Best practises for listening to location updates with Java ME

This article describes best practices for making an application listen frequently for location updates, when using the Location API (JSR-179). There are two common ways of retrieving the device's current location, by either using the getLocation() method within a loop or by setting a location listener. The practical differences between these two methods are described in the code examples below, and suggestions are given as to which one should be preferred. Also some general guidelines are given when the application no longer needs to be location aware, in order to save power consumption. Important information is also given, regarding the Cell-ID based location retrieval.

Setting a Location Listener

The location API (JSR-179) provides a built-in interface, for listening for any location updates. This is done by setting a location listener to the location provider object. The location object is then returned by the locationUpdated() method as follows:

```
import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.StringItem;
import javax.microedition.location.Criteria;
import javax.microedition.location.Location;
import javax.microedition.location.LocationListener;
import javax.microedition.location.LocationProvider;
import javax.microedition.location.QualifiedCoordinates;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;

public class LocationWithListener extends MIDlet implements
Runnable,LocationListener,CommandListener {
 Form mainform;
 Thread t;
 LocationProvider lp;
 Display display;
 StringItem latitude;
 StringItem longitude;
 Command exitCommand=new Command("Exit",Command.OK,0);
 Criteria criteria;

 protected void destroyApp(boolean arg0) {}
 protected void pauseApp() {}

 protected void startApp() throws MIDletStateChangeException
 {
  mainform=new Form("Location Listener");
  latitude=new StringItem("Latitude:","N/A");
  longitude=new StringItem("Longitude:","N/A");
  Display.getDisplay(this).setCurrent(mainform);
  mainform.append(latitude);
  mainform.append(longitude);
  mainform.addCommand(exitCommand);
  mainform.setCommandListener(this);
  t=new Thread(this);
```


(C) Copyright Nokia 2014. All rights reserved.

```
  t.start();
 }
    public void run()
    {
     try{
      criteria=new Criteria();
      criteria.setPreferredPowerConsumption(Criteria.POWER_USAGE_HIGH);
      lp = LocationProvider.getInstance(criteria);
      lp.setLocationListener(this, -1, -1, -1);
     }catch(Exception e)
     {
       Alert alert = new Alert("Error", "Could not retrieve location!", null,
AlertType.ERROR);
       display.setCurrent(alert);
     }
    }
 public void locationUpdated(LocationProvider arg0, Location loc)
 {
  if(loc.isValid())
  {
   QualifiedCoordinates c=loc.getQualifiedCoordinates();
   latitude.setText(String.valueOf(c.getLatitude()));
   longitude.setText(String.valueOf(c.getLongitude()));
  }
 }
 public void providerStateChanged(LocationProvider provider, int newState) { }
 public void commandAction(Command cmd, Displayable arg1)
 {
  if(cmd==exitCommand)
  {
   notifyDestroyed();
  }
 }

}
```

Making repeated calls to the getLocation() method

Another way that some developers use to listen for location updates, is to call the getLocation() method within a loop. In the code example below, the current location is retrieved every second, by forcing the running thread to sleep, for the same amount of time:

```
import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.AlertType;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.StringItem;
import javax.microedition.location.Location;
import javax.microedition.location.LocationProvider;
import javax.microedition.location.QualifiedCoordinates;
import javax.microedition.midlet.MIDlet;
import javax.microedition.midlet.MIDletStateChangeException;

public class LocationWithPolling extends MIDlet implements Runnable, CommandListener {
```


