Creating a custom Map Item with Maps API for Java ME

This article explains how to implement a `CustomItem` that displays a map by using the HERE Maps API for Java ME

Introduction

The HERE Maps API for Java ME offers a `MapDisplay` class that is used to display a portion of a geographical map. By using the methods of this class, and specifically its `paint()` method, it is possible to build a `CustomItem` that embeds a `MapDisplay`, and so to show a geographical map in a Java ME Form screen.

Implementing the MapCustomItem

Since `MapCustomItem` extends the `CustomItem` class, its base skeleton can be written as:

```
public class MapCustomItem extends CustomItem
{
    MapDisplay map = null;
}
```

The `MapCustomItem` defines a `MapDisplay` variable that will hold a reference to the `MapDisplay` object used to display the geographical map.

The constructor

Now, it is necessary to define an explicit constructor, since the `CustomItem` class does not define a default one. The class constructor should initialize the `MapDisplay` object that will be used to show the map within the item itself: this can be done by using the `MapFactory` class, whose purpose is to instantiate `MapDisplay` and `MapObject` implementations.

When creating a `MapFactory` object, three parameters are necessary: the size of map tiles, the map width, and the map height. The tile size must be one of the values defined by the `MapResolutionEnum` class.

So, the arguments that can be passed to the `MapCustomItem` constructor can be:

- the Item label
- the map width
- the map height

And the constructor can be written as:

```
public MapCustomItem(String label, int width, int height)
{
    super(label);

    MapFactory mapFactory = MapFactory.createMapFactory(
            128,
            width,
```


(C) Copyright Nokia 2013. All rights reserved.

```
            height );

    this.map = mapFactory.createMapDisplay();
}
```

Implementing CustomItem methods

When implementing a `CustomItem`, some size-related methods need to be implemented, in order to tell what are the minimum and preferred size of the item itself. For the `MapCustomItem`, a trivial implementation could assume that both the preferred and minimum size are equal to the map width and height.

```
protected int getMinContentHeight() {
    return map.getHeight();
}

protected int getMinContentWidth() {
    return map.getWidth();
}

protected int getPrefContentHeight(int arg0) {
    return map.getHeight();
}

protected int getPrefContentWidth(int arg0) {
    return map.getWidth();
}
```

Painting the map

The `CustomItem paint()` method is responsible for painting the item content: since the `MapCustomItem` has to show the `MapDisplay` object, it would be enough to call its own `paint()` method, as shown below:

```
protected void paint(Graphics graphics, int width, int height)
{
    this.map.paint(graphics);
}
```

Exposing the MapDisplay object

Since it would be useful to get a reference to the `MapDisplay` object from outside the `CustomMapItem` class, in order to add some interactivity to the map, a new method is added:

```
public MapDisplay getMapDisplay() {
    return map;
}
```

Using the MapCustomItem

A base MIDlet that may want to use and display the `MapCustomItem` inside a `Form` can be written as follows.

```
public class MapFormMIDlet extends MIDlet implements ItemStateListener {

        MapCustomItem mapItem = null;

        protected void startApp() throws MIDletStateChangeException {

                ApplicationContext ctx = ApplicationContext.getInstance();

                ctx.setAppID("MyAppId");
                ctx.setToken("MyToken");

                Form f = new Form("Map Form");

                mapItem = new MapCustomItem("A map", 220, 220);

                mapItem.getMapDisplay().setZoomLevel(12, 0, 0);

                mapItem.getMapDisplay().setCenter(new GeoCoordinate(41.90311, 12.49576, 0.0f));

                f.append(mapItem);

                Display.getDisplay(this).setCurrent(f);
        }
        protected void destroyApp(boolean unconditional)
                        throws MIDletStateChangeException {

        }

        protected void pauseApp() {
        }
}
```

In the above code, the `getMapDisplay()` method is used to grab a reference to the `MapDisplay` object and call some of its own methods (for instance, to set a center and a default zoom level for the map).

Interacting with other Form items

A `Gauge` item can be added to the `Form` in order to let the user **change the zoom level of the map**. The `Gauge` maximum and initial values can be

(C) Copyright Nokia 2013. All rights reserved.

dynamically set by retrieving the maximum and current zoom values of the `MapDisplay` object (retrieved with the `MapCustomItem.getMapDisplay()` method).

Since the MIDlet has to be notified when the `Gauge` value changes, it also has to implement the `ItemStateListener` interface.

```
public class MapFormMIDlet extends MIDlet implements ItemStateListener
{
        MapCustomItem mapItem = null;

        Gauge zoomGauge = null;

        protected void startApp() throws MIDletStateChangeException
        {
                [...]

                int maxZoom = (int)mapItem.getMapDisplay().getMaxZoomLevel();
                int initialZoom = (int)mapItem.getMapDisplay().getZoomLevel();

                zoomGauge = new Gauge("Map zoom", true, maxZoom, initialZoom);

                f.append(zoomGauge);

                f.setItemStateListener(this);
        }
}
```

The `ItemStateListener` interface defines a method, `itemStateChanges()`, that must be implemented by the `MapFormMIDlet` class in order to detect changes to `Form` items. When the `Gauge` value gets changed by the user, this method has to modify the map zoom value, and this can be performed by using the `MapDisplay.setZoomLevel()` method, as shown in the code snippet below:

```
public void itemStateChanged(Item item)
{
    if(item == zoomGauge)
    {
        double newZoom = zoomGauge.getValue();

        mapItem.getMapDisplay().setZoomLevel(newZoom, 0, 0);
    }
}
```

Resources

A sample MIDlet showing the `MapCustomItem` in action is available here: File:JavaMELocationAPI MapCustomItem.zip

The source code of the `MapCustomItem` class is available for download here: File:MapCustomItem.java.zip

The source code of the `MapFormMIDlet` midlet is available for download here: File:MapFormMIDlet.java.zip

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.