How to use ProximityListener in Java ME

Overview

The Location API (JSR-179) has been part of Java implementations in S60 devices since S60 3rd Edition and in Series 40 devices since Series 40 6th Edition. The API provides classes for getting coordinates from a GPS receiver and also for accessing the phone's Landmarks (for both viewing and editing). The ProximityListener is one of the Location API features, that is not mandatory. It has not been supported in Location API implementations in Nokia phones before Symbian^3 (JRT 2.1). At the moment (October 2011) it is not supported in Series 40 devices. The class represents a listener to events associated with detecting proximity to some registered coordinates.

The code sample below shows a simple way of registering a ProximityListener to a MIDlet:

```
Form f=new Form("");
Display.getDisplay(this).setCurrent(f);
double latitude=1;    //Latitude
double longitude=1;   //Longitude
float altitude=1;     //Altitude
float proxRadius=500; //Proximity Radius
Coordinates coordinates=new Coordinates(latitude, longitude, altitude);

try {
    provider = LocationProvider.getInstance(null);
    if (provider != null) {
        provider.setLocationListener(this, -1, -1, -1);
        LocationProvider.addProximityListener(this, coordinates, proxRadius);
    }
    else
      f.append("LocationProvider = null!");
} catch (LocationException le) {
    f.append("LocationException! "+"ProximityListener might not be supported: " + le.getMessage());
}
```

In the code sample above, **this** refers to the MIDlet that should implement the ProximityListener interface. The Location Exception is thrown when registering a ProximityListener for tracking the distance to the registered coordinates is not supported by the terminal.

Usage of the ProximityMIDlet

The MIDlet has four classes:

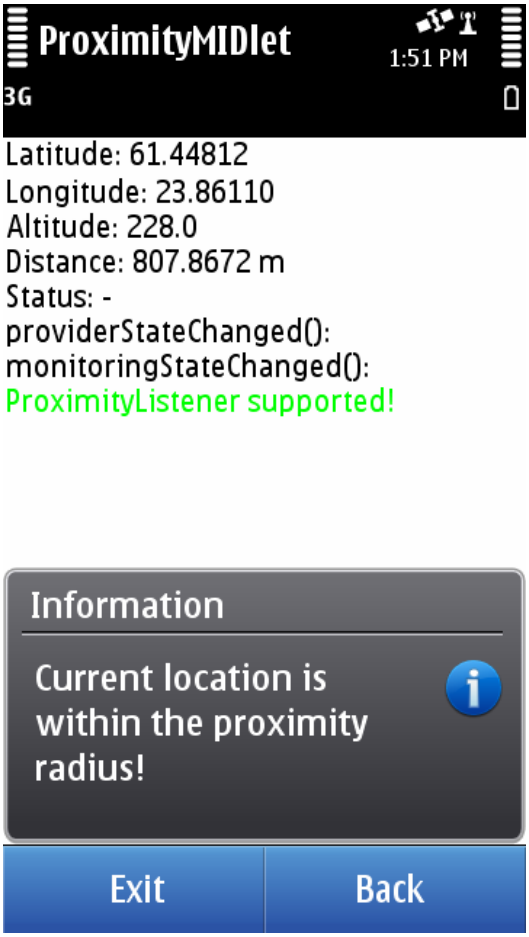
- ProximityMIDlet, the main MIDlet class
- LandmarkForm, screen for selecting a landmark, whose coordinates are used for the proximity listener
- LocationCanvas, screen for showing coordinates and other info about the application status
- MovementTracker, implements LocationListener and ProximityListener, takes care of all the Location API-related tasks

The idea of the application is to select one of the landmarks already saved in the device as the center point for registering the proximity listener. The user can set the distance from the center point within which proximity is defined. In order to use this application, there must be at least one saved landmark in the device. These can be easily created for example by using the Nokia Maps application, or they can be sent from other devices as multimedia message attachments.

When the user approaches the center point within the defined proximity distance, an Alert is shown to the user. The MIDlet then calls the proximityEvent method. Note, that the proximityEvent method is called only once when the terminal enters the proximity area of the registered coordinates. The proximity registration for these coordinates is cancelled as soon as the the proximityEvent is called. If the application should be notified again, a new ProximityListener registration is required.

The images below show some screenshots from the MIDlet running in Nokia N8:

(C) Copyright Nokia 2013. All rights reserved.



Below are source code for the two most important classes, LandmarkForm.java and MovementTracker.java.

