Advanced Routing with Java ME

This article is an extension of the standard HERE Maps API for Java ME Routing Example, and explains how to use the RESTful Map API in order to display the turnpoint and junction view associated with each manuever in a route

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

A simple routing example is bundled with the HERE maps code examples, it searches for a route and displays it on screen. This code snippet extends the concept by adding a *Form* to display turn-by-turn instructions along with turnpoint images and junction views (where they exist) for each manuever.

Preparation

Firstly we need to set up a couple of `Command` buttons to cycle through the `manuevers` - the `CommandListener` for these buttons is held elsewhere, but the relevant code snippet has been added below.

```
public class TurnByTurnForm extends Form {
     // three Command buttons.
    final static Command BACK = new Command("Ok", Command.BACK, 1);
    final static Command PREVIOUS = new Command("Previous", Command.SCREEN, 1);
    final static Command NEXT = new Command("Next", Command.SCREEN, 1);
```

We also need to wire in the access to the `Form` from the main `RoutingDemo`

```
public class RoutingDemo  {
...etc

// Add a button to the new form...
 private final Command SHOW_TURN_BY_TURN = new Command("Turn by Turn", Command.OK, 3);
 private static TurnByTurnForm turnByTurn;
...

 protected void commandRun(Command c) {
...
    if (c == TurnByTurnForm.BACK) {
        display.setCurrent(this);/// i.e. return to showing previous display.
    }
    else if (c == TurnByTurnForm.NEXT) {
        turnByTurn.showNext();
    }
    else if (c == TurnByTurnForm.PREVIOUS) {
        turnByTurn.showPrevious();
```


(C) Copyright Nokia 2014. All rights reserved.

```
    }
    else if (c == SHOW_TURN_BY_TURN) {
        // Construct the Turn-by-Turn Form from a previously calculted Route.
 turnByTurn = new TurnByTurnForm(routes[0]);
 turnByTurn.setCommandListener(this);
        display.setCurrent(turnByTurn);
    }
...etc
}
```

Secondly we need to add a couple of constants to use the Map Image API (i.e. the base URL for junctions and the base URL for turnpoints, and extract some route information from the constructor. The relevant information we will need to maintain is the starting `manuever` and the overall shape of the `route`.

```
public class TurnByTurnForm extends Form {
...
    private static final String TURNPOINT_URL = "http://m.nok.it/turnpoint?r0=";
    private static final String JUNCTION_VIEW_URL = "http://m.nok.it/junction?r=";
    private final GeoCoordinate[] geometry;
    private final String widthHeightAndToken;
    private RouteManeuver currentManuever;

    public TurnByTurnForm(Route route) {
        super("");
        widthHeightAndToken = "&app_id=" + ApplicationContext.getInstance().getAppID()
                + "&token=" + ApplicationContext.getInstance().getToken()
                + "&h=" + ((int)(getHeight() * 0.8)) + "&w=" + getWidth();
        currentManuever = route.getFirstManeuver();
        geometry = route.getShape();
        showTurnByTurn();
        addCommand(BACK);
    }
```

Each `manuever` is linked to the next `manuever` in the chain by a doubly linked list, so it is simple traverse the `manuever` list using the `getNextManeuver()` and `getPreviousManeuver()` methods.

```
    public void showNext() {
        currentManuever = currentManuever.getNextManeuver();
        showTurnByTurn();
    }

    public void showPrevious() {
        currentManuever = currentManuever.getPreviousManeuver();
        showTurnByTurn();
    }
```

Placing manuever information on screen

The `showTurnByTurn()` method clears the `Display`, and then builds up an `Item` list consisting of the `manuever` instruction, the junction image and the turnpoint. The next and previous `Command` buttons are displayed where necessary, that is if there is another link left in the chain.

```
    private void showTurnByTurn() {
        deleteAll();
```


(C) Copyright Nokia 2014. All rights reserved.

```
        append(new StringItem("", currentManuever.getInstruction(), Item.PLAIN));
        addJunctionImage(currentManuever.getPosition(),
turnCodeToDirection(currentManuever.getDirection()));
        addTurnpoint(currentManuever.getPosition());
        if (currentManuever.getNextManeuver() != null) {
            addCommand(NEXT);
        } else {
            removeCommand(NEXT);
        }

        if (currentManuever.getPreviousManeuver() != null) {
            addCommand(PREVIOUS);
        } else {
            removeCommand(PREVIOUS);
        }
    }
```

