Implementing Pinch zooming with Gesture API for Maps in Java ME

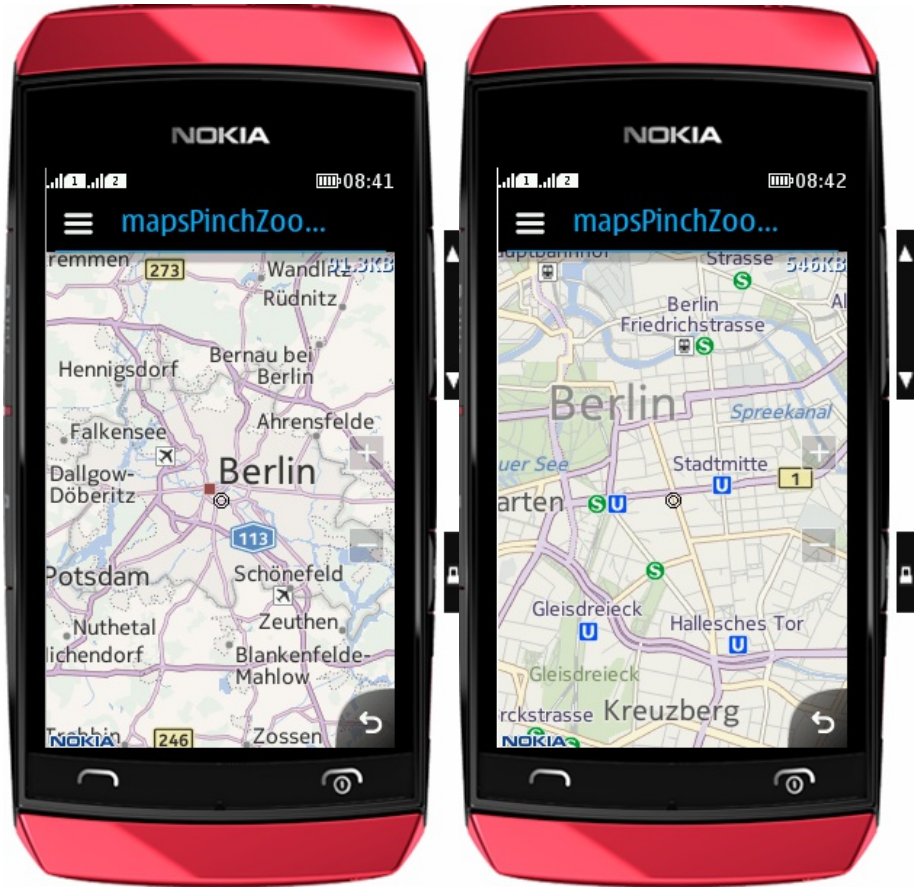
This article explains how to implement own pinch zooming (zoom in and out) support for maps implemented with Maps API in Series 40 Java 2.0 runtime.

29 Jul 2012

Archived: This article is archived because it is not considered relevant for third-party developers creating commercial solutions today. If you think this article is still relevant, let us know by adding the template {{ReviewForRemovalFromArchive|user=~~~~|*write your reason here*}}.

The article is believed to be still valid for the original topic scope.

Introduction



The Maps API for Java ME makes it possible to embed Nokia maps into Java ME applications. However the Maps API is missing pinch zoom functionality. If Java ME applications want to support pinch zoom functionality in maps then they have to implement custom pinch zooming in their application. Note that this custom pinch zooming works only on Series 40 full touch devices and will not work on other devices.

Enabling Pinch zoom

Generally there are two ways to implement pinch action, one being MultiPoint touch events and other one is Gesture API. Using Gesture API is the right choice since Gesture API provides ready-made pinch implementation. Pinch zooming support for maps in Series 40 Java can be implemented using Gesture API . Pinch zooming happens in Series 40 devices when two fingers are pressed on screen and one finger is moved near to other finger (pinch zoom in) or moved away (pinch zoom out).

Gesture event handling

To implement support for pinch zoom in the map, a gesture event of pinch has to be registered and handled inside *MapCanvas* class, which would result in MapCanvas class looking like the following:

```
public class MyMapCanvas extends MapCanvas implements GestureListener,… {
 …
private GestureInteractiveZone gestureZone;
 public MyMapCanvas(Display display, MIDlet midlet) {
  super(display);
…
  GestureRegistrationManager.setListener(this, this);
```


(C) Copyright Nokia 2013. All rights reserved.

```
  // Register for pinch events in the whole canvas area
  GestureInteractiveZone gestureZone = new GestureInteractiveZone(
    GestureInteractiveZone.GESTURE_PINCH);
  GestureRegistrationManager.register(this, gestureZone);
…
 }
public void gestureAction(Object arg0, GestureInteractiveZone arg1,
   GestureEvent arg2) {
...
}
...
}
```

Pinch zooming is actually handled by calculating the difference of the distances between start and end point of moving finger. When pinch is happened, *gestureAction(...)* method will be called. Check the *GestureInteractiveZone.GESTURE_PINCH* type and do the calculation for pinch zoom. Distance is calculated based on the difference of distance between current pinch distance (distance between two fingers after pinch) and starting pinch distance (distance between two fingers before pinch). 100 pixels difference is used as one zoom level change, and the max/min zoom value is checked before setting the zoom. Finally the map is set to the original center.

```
public void gestureAction(Object arg0, GestureInteractiveZone aGestureZone,
   GestureEvent aGestureEvent) {
  int eventType = aGestureEvent.getType();
  switch (eventType) {
  case GestureInteractiveZone.GESTURE_PINCH: // Pinch detected
   int curPinchDistance = aGestureEvent.getPinchDistanceCurrent();
   int startingPinchDistance = aGestureEvent
     .getPinchDistanceStarting();
   int zoomNew = zoomOrg
     + (int) ((curPinchDistance - startingPinchDistance) / 100);
   if (zoomNew > getMapDisplay().getMaxZoomLevel()) {
    zoomNew = (int) getMapDisplay().getMaxZoomLevel();
   }
   if (zoomNew < getMapDisplay().getMinZoomLevel()) {
    zoomNew = (int) getMapDisplay().getMinZoomLevel();
   }
   if (zoomNew != zoomOrg) {
    getMapDisplay().setZoomLevel(zoomNew, 0, 0);
    getMapDisplay().setCenter(orgCenter);
   }
   zoomOrg = getMapDisplay().getZoomLevel();
   break;
  }
 }
```

Resources

Full source codes for the example can be found from File:JavaMeMaps Gesture PinchZoom.zip

Summary

The Maps API for Java ME has much more functionality which makes it possible to integrate all Nokia Maps features into Java ME applications.

(C) Copyright Nokia 2013. All rights reserved.

 (C) Copyright Nokia 2013. All rights reserved.