapRoute);
  directionsRenderer.setRoute(routes[0]);

  //Zoom to the bounding box of the route
  map.zoomTo(mapRoute.getBoundingBox(), false, "default");
 } else if(value == "failed") {
  alert("The routing request failed.");
 }
}
```

`directionsRenderer` is a separate map component, it has been set up as follows:

```
 function addDirectionsRenderer (map){
 extend(DirectionsRenderer,
  nokia.maps.map.component.Component);
 directionsRenderer = new DirectionsRenderer(document.getElementById("directions"));
 map.addComponent(directionsRenderer);
}
```

The method `setRoute()`, obtains and styles the instructions from the route:

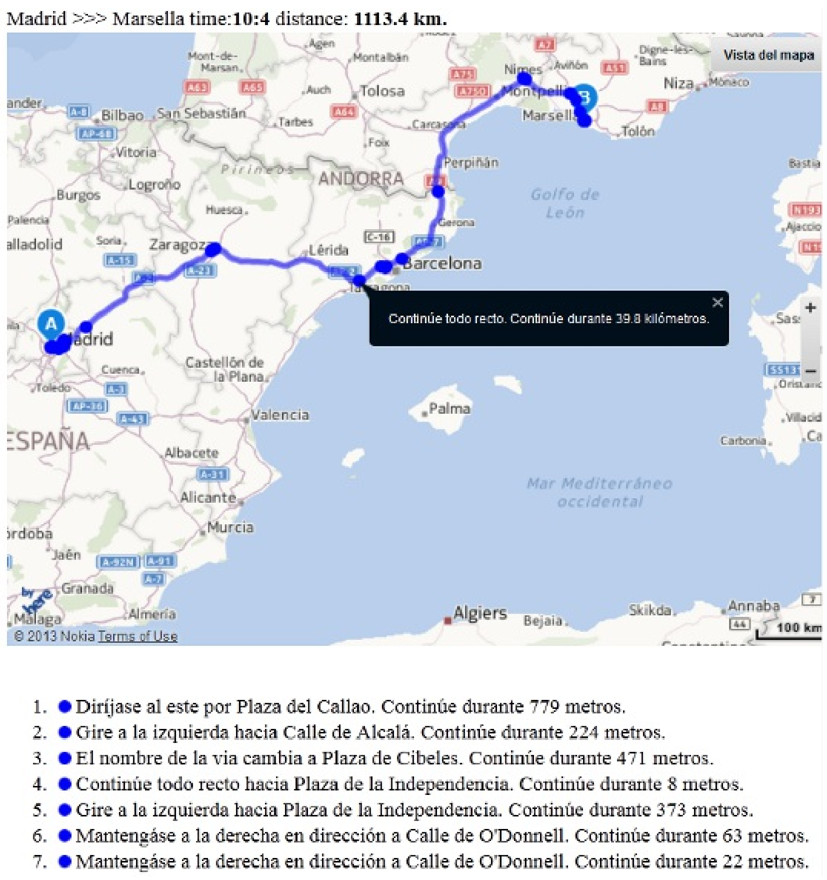
```
this.setRoute = function  (route) {

 if (nodeOL !== undefined){
  nodeOL.parentNode.removeChild(nodeOL);
  if ((this.bubble !== undefined)&&
   (this.bubble.getState() == "opened" )){
   this.bubble.close();
  }
 }
 nodeOL  = document.createElement("ol");
```

```
var showImperialUnits = that.showImperialUnits();

for (var i = 0;  i < route.legs.length; i++){
 for (var j = 0;  j <
  route.legs[i].maneuvers.length; j++){
  var details =  document.createElement("li");

  // Get the next maneuver.
  maneuver = route.legs[i].maneuvers[j];
  var instructions = maneuver.instruction;
  // For imperial measurements, extract the
  // distance span

    if (showImperialUnits == true){
      var ls = instructions.indexOf
        ("<span class=\"length\">")
      var ln = instructions.indexOf
        ("</span>" , ls);
      if (ls > -1 && ln > -1){
     distNode = instructions.substring(ls + 21, ln);
     var n=distNode.split(" ");
     if (n[1] == "meters"){
      imperialText =
        calculateDistance(n[0], false);
     } else {
       imperialText =
        calculateDistance(n[0] * 1000, false);
    }
     instructions = instructions.substring(0, ls + 21)
      +  imperialText + instructions.substring(ln);
      }
    }

    if (instructions.trim() != ""){
      // Finally add the instruction
      // to the list along with a link back to
      // infobubble.
    details.position = route.legs[i].maneuvers[j].position;
    details.onclick = function() {
      var infoBubbles = that.getInfobubbles();
      if (infoBubbles !== undefined){
      that.bubble = infoBubbles.openBubble(
       this.innerHTML , this.position);
     }
    };
    details.appendChild (document.createTextNode(' '));
    manueverText = addText(instructions);
    manueverText.className  ="manuever_instruction";
    details.appendChild (manueverText);

    nodeOL.appendChild(details);
    }
 }

}
this.panel.appendChild(nodeOL);
```

```
}
```

Summary

Using the library of functions supplied in the worked example it is possible to add an in-depth routing service to a web application.


Madrid >>> Marsella time:**10:4** distance: **1113.4 km.**

Continúe todo recto. Continúe durante 39.8 kilómetros.

1. ● Diríjase al este por Plaza del Callao. Continúe durante 779 metros.
2. ● Gire a la izquierda hacia Calle de Alcalá. Continúe durante 224 metros.
3. ● El nombre de la vía cambia a Plaza de Cibeles. Continúe durante 471 metros.
4. ● Continúe todo recto hacia Plaza de la Independencia. Continúe durante 8 metros.
5. ● Gire a la izquierda hacia Plaza de la Independencia. Continúe durante 373 metros.
6. ● Mantengáse a la derecha en dirección a Calle de O'Donnell. Continúe durante 63 metros.
7. ● Mantengáse a la derecha en dirección a Calle de O'Donnell. Continúe durante 22 metros.

A working example can be found here:

- Reading a Query String + Rendering Directions

Alternatively, a user can be forwarded directly to here.com as shown in the example below:

- Query String forwarding to HERE Maps

Furthermore, the directions rendering library can be used in multiple situations, working examples can be seen combining the library with the following

- Simple Directions Renderer example
- Routing and Directions via context menu input
- Routing and Directions based on Geocoding from a text box
- Routing and Directions based input from two Suggestions Boxes