Adding the Junction View Image

Adding a junction image is simple, we merely need to append the latitude and longitude of the current location to the base URL for a Junction View. If no junction is found, a single pixel gif will be returned. The size of the image is defined by the **h** and **w** parameters which have been defined to fill an appropriate section of the display based on the screen size. The app id and token which are set in the `ApplicationContext` are also sent to verify which application is requesting the junction view. Since App id and token are set on a per application basis, the one from the HERE Maps API for Java ME can be re-used.

```
    private void addJunctionImage(GeoCoordinate position, String turn) {
        Image im = downloadImage(JUNCTION_VIEW_URL + position.getLatitude()
                + "," + position.getLongitude() + turn
                + widthHeightAndToken);
        if (im != null) {
            append(
                    new ImageItem("", im, ImageItem.LAYOUT_DEFAULT, "", Item.PLAIN));

        }
    }
```

It should be possible to obtain the direction of the turn to add an arrow to the junction view.

```
    private String turnCodeToDirection(int turnCode) {
        String turn = "";
        switch (turnCode) {
            case Direction.BEAR_RIGHT:
            case Direction.HARD_RIGHT:
            case Direction.RIGHT:
            case Direction.LIGHT_RIGHT:
                turn = "&turn=r";
                break;
            case Direction.BEAR_LEFT:
            case Direction.HARD_LEFT:
            case Direction.LEFT:
            case Direction.LIGHT_LEFT:
                turn = "&turn=l";
                break;
            default:
                // Other instructions such as continue, or roundabouts won't have
```


(C) Copyright Nokia 2014. All rights reserved.

```
junction views.
                turn = "";
        }
        return turn;
    }
```

Adding the Turnpoint Image

Adding a turnpoint is slightly trickier since we need to send the path of the turn in the URL. This is achieved by scanning through the `geometry` of the entire `route`, and taking the `GeoCoordinates` of a few points around the location of the `manuever`. Again the height **h**, width **w**, app id and token can be appended as above.

Tip: It would also be possible to alter the base language of the turnpoint map, by appending a MARC language code based on the `defaultLanguage` attribute of the `ApplicationContext`

```
    private void addTurnpoint(GeoCoordinate position) {

        int indexOf = 0;
        while ((!geometry[indexOf].equals(position)) && (indexOf < geometry.length)) {
            indexOf++;
        }
        if (indexOf < 2) {
            indexOf = 0;
        } else if (indexOf + 6 > geometry.length) {
            indexOf = geometry.length - 6;
        } else {
            indexOf = indexOf--;
        }

        String URL = TURNPOINT_URL;
        for (int i = 0; i < 6; i++) {
            URL = URL + geometry[indexOf + i].getLatitude() + "," + geometry[indexOf +
i].getLongitude();
            if (i < 5) {
                URL = URL + ",";
            }
        }
        URL = URL + widthHeightAndToken;

        Image im = downloadImage(URL);
        if (im != null) {
            append(
                    new ImageItem("", im, ImageItem.LAYOUT_DEFAULT, "", Item.PLAIN));

        }
    }
```

Downloading the Images

The images can be downloaded using a standard helper function as shown below. It would be more efficient to cache these images rather than request new ones each time, but this has been left as a single request for simplicity.

```
    private Image downloadImage(String url) {

        Image im = null;
```


(C) Copyright Nokia 2014. All rights reserved.

```
        ContentConnection c = null;
        DataInputStream dis = null;

        if (url != null) {

            try {
                try {
                    c = (ContentConnection) Connector.open(url);
                    int len = (int) c.getLength();

                    dis = c.openDataInputStream();
                    if (len > 0) {
                        byte[] data = new byte[len];

                        dis.readFully(data);
                        im = Image.createImage(data, 0, data.length);
                    }
                } catch (IOException ioe) {// Failed to read the url. Can't do anything
about it, just don't
                    // update the image.
                } finally {
                    // Regardless of whether we are successful, we need to close
                    // Connections behind us. Basic Housekeeping.
                    if (dis != null) {
                        dis.close();
                    }
                    if (c != null) {
                        c.close();
                    }
                }
            } catch (IOException ioe) {// closure of connections may fail, nothing we
can do about it.
            }
        }

        return im;
    }
```

Summary

A lot of additional information can be extracted from a route request made through the Map API for Java ME. Additional pictorial information can be added using the RESTful Map API service.

(C) Copyright Nokia 2014. All rights reserved.

(C) Copyright Nokia 2014. All rights reserved.