How to create a draggable marker with Maps API for Java ME

This article explains how to create a `MapComponent` which allows your `MapMarkers` to be draggable.

Introduction

In the HERE Maps API for Java ME, by default, when a `MapMarker` is added to the map, it is a static marker. It is "burnt" into the underlying background of the map tile, so that the map itself is smoothly responsive to panning events. Whilst this is fine for external data where the positions of the data points cannot be altered by the user, it is not sufficient for input data points, where the user must be able to alter/correct the desired position. This article offers a mechanism to create smoothly draggable `MapMarkers` and allow the user to alter the locations, and explains the underlying issues behind the code.

Definition of the Issue

Like any `MapComponent`, the `MarkerDragger` extends the basic functionality through overriding a series of standard events. The required functionality can be summarised below:

- When `attach()` is called, the `MapComponent` must gain a reference to the underlying `MapDisplay`
- When `pointerPressed()` is called, the `MapComponent` must initialise the display to start the dragging action. This should provide feedback to the user.
- Whenever `pointerDragged()` is called, the `MapComponent` the marker must be dragged across the map.
- When `pointerReleased()` is called, the final location of the `MapStandardMarker` must be updated.
- The `paint()` method is used to provide updates to the display whilst the `MapComponent` is active.

Since the `MapComponent` uses pointer events, it needs to define its own `EventListener` to process these events.

Creating the `MarkerDragger MapComponent`

Attaching the map

Maintaining a reference to the currently attached map is such a common feature of a `MapComponent`, that the code is usually placed in an abstract base class (in this case called `AbstractMapComponent`). The protected variable `map` can then be used throughout the subclass `MapComponent`. The `AbstractMapComponent` also handles the `getVersion()` and `getId()` methods.

```
protected MapDisplay map;
```


(C) Copyright Nokia 2014. All rights reserved.

```
...
    public void attach(MapDisplay map) {
        this.map = map;
    }

...
    public void detach(MapDisplay map) {
        this.map = null;
    }
```

The constructor of the `MarkerDragger` extends the `AbstractMapComponent` class as shown. The initialization code also creates an `Image` mask which is used to display the "shadow" of a marker as it is being moved. Within the API jar itself, the image assets used to create the so-called `MapStandardMarker` are found in the `nma_res` folder. This is a standard location, and the assets will be present whenever the API is used. Re-using the standard assets means that no additional `Images` need to be bundled with the JAR when the `MarkerDragger` component is used.

Note: To extend the `MarkerDragger` to drag custom `MapMarkers` you could re-use the custom `Images` you have already bundled with your app

```
public class MarkerDragger extends AbstractMapComponent ... {

...
    private Image markerIcon;

...
    public MarkerDragger() {
        super(ID, VERSION);
     // load image resource from MIDlet's jar file
        try {
            markerIcon = Image.createImage("/nma_res/mask.png");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

Creating an `EventListener`

To react to events (other than the basic `paint()`, `attach` and `detatch()` methods) a `MapComponent` needs to offer an `EventListener`. This can be done by implementing the `EventListener` interface. The three key events (`keyPressed()`, `keyReleased()` and `keyRepeated()`) are not processed by the `MarkerDragger` and can be stubbed out to return `false`

```
public class MarkerDragger extends AbstractMapComponent implements EventListener {

...

    public EventListener getEventListener() {
        return this;
    }

    public boolean keyPressed(int keyCode, int gameAction) {
        return false;
    }
```


(C) Copyright Nokia 2014. All rights reserved.

```
    public boolean keyReleased(int keyCode, int gameAction) {
        return false;
    }

