Creating a Map Type Selector for the Maps API for Java ME

A map selector is a UI component for choosing the type of map to display (e.g. street map, terrain, satellite view etc.) This article explains how to create a Map Type Selector for both full-touch and non full-touch Java ME phones. An appropriate UI is dynamically created at run time to utilise Full Touch features where they are available. The code also demonstrates how to create a fallback option so simpler feature phones are also supported.

07 Oct 2012

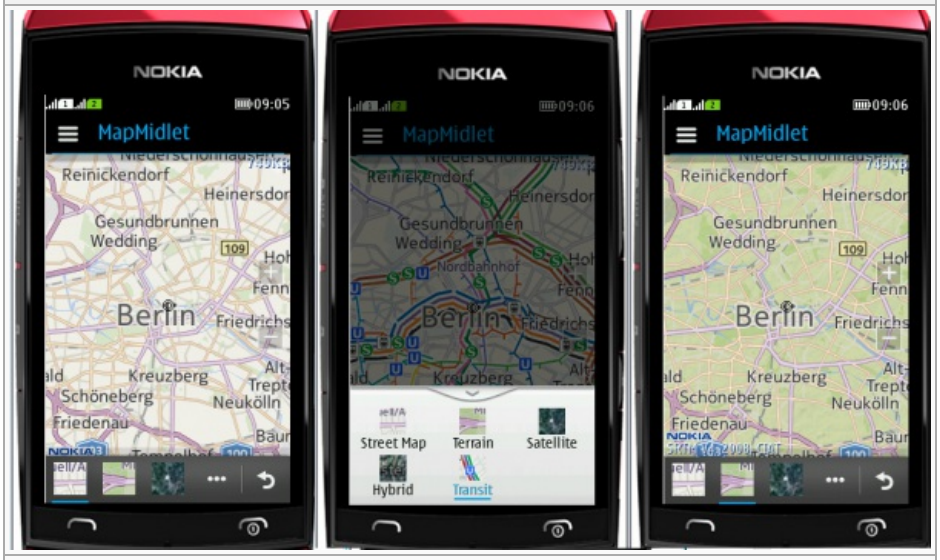
Note: This is an entry in the Asha Touch Competition 2012Q3

Tip: The latest 1.3 release of the HERE Maps API for Java ME now includes an integrated touchable component framework. This also includes the code for the Map Type Selector - see HERE Map Code Examples. These components are fully backwards compatible for older phones . The API has been integrated as a plug-in into the Asha SDK 1.0.

Introduction

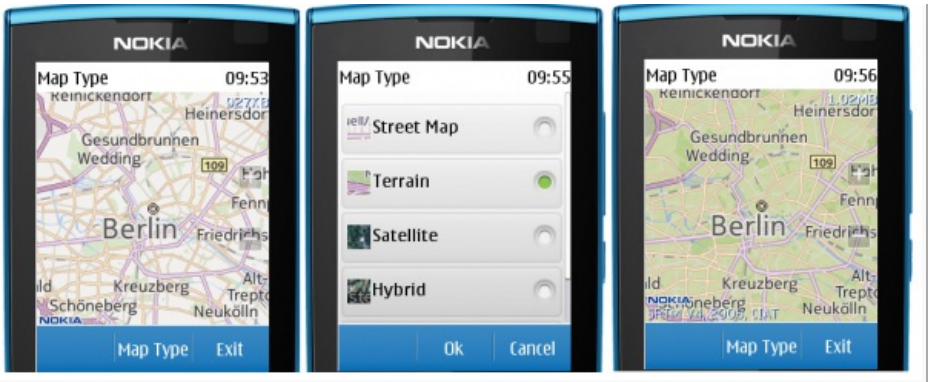
When creating map-based applications for Java ME phones, it is important to take into consideration the capabilities of the device. Users have come to expect an application which fully utilises the features of the phone in question. With the modern, full touch devices, it is no longer necessary to switch out the `Display` when making a selection from a `ChoiceGroup`. When the using the new `CategoryBar`, a user is able to make a selection directly from the icons shown on the `Display`. In the case of a map selector, the user will need to make a choice between one of the five standard supported map types.

This code example explains how to create a Map Type Selector and more importantly how to organise your code so that both Full Touch and non Full-Touch phones receive an appropriate UI. It should be noted that the Fallback UI requires an extra button press in order to see the result up on screen.

Full Touch UI

Screenshots of the Map Selector UI optimised for Full Touch phones.

