Location API

Version 0.9, Draft

JSR 179

COPYRIGHT

Trademarks and Service Marks

About This Document

This document describes the JSR 179 Location API followed by sample code.

Scope

This document is intended for MIDP developers who want to develop mobile Java applications. This document introduces you to the Location API for Java ME(JSR 179) API that can be used for developing location-based services. It assumes good knowledge of java programming language.

To know more about Java ME basics and Java programming language, refer to the Knowledge Base under Samsung Mobile Innovator (SMI).

http://innovator.samsungmobile.com/platform.main.do?platformId=3

Document History:

Date	Version	Comment
22/06/09	0.9	Draft

References:

1. Location API JSR : http://jcp.org/en/jsr/detail?id=179
2. Location API Article : http://developers.sun.com/mobility/apis/articles/location/

Abbreviations:

JSR	Java Specification Request
API	Application Programming Interface
CLDC	Connected Limited Device Configuration
BTS	Base Transceiver Station
GPS	Global Positioning System

Table of Contents

List of Tables

Introduction

Location API is an Optional Package *javax.microedition.location* that provides access to location based information. Location API provides a standard for developers to write mobile location-based applications. Location API gives information about the present physical location of the device.

It can be used with many Java ME profiles. The minimum platform is CLDC and the targets are low memory devices. This API implementation footprint is:

- ROM budget max. 20 KB
- RAM budget max. 2 KB

Expressing Location:

Locations can be expressed in spatial terms or text descriptions.

A spatial location is expressed in the form of latitude-longitude-altitude coordinate system. Latitude is expressed as 0-90 degrees north or south of the equator and longitude as 0-180 degrees east or west of the prime meridian, which passes through Greenwich, England. Altitude is expressed in meters above sea level. A text description is usually expressed as a street address, including city, postal code, and so on.

Device Location:

Applications can use the following methods to determine the device location:

Cell ID (Using the mobile phone network)

Cell ID method can be used to determine device location by identifying the Base Transceiver Station (BTS), that the device is communicating with and the location of that BTS. The accuracy is less since the accuracy of this method depends on the size of the cell.

GPS (Using satellites)

The Global Positioning System (GPS) is potentially the most accurate method but it has some drawbacks. The extra hardware can be costly, consumes battery while in use, and requires some warm-up after a cold start to get an initial fix on visible satellites. It also suffers from "canyon effects" in cities, where satellite visibility is intermittent.

Bluetooth (Using short-range positioning)

In relatively small areas, such as a single building, a local area network can provide locations along with other services. For example, appropriately equipped devices can use Bluetooth for short-range positioning.

Overview

The features of the Location API are:

- Location -To obtain information about the device location.
- Landmarks - To create, edit, store, and retrieve landmarks.
- Orientation - To obtain the orientation of a device.

Location API package *javax.microedition.location* contains the basic classes and interface to access location information.

API Description:

Package contains 9 classes, 2 interfaces and 2 exceptions. The following table shows the classes, interface and exceptions.

Table 1: Class Information

Class	Description
AddressInfo	AddressInfo class holds textual address information about a location.
Coordinates	Coordinates class represents coordinates as latitude-longitude-altitude values.
Criteria	The criterion used for the selection of the location provider is defined by the values in this class.
Landmark	The Landmark class represents a landmark, i.e. a known location with a name.
LandmarkStore	The LandmarkStore class provides methods to store, delete and retrieve landmarks from a persistent landmark store
Location	The Location class represents the standard set of basic location information.
LocationProvider	This is the starting point for applications using this API and represents a source of the location information.
Orientation	The Orientation class represents the physical orientation of the terminal
QualifiedCoordinates	The QualifiedCoordinates class represents coordinates as latitude-longitude-altitude values that are associated with an accurate value

Table 2: Interface Information

Interface	Description
LocationListener	The LocationListener represents a listener that receives events associated with a particular LocationProvider.
ProximityListener	This interface represents a listener to events associated with detecting proximity to some registered coordinates

Table 3: Exception Information

Exception	Description
LandmarkException	The LandmarkException is thrown when an error related to handling landmarks has occurred.
LocationException	The LocationException is thrown when a location API specific error has occurred.

Location

Location feature provides location and related information about the device. Location and LocationProvider are the classes, which provide methods to obtain Location information.

The Location class abstracts the location information. Information includes timestamp, coordinates, accuracy, speed, course, information and optional textual address information.

Obtaining a Location

Following are the steps to obtain device Location:

1. Specify Criteria
2. Set Criteria to LocationProvider
3. Get LocationProvider instance as per the Criteria specified
4. Get Location object from LocationProvider
5. Get Coordinates from Location

To obtain Location information, application needs to obtain LocationProvider instance by specifying the criteria. LocationProvider is the starting point of the application to get Location Information. It represents a Location module generating Locations.

The application can specify criteria for the selection of LocationProvider. It is up to the application to determine and specify criteria for selecting the location method.

Criteria is used by LocationProvider factory method

static LocationProvider getInstance(Criteria criteria)

to get LocationProvider instance