(C) Copyright Nokia 2014. All rights reserved.

```
 Form mainform;
 Thread t;
 LocationProvider lp;
 Display display;
 StringItem latitude;
 StringItem longitude;
 Command exitCommand=new Command("Exit",Command.OK,0);

 protected void destroyApp(boolean arg0) {}
 protected void pauseApp() {}

 protected void startApp() throws MIDletStateChangeException
 {
  mainform=new Form("Location Polling");
  latitude=new StringItem("Latitude:","N/A");
  longitude=new StringItem("Longitude:","N/A");
  Display.getDisplay(this).setCurrent(mainform);
  mainform.append(latitude);
  mainform.append(longitude);
  mainform.addCommand(exitCommand);
  mainform.setCommandListener(this);
  t=new Thread(this);
  t.start();
 }
    public void run()
    {
     try{
      lp = LocationProvider.getInstance(null);
      while(true)
      {
       Location loc=lp.getLocation(5000);
       QualifiedCoordinates c=loc.getQualifiedCoordinates();
       latitude.setText(String.valueOf(c.getLatitude()));
       longitude.setText(String.valueOf(c.getLongitude()));
       Thread.sleep(1000);
      }
     }catch(Exception e)
     {
       Alert alert = new Alert("Error", "Could not retrieve location!", null,
AlertType.ERROR);
       display.setCurrent(alert);
     }
    }
 public void providerStateChanged(LocationProvider provider, int newState) { }
 public void commandAction(Command cmd, Displayable arg1)
 {
  if(cmd==exitCommand)
  {
   notifyDestroyed();
  }
 }

}
```

Cell-ID based versus GPS based location update

(C) Copyright Nokia 2014. All rights reserved.

Please note that when the current location is retrieved by using the Cell-ID option, it is not possible to assign a location listener to the Cell-ID based location provider. Therefore it is not possible in the network driven location finding, to use the first option described in this article and retrieve the location object from within the locationUpdated method. Even though, the use cases for frequent Cell-ID based location retrievals are rare, due to the low accuracy provided by this method, it is possible to call the getLocation method within a loop and provide location updates as described above. This article focuses on GPS-based location updates.

Setting a Location Listener versus making repeated calls to the getLocation() method

The major difference between these two methods, is that the values of the interval update and time out parameters are set in a different way. In the first example both parameters are passed as arguments to the setLocationListener() method, while in the second example, the interval value is indirectly set by forcing the running thread to sleep for the given time, and the time out parameter is passed as argument when calling the getLocation() method.

When frequent location queries are needed, setting a Location Listener should be preferred compared to repeatedly calling the getLocation() method for the following reasons:

a) The call to getLocation() generates screen prompts because it requires the user's permission to query the location information. The same applies for the setLocationListener() method. The difference is, that the former is called continuously within the loop, thus making it impractical to use, unless the application is signed, while the latter is called only once. Note that whether the getLocation() method requires signing, in order not to prompt for the location retrieval, depends on the permission level needed per device. Using the setLocationListener() method, ensures that there are minimal prompts without the need to sign the application among the largest set of devices possible.

b) By using the setLocationListener() method, it is possible for the implementation to reuse location results if it already has a recent location result when the update is due to be delivered. This is done by passing an additional argument, the maxAge parameter that defines how old the location result is allowed to be, when the update is made.

However when the application needs to retrieve the device's location only once, the getLocation() method should be preferred. This is due to simplicity in the implementation. If the setLocationListener() method was preferred in this case, some additional logic would be needed so that the location listener was replaced by another one that does not listen for location updates, as soon as a valid location has been retrieved once. Otherwise, even if the application ignored the location updates provided by the locationUpdated() method, those would still run in the background, causing increased usage of battery power.

How to save battery power when the application no longer requires to be location aware

One important thing, that requires attention when using the setLocationListener() method, is to stop the listener, if at some point the application no longer requires to be location aware. This is done with the following code:

```
provider.setLocationListener(this, 0, 0, 0);
```

This is in order for the location listener to be replaced with a new object that does not listen for location updates. Only one location listener can be registered with the given location provider. Registering again a location listener to the given provider, means that the new one replaces the old one. Setting the interval of the listener to 0, indicates that the application wants to receive only provider status updates and not location updates. In this case, the values of time out and maxAge, don’t matter.

The power consumption when the application stops listening for location updates has been greatly improved on Symbian platforms after S60 3rd Edition Feature Pack 1. Power consumption tests on Symbian Anna, show a great reduction in battery usage, when the GPS module is stopped from within the application, after the above changes in the code have been made.

Resources

The source file as well as the binaries for the first snippet that uses a Location Listener can be downloaded from here: File:LocationWithListener.zip

The source file as well as the binaries for the second snippet that uses the getLocation method in a repeating thread can be

 (C) Copyright Nokia 2014. All rights reserved.

See also

- Java Developer's Library : Specifying the location provider

(C) Copyright Nokia 2014. All rights reserved.