Source code: LandmarkForm.java

```
import javax.microedition.lcdui.*;
import javax.microedition.location.AddressInfo;
import javax.microedition.location.Coordinates;
import javax.microedition.location.Landmark;
import javax.microedition.location.LandmarkStore;
import javax.microedition.location.LocationException;
import javax.microedition.location.QualifiedCoordinates;
import java.util.Enumeration;
import java.util.Vector;
import java.io.IOException;

public class LandmarkForm extends Form implements CommandListener, ItemStateListener {
    private ProximityMIDlet midlet;
    private LocationCanvas locationCanvas;
    private Command setCommand;
    private Command exitCommand;
    private Enumeration enumeration;
    private ChoiceGroup landmarks;
    private StringItem description, street, latitude, longitude;
    private Vector descriptions;
    private Vector streets;
    private Vector latitudes;
    private Vector longitudes;
    private Gauge radiusGauge;

    public LandmarkForm(String title, ProximityMIDlet midlet) {
        super(title);
        this.midlet = midlet;
        setCommand = new Command ("Set", Command.OK, 1);
        exitCommand = new Command("Exit", Command.EXIT, 1);
        this.addCommand(setCommand);
        this.addCommand(exitCommand);
        this.setCommandListener(this);
        radiusGauge = new Gauge("Proximity radius (m)", true, 5000, 500);
        this.append(radiusGauge);
        landmarks = new ChoiceGroup("Landmarks", Choice.POPUP);
        this.append(landmarks);
        this.setItemStateListener(this);
        descriptions = new Vector();
        streets = new Vector();
        latitudes = new Vector();
        longitudes = new Vector();
        getLandmarks();
        if (landmarks.size() > 0) {
            description = new StringItem("Description:", "" + descriptions.elementAt(landmarks.getSelectedIndex()));
            this.insert(2, description);
            street = new StringItem("Street address:", "" + streets.elementAt(landmarks.getSelectedIndex()));
            this.insert(3, street);
            latitude = new StringItem("Latitude:", "" + latitudes.elementAt(landmarks.getSelectedIndex()));
            this.insert(4, latitude);
            longitude = new StringItem("Longitude:", "" + longitudes.elementAt(landmarks.getSelectedIndex()));
```


(C) Copyright Nokia 2013. All rights reserved.

```
            this.insert(5, longitude);
        }
    }

    public void commandAction(Command c, Displayable d) {
        if (c == setCommand) {
            try {
                Double lat = (Double)latitudes.elementAt(landmarks.getSelectedIndex());
                Double lon = (Double)longitudes.elementAt(landmarks.getSelectedIndex());

                midlet.target = new Coordinates(lat.doubleValue(), lon.doubleValue(), Float.NaN);
                locationCanvas = new LocationCanvas(midlet);
                Display.getDisplay(midlet).setCurrent(locationCanvas);
            } catch (LocationException le) {
                midlet.showError("LocationException", le.getMessage());
            }
        }
        if (c == exitCommand) midlet.notifyDestroyed();
    }

    /**
     * This method reads the landmarks from the default landmark store and appends
     * them on the form.
     * @param form The form, where the landmarks are listed.
     */
    protected void getLandmarks() {
        try {
            LandmarkStore store = LandmarkStore.getInstance(null);
            if (store != null) {
                enumeration = store.getLandmarks();
                if (enumeration == null) return;
                int i = 1;
                while (enumeration.hasMoreElements()) {
                    Landmark landmark = (Landmark)enumeration.nextElement();
                    String name = landmark.getName();
                    String descr = landmark.getDescription();
                    if (descr == null) descr = "N/A";
                    descriptions.addElement(descr);
                    QualifiedCoordinates qCoordinates = landmark.getQualifiedCoordinates();
                    AddressInfo addressInfo = landmark.getAddressInfo();
                    if (addressInfo != null) {
                        String streetString = addressInfo.getField(AddressInfo.STREET);
                        if (streetString == null) streetString = "N/A";
                        streets.addElement(streetString);
                    }
                    double lat = qCoordinates.getLatitude();
                    latitudes.addElement(new Double(lat));
                    double lon = qCoordinates.getLongitude();
                    longitudes.addElement(new Double(lon));
                    landmarks.append(name, null);
                    i++;
                }
            }
        } catch (IOException ioe) {
            midlet.showError("getLandmarks(): IOException", ioe.getMessage());
        } catch (SecurityException se) {
            midlet.showError("getLandmarks(): SecurityException", se.getMessage());
        }
    }

    public void itemStateChanged(Item item) {
        if (item.equals(landmarks)) {
            description = new StringItem("Description:", landmarks.getString(landmarks.getSelectedIndex()));
            this.set(2, description);
            street = new StringItem("Street address:", "" + streets.elementAt(landmarks.getSelectedIndex()));
            this.set(3, street);
            latitude = new StringItem("Latitude:", "" + latitudes.elementAt(landmarks.getSelectedIndex()));
            this.set(4, latitude);
            longitude = new StringItem("Longitude:", "" + longitudes.elementAt(landmarks.getSelectedIndex()));
            this.set(5, longitude);
        }
        if (item.equals(radiusGauge)) {
            midlet.radius = radiusGauge.getValue();
        }
    }
}
```

