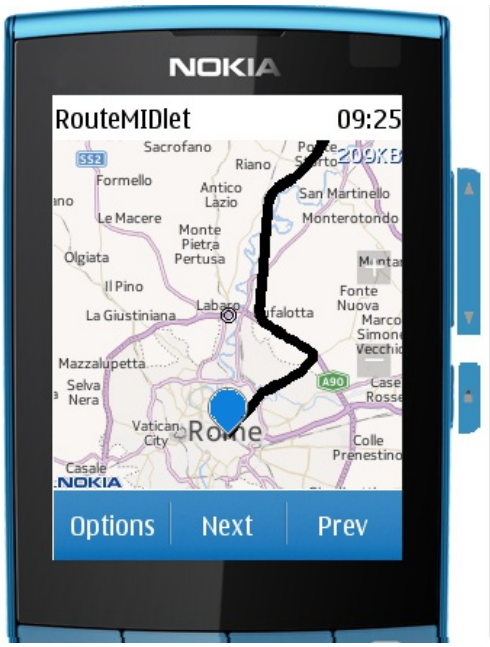
How to calculate and show a route with Maps API for Java ME

This article explains how to calculate a route between two points by using the HERE Maps API for Java ME, and how the route can be displayed on a map and how the user can select to display on the screen each of the critical turning points of the route in a Java ME application.

16 Oct 2011

Introduction



The HERE Maps API for Java ME offers routing features that allow developers to easily integrate this functionality into their Java ME apps.

RouteManager and RouteListener

The route functionality is based on two main classes:

- `RouteFactory`: this class is responsible for creating route requests, based on the waypoints and options passed as arguments
- `RouteListener`: the RouteListener interface gets notified by the `RouteRequest` when an **aysnchronous** operation completes or when an error is raised.

The RouteMIDlet base class

A base MIDlet that shows a map to the user, by using a `MapCanvas`, is the following one:

```
public class RouteMIDlet extends MIDlet  {

    MapCanvas mapCanvas = null;

    protected void startApp() throws MIDletStateChangeException {

        ApplicationContext ctx = ApplicationContext.getInstance();
        ctx.setAppID("MyAppId");
        ctx.setToken("MyToken");

        Display display = Display.getDisplay(this);
        mapCanvas = new RouteCanvas(display, this);

        display.setCurrent(mapCanvas);

 }
    protected void destroyApp(boolean unconditional)
            throws MIDletStateChangeException {
    }
    protected void pauseApp() {
    }
}
```

Implementing the RouteListener interface

The *RouteListener* interface defines two methods:

- `onRequestComplete(RouteRequest request, Route[] routes)`: called when a calculate route operation successfully completes

(C) Copyright Nokia 2013. All rights reserved.

- `onRequestError(RouteRequest request, Throwable error)`: called when an error occurs during an operation

Before an asynchronous RouteRequest can be made by the RouteFactory, the code must implement the RouteListener interface, so that it can get notified of the route result.

```
public class RouteCanvas extends MapCanvas implements RouteListener, CommandListener {
    [...]
}
```

The `onRequestComplete()` method returns an array of routes as an object argument, and it can be used to check if any routes were found, and to retrieve the actual route data. It is good practice in real world applications to let the user check and choose one of the available routes. Anyway, this example will simply use the first available route.

```
 public void onRequestComplete(RouteRequest request, Route[] routes) {
        if (routes.length > 0) {
            Route firstRoute = routes[0];
    }
}
```

The other method - *onRequestError()* needs to be implemented to complete the `RouteListener` interface declaration, and this to handle errors . A simple implementation is shown in the following code snippet:

```
  public void onRequestError(RouteRequest request, Throwable error) {
        Alert alertView = new Alert("Route search error: " + error.getMessage());
        display.setCurrent(alertView);
    }
```

Displaying the Route shape

The `Route` class defines a `getShape()` method that returns an array of GeoCoordinate objects. Those coordinates can be used to create a `MapPolyLine` object. Once the MapPolyline is created, it can be easily shown on a map by using the MapDisplay `addMapObject()` method.

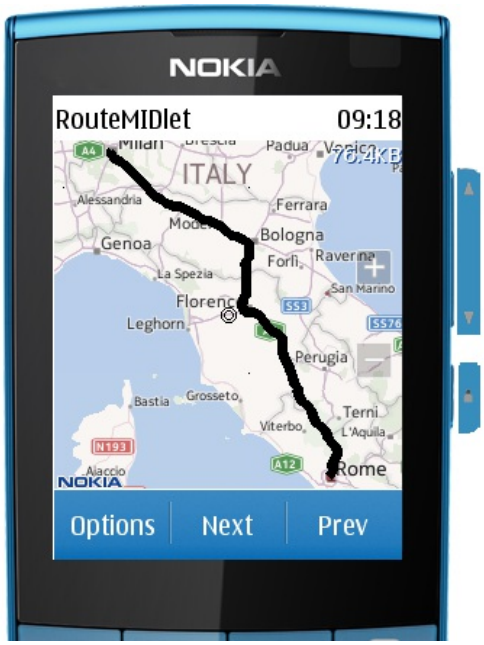
The `onRequestComplete()` can then be modified as follows:

```
public void onRequestComplete(RouteRequest request, Route[] routes) {

        if (routes.length > 0) {
            Route firstRoute = routes[0];
            map.addMapObject(mapFactory.createMapPolyline(firstRoute.getShape(), 3));
        }

    }
```

The Route shape is now displayed on the MapCanvas:



Displaying the maneuvers

A Route is composed of multiple maneuvers, that indicates the directions that the user should follow to navigate through the Route waypoints.