    public boolean keyRepeated(int keyCode, int gameAction, int repeatCount) {
        return false;
    }
```

Reacting to Events

The key to obtaining a smooth animation of the marker as it is dragged is to understand the difference between between drawing `MapObjects` on the `MapCanvas` and drawing `MapComponents` on the screen.

The process of rendering a map has three stages:

1. If necessary, any new map tiles are downloaded from the back-end server. For a typical Java ME application, the size of a map tile is 128 x 128 pixels.
2. Each map tile has any associated visible `MapObjects` burnt into the tile. These *map-tiles-with-embedded-objects* `Images` are used when the `MapCanvas` is displayed.
3. After the underlying `MapCanvas` is rendered, the `paint()` method of each attached `MapComponent` is called to add any additional visual components.

Through understanding this process, it should be clear that **moving** a `MapObject` across the `MapCanvas` will be slow and processor intensive, as a whole 128 x 128 pixel tile will need to be processed whenever the `Geocoordiate` of the `MapObject` is updated, where as rendering a 20 x 20 pixel "mask" using a `MapComponent` and just altering the anchor point of the x and y pixels should be relatively fast.

Tip: Updating `MapObjects` on the `MapCanvas` is **slow**, and should be avoided when possible.

When a `pointerPressed()` event occurs, the code checks to see if a `MapStandardMarker` is found at the current location and hides the marker. There is also a calculation to maintain the relative position of the pixel point pressed to the anchor point of the marker. Assuming we are dragging, the `paint()` method below merely draws an `Image` of the "mask" at the current location of the pointer. Only once the pointer is released is the actual `MapStandardMarker` moved. It shoudl be noted that the `pointerDragged()` method returns `true` whilst dragging to stop the panning of the underlying map, and that the current location of the pointer is only updated if it has moved a significant distance (more than 4 pixels) due to the potential scatter in the calculation of the pixel touched under a large object such as a finger.

```
public class MarkerDragger extends AbstractMapComponent implements EventListener {
...
    private MapStandardMarker draggableMarker;
    private Point currentPoint;
    private Point currentOffset;
...

    public void paint(Graphics g) {

        if (draggableMarker != null){
         g.drawImage(markerIcon,  currentPoint.getX() - markerIcon.getWidth()/2
           + currentOffset.getX(),
           currentPoint.getY()  - markerIcon.getHeight() +
           currentOffset.getY() , Graphics.TOP | Graphics.LEFT);;
        }
    }


    public boolean pointerDragged(int x, int y) {
        if (draggableMarker != null) {
```


(C) Copyright Nokia 2014. All rights reserved.

```
            int dx = currentPoint.getX() - x;
            int dy = currentPoint.getY() - y;

            if ((dx * dx) + ((dy * dy)) > 16) {
                currentPoint = new Point(x, y);
            }
        }
        return (draggableMarker != null);
    }


    public boolean pointerPressed(int x, int y) {
        currentPoint = new Point(x, y);
        MapObject mapObject = map.getObjectAt(currentPoint);

        if (mapObject != null && mapObject instanceof MapStandardMarker) {
            draggableMarker = (MapStandardMarker) mapObject;
            Point markerFocus = map.geoToPixel(draggableMarker.getCoordinate());
            currentOffset = new Point( markerFocus.getX() - currentPoint.getX(),
              markerFocus.getY() - currentPoint.getY());
            draggableMarker.setVisible(false);
        }
        return (draggableMarker != null);
    }


    public boolean pointerReleased(int x, int y) {
        if (draggableMarker != null) {
           Point markerFocus = new Point (currentPoint.getX() +  currentOffset.getX(),
             currentPoint.getY() +  currentOffset.getY() );
            draggableMarker.setCoordinate(map.pixelToGeo(markerFocus));
            draggableMarker.setVisible(true);
            draggableMarker = null;
            return true;
        }
        return false;
    }
}
```

Using the MarkerDragger Map Component

The `MarkerDragger` can be attached to the map as shown below. Once attached, it will allow a user to drag any an instance of a `MapStandardMarker` to a new position.

```
    public MarkerExample(Display display, MIDlet midlet) {
       ...
            map.addMapComponent(
                    new MarkerDragger());
        }
```

Summary

Adding the ability to drag a `MapStandardMarker` is a relatively simple procedure. To maintain the usability of the UI however, care needs to be take avoid unnecessarily updating the `Geocoordinate` of the `MapStandardMarker` whilst dragging. It must be

(C) Copyright Nokia 2014. All rights reserved.

remembered that the `pointerDragged()` event is supplied by the device itself, and the rate of these events could exceed the ability of the API to supply map tiles to process `paint()`. Using a `MapComponent` alleviates this problem, since a much smaller area needs to be updated to process each `paint()`.

(C) Copyright Nokia 2014. All rights reserved.