Fallback UI

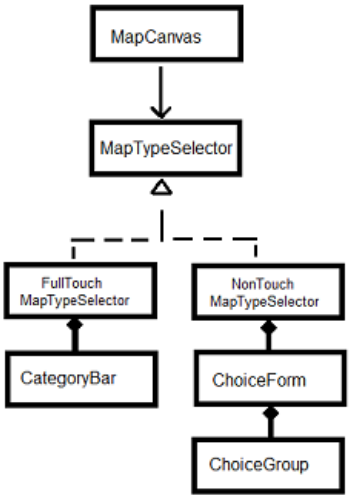
(C) Copyright Nokia 2014. All rights reserved.



Screenshots of the Map Selector UI optimised for non-Full Touch phones.

Defining the UI for Map Type Selection

The relationship between the two `MapTypeSelectors` can be seen in the class diagram below, a commentary on the design highlighting the use of each element follows.



Full Touch Map Type Selector

The core of the full touch example, is the use of the `CategoryBar`. A `CategoryBar` is created with five `Images` and appropriate labels (Street Map, Terrain, Satellite, Hybrid and Transit). The `FullTouchMapTypeSelector` itself handles icon presses, through implementing the `ElementListener` interface. When an icon is pressed, the base map type of the `MapDisplay` is altered based on the icon pressed. Since the Icons have been set up using the same order as defined in the`MapSchemeType` Enumerations, it is possible to switch by using the index directly.

```
public class FullTouchMapTypeSelector extends MapTypeSelector implements
  ElementListener {

 private CategoryBar cbar;
...
 public void notifyElementSelected(CategoryBar b, int index) {
  if (b == cbar) {
   mapCanvas.getMapDisplay().setBaseMapType(index);
  }
 }
}
```

A back button is achieved by creating an `IconCommand`. This is an extension of the standard `Command` button, and the `CommandListener` for this needs to be placed elsewhere - preferably where any other `Commands` are processed for the `MapDisplay`.

(C) Copyright Nokia 2014. All rights reserved.

The `FullTouchMapTypeSelector` has a delegate with the same signature which shows/hides the `CategoryBar` .

```
..
 private static final Command BACK = new IconCommand("Back", IconCommand.BACK, 1, 1);

 protected void setVisible(boolean visible) {

  if (visible) {
   mapCanvas.addCommand(BACK);
  } else {
   mapCanvas.removeCommand(BACK);
  }
  cbar.setVisibility(visible);

 }

 protected void commandAction(final Command c, Displayable d) {
  if (c == FullTouchMapTypeSelector.BACK) {
   setVisible(false);
  }
 }
```

Fallback UI

Since the `CategoryBar` is not available on older phones, it is necessary to switch out the `MapDisplay` and replace it with a simple modal dialog box. The user then makes an **exclusive** `ChoiceGroup` selection and if the `OK Command` is selected, the map type is updated as requested, and control passed back to the `MapDisplay`.

```
public class NonTouchMapTypeSelector extends MapTypeSelector implements
  CommandListener {

 private final ChoiceForm form;
 private final Display display;

 private final static Command OK = new Command("Ok", Command.OK, 1);
 private final static Command CANCEL = new Command("Cancel", Command.CANCEL,
   1);

 protected NonTouchMapTypeSelector(Display display) {

  super();
  this.display = display;
  form = new ChoiceForm();
  form.setCommandListener(this);

 }

...
 protected void setVisible(boolean visible) {
  if (visible) {
   display.setCurrent(form);
  } else {
   display.setCurrent(mapCanvas);
  }
 }
```


(C) Copyright Nokia 2014. All rights reserved.

```
...
 public void commandAction(Command c, Displayable d) {

  if (c == NonTouchMapTypeSelector.OK) {
   mapCanvas.getMapDisplay().setBaseMapType(form.getChoice());
  }
  setVisible(false);
 }
```

Initialising the Map Type Selector

As can be seen from the screenshots above, there is a high degree of similarity between the look-and-feel of the two map type selectors. Both of them have the same labels and images for the selection choices. The common set-up code has been abstracted down into a base class rather than repeating it twice. The images are initialised from resource files, and the label texts switched based on the locale of the device. The code example switches labels between English and Spanish, but can be extended to other languages as desired. In the example, the label texts are embedded as strings in the code, but they could equally be read from a **localised** JAD file

Tip: It is possible to obtain the default language of the **phone** by requesting the `"microedition.locale"` system property.

```
System.getProperty("microedition.locale");
```