So, it would be useful to display those maneuvers directly on the map, and this can be done by using MapMarker objects, and by allowing the user to navigate the maneuvers one by one.

To do this, the RouteCanvas defines two different variables:

- a RouteManeuver object, that holds a reference to the currently displayed maneuver

(C) Copyright Nokia 2013. All rights reserved.

- a MapMarker object, holding a reference to the marker associated with the current maneuver

```
public class RouteCanvas extends MapCanvas implements RouteListener, CommandListener {

    [...]

    RouteManeuver currentManeuver = null;
    MapStandardMarker currentMarker = null;

    [...]
}
```

Now, a `setCurrentManeuver()` method can be defined, in order to:

- update the current maneuver reference
- display the marker associated with the newly set maneuver
- center the map in the associated geographical position

This is done as follows:

```
 private synchronized void setCurrentManeuver(RouteManeuver maneuver) {
        if (currentMarker == null) {
             currentMarker = mapFactory.createStandardMarker(maneuver.getPosition());
             map.addMapObject(currentMarker);
        } else {
            currentMarker.setCoordinate(maneuver.getPosition());
        }

        currentManeuver = maneuver;
        map.setCenter(currentManeuver.getPosition());
    }
```

The `onRequestComplete()` can be then modified in order to show the first maneuver on the map, with the following code:

```
setCurrentManeuver(firstRoute.getFirstManeuver());
```

Navigating through RouteManeuvers

Each RouteManeuver object has a `getNextManeuver()` and a `getPreviousManeuver()` method that makes it possible to navigate to the next or previous maneuver on the Route path. Those methods make it possible to implement a simple navigation functionality in the RouteMIDlet.

First, two Command variables are defined and added to the MapCanvas instance:

```
public class RouteCanvas extends MapCanvas implements RouteListener, CommandListener {

    [...]

    private final Command NEXT = new Command("Next", Command.OK, 2);
    private final Command BACK = new Command("Prev", Command.BACK, 3);

     public void onRequestComplete(RouteRequest request, Route[] routes) {

...
        addCommand(NEXT);
        addCommand(BACK);

    }
}
```

Then, it is necessary for the RouteMIDlet to implement the CommandListener interface. This means implementing the `commandAction()` method, that must handle the next and previous Commands, allowing the user to go to the following or preceding RouteManeuver on the Route path. This can be done as follows:

```
public void commandAction(Command c, Displayable d) {
        if (c == NEXT && currentManeuver.getNextManeuver() != null) {
            setCurrentManeuver(currentManeuver.getNextManeuver());
        } else if (c == BACK && currentManeuver.getPreviousManeuver() != null) {
            setCurrentManeuver(currentManeuver.getPreviousManeuver());
        }
    }
```

Using the route functionality

Now that a RouteListener interface has been completely implemented, it is possible to use the route features of the Java ME Location API, and specifically the `calculateRoute()` method. This method accepts at least two arguments:

- a `WaypointParameterList` object, consisting of a list of WayPoint objects that will be used to calculate the Route
- a `RoutingMode` array, with each RoutingMode object defining the parameters for one route calculation
- For **asynchronous** routing requests, an additional third argument is required holding the *RouteListener* itself

In this sample, the WaypoinParameterList holds the coordinates of two Italian cities, Rome and Milan:

```
WaypointParameterList waypoints = new WaypointParameterList();
waypoints.addCoordinate(new GeoCoordinate(41.90311, 12.49576, 0.0f));
waypoints.addCoordinate(new GeoCoordinate(45.46894, 9.18103, 0.0f));
```

The RoutingMode argument will hold the parameters to calculate a route based on the fastest one, using car as the transport mode:

(C) Copyright Nokia 2013. All rights reserved.

```
RoutingMode[] routingModes = new RoutingMode[1];
routingModes[0] = new RoutingMode(
        RoutingType.FASTEST,
        new int[]{TransportMode.CAR},
        new String[0]
);
```

The constructor of the *RouteCanvas* can then be modified to create a route request and call the `calculateRoute()` method with the above parameters:

```
public RouteCanvas(Display display, MIDlet midlet) {
        super(display);{

    [...]

       RouteFactory rf = RouteFactory.getInstance();
     rf.createRouteRequest().calculateRoute(waypoints, routingModes, this);

}
```

Resources

A MIDlet showing the above code in action can be downloaded here: File:RouteMIDlet.zip

Full source code of the RouteMIDlet is available here: File:RouteMIDlet.java.zip

Summary

This article shows some of the features of the routing functionality provided by the Map API for Java ME, such as the ability to display the Route shape on a Map and to access the RouteManeuvers with path indications and directions.

Many more routing features are available, and can be explored starting on the developer pages of the Map API for Java ME.

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.