Source code: MovementTracker.java

```
import javax.microedition.location.Coordinates;
import javax.microedition.location.LocationListener;
import javax.microedition.location.LocationProvider;
import javax.microedition.location.Location;
import javax.microedition.location.LocationException;
import javax.microedition.location.QualifiedCoordinates;
import javax.microedition.location.ProximityListener;

class MovementTracker implements LocationListener, ProximityListener {
    private LocationProvider provider;
    private UpdateHandler handler;
    private boolean done;
    protected boolean proximity = false;
    protected boolean active = false;
    private ProximityMIDlet midlet;
    private LocationCanvas canvas;
    protected String statusString = "-";

    public MovementTracker(ProximityMIDlet midlet, LocationCanvas canvas) {
        this.canvas = canvas;
        this.midlet = midlet;
        done = false;
    }

    public void start() {
        handler = new UpdateHandler();
        new Thread(handler).start();
        try {
            provider = LocationProvider.getInstance(null);
            if (provider != null) {
                provider.setLocationListener(this, -1, -1, -1);
                LocationProvider.addProximityListener(this, midlet.target, midlet.radius);
                proximity = true;
            }
            else midlet.showError("Error", "LocationProvider = null!");
        } catch (LocationException le) {
```


(C) Copyright Nokia 2013. All rights reserved.

```
            midlet.showError("LocationException", "ProximityListener might not be supported: " + le.getMessage());
            le.printStackTrace();
            proximity = false;
        }
    }

    /**
     * Sets the proximity listener again by using the previously used values for
     * coordinates and radius.
     */
    protected void setProximityListener() {
        try {
            if (provider == null) provider = LocationProvider.getInstance(null);
            if (provider != null) {
                provider.setLocationListener(this, -1, -1, -1);
                LocationProvider.addProximityListener(this, midlet.target, midlet.radius);
                proximity = true;
            }
            else midlet.showError("Error", "LocationProvider = null!");
        } catch (LocationException le) {
            midlet.showError("LocationException", "ProximityListener might not be supported: " + le.getMessage());
            le.printStackTrace();
            proximity = false;
        }
    }

    /**
     * Called by the LocationProvider to which this listener is registered.
     * This method will be called periodically according to the interval defined
     * when registering the listener to provide updates of the current location.
     * @param provider the source of the event
     * @param location the location to which the event relates, i.e. the new position
     */
    public void locationUpdated(LocationProvider provider, Location location) {
        handler.handleUpdate(location);
    }

    public void providerStateChanged(LocationProvider provider, int newState) {
        canvas.providerString = "providerStateChanged(): " + newState;
        canvas.repaint();
    }

    /**
     * After registering this listener with the LocationProvider, this method will be called by the
     * platform when the implementation detects that the current location of the terminal is within
     * the defined proximity radius of the registered coordinates.
     *
     * NOTE: The listener is called only once when the terminal enters the proximity of the registered coordinates.
     * The registration with these coordinates is cancelled when the listener is called. If the application wants
     * to be notified again about these coordinates, it must re-register the coordinates and the listener (see
     * setProximityListener() method).
     * @param coordinates the registered coordinates to which proximity has been detected
     * @param location the current location of the terminal
     */
    public void proximityEvent(Coordinates coordinates, Location location) {
        midlet.showInfo("Information", "Current location is within the proximity radius!");
        statusString = "Current location is within the proximity radius!";
        canvas.repaint();
        canvas.addCommand(canvas.setCommand);
        //setProximityListener();
    }

    /**
     * Called to notify that the state of the proximity monitoring has changed.
     * @param isMonitoringActive a boolean indicating the new state of the proximity monitoring
     */
    public void monitoringStateChanged(boolean isMonitoringActive) {
        canvas.monitoringString = "monitoringStateChanged(): " + isMonitoringActive;
        canvas.repaint();
    }

    class UpdateHandler implements Runnable {
        private Location updatedLocation = null;
        // The run method performs the actual processing of the location updates
        public void run() {
            while (!done) {
                synchronized(this) {
                    if (updatedLocation == null) {
                        try {
                            wait();
                        } catch (InterruptedException e) {// Handle interruption
                        }
                    }
                    updatedLocation = null;
                }
            }
        }

        /**
         * Handles the updated location, updates the coordinate Strings on the
         * Canvas and prints out the location's extra info.
         * @param update updated location
         */
        public synchronized void handleUpdate(Location update) {
            updatedLocation = update;
            QualifiedCoordinates coordinates = update.getQualifiedCoordinates();
            if (coordinates != null) {
                canvas.latitude = coordinates.getLatitude();
                canvas.longitude = coordinates.getLongitude();
                canvas.altitude = coordinates.getAltitude();
                canvas.distance = coordinates.distance(midlet.target);
            }
            if (canvas.latitude == 0) canvas.latitudeString = "Latitude: Wait...";
            else canvas.latitudeString = "Latitude: " + canvas.latitude;
            canvas.longitudeString = "Longitude: " + canvas.longitude;
            canvas.altitudeString = "Altitude: " + canvas.altitude;
            canvas.distanceString = "Distance: " + canvas.distance;
            canvas.repaint();
            notify();
        }
    }
}
```

Example application

- ProximityMIDlet.zip containing ProximityMIDlet.jad and ProximityMIDlet.jar

(C) Copyright Nokia 2013. All rights reserved.

See also

- Java ME Location API articles in Nokia Developer wiki
- ProximityListener in Java Developer's Library

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.