This will retrieve a two letter MARC Code.

```
public abstract class MapTypeSelector {
...
        protected static final Image[] SELECTED_IMAGES = new Image[5];
 protected static final Image[] UNSELECTED_IMAGES = new Image[5];
 private static final String[] ENGLISH_LABELS = new String[] { "Street Map", "Terrain",
"Satellite", "Hybrid", "Transit" };
 private static final String[] SPANISH_LABELS = new String[] { "Callejero","Terreno",
"Satelite", "Hibrido", "Transporte Publico" };

 protected MapTypeSelector()   throws IOException{
                // Common initialisation goes here ...
  UNSELECTED_IMAGES[0] = Image.createImage("/normal.png");
  UNSELECTED_IMAGES[1] = Image.createImage("/terrain.png");
... etc
             }

private void setLanguage(String language) {
  if ("es-ES".equals(language)) {
   selector.setLabels((SPANISH_LABELS);
  else {
   selector.setLabels((ENGLISH_LABELS);
  }
 }
```

The instantiated `MapTypeSelector` itself is a singleton hidden behind a factory pattern, the factory attempts to create a `FullTouchMapTypeSelector` based on reflection using `Class.forName()`. If this fails, the `NonTouchMapTypeSelector` is used instead. The `new` operator cannot be used here because the application will fail on start-up on non Full Touch phones with a `NoClassDefFoundError`. By using reflection, the `NoClassDefFoundError` occurs at run time, where it can be caught and handled appropriately. The factory exposes a `Command` for showing the selector which can be added to a `Display`

(C) Copyright Nokia 2014. All rights reserved.

```
public abstract class MapTypeSelector {

       private Command commandButton;
       private static MapTypeSelector selector;
 protected MapTypeSelector()   throws IOException{
  // I'm a Singleton.
                UNSELECTED_IMAGES[0] = Image.createImage("/normal.png");
  UNSELECTED_IMAGES[1] = Image.createImage("/terrain.png");
...etc...
 }

public static final void init(Display display,
   MapCanvas mapCanvas) throws IOException {

  if (selector == null) {
   try {
    Class clazz = Class
      .forName("com.nokia.maps.selector.FullTouchMapTypeSelector");
    selector = (MapTypeSelector) clazz.newInstance();
   } catch (NoClassDefFoundError e) {
    selector = new NonTouchMapTypeSelector(display);
   } catch (Exception e) {
    // Class.forName potentially throws some fatal error
    // messages we won't handle them here for clarity, but wrap them
    // instead.
    throw new RuntimeException(e.getMessage()!= null ? e.getMessage(): e.toString());
   }

   // Base the menu texts on the locale of the device.

   selector.setLanguage(System.getProperty("microedition.locale"));
   selector.mapCanvas = mapCanvas;
  }

  return;

 }

public static final Command getCommand() {
  return selector.commandButton;
}
```

The `handleCommandAction()` method enables all the menu interaction of the `MapTypeSelector` to be self contained. Further `IconCommands` such as the **BACK** icon are delegated to the sub-class as necessary. An additional `toggle()` method is also exposed in case the developer wishes to instantiate the Map selector directly (e.g. using a touchable custom `MapComponent`)

```
public static final void handleCommandAction(final Command c, Displayable d) {
  if (selector != null) {
   if (c == commandButton) {
    selector.setVisible(true);
   } else {
    selector.commandAction(c, d);
   }
  }
 }
```


(C) Copyright Nokia 2014. All rights reserved.

```
 public static final void toggle() {
  selector.setVisible(!selector.isVisible());
 }
```

Wiring up a Map selector into an App

Adding a command Button

All that remains is to add the `MapSelector` to the map. The intialiser takes two parameters, the `Display` for screen switching and the `MapCanvas` itself. The `Command` created needs to be added to the `MapCanvas`.

```
MapTypeSelector.init(display, this);
addCommand(MapTypeSelector.getCommand());;
```

Also the `CommandListener` for the `MapCanvas` needs to allow the `MapTypeSelector` to show/hide itself when selected

```
public void commandAction(final Command c, Displayable d) {
...
   MapTypeSelector.handleCommandAction(c, d);
}
```

Adding a Map Component Button

It would also be possible to add the switch to the `MapCanvas` directly using a Map Component Button (see the videos below.) The extension to the code behind this example and other custom map components can be found in the Map Components project.

The media player is loading...

The media player is loading...

Summary

The code example encapsulates all the work required to change from one `MapSchemaType` to another. The `MapTypeSelector` created will be automatically optimised to use full touch controls where they exist. The decision to switch between Full Touch and non Full Touch UI components is made using reflection and delayed until runtime, allowing the same code base to be used by all phone types. This method described of combining a singleton, reflection and using factory pattern is not restricted to the `MapCanvas` and may also be used to extend the concept to optimise UIs for other applications.

(C) Copyright Nokia 2014. All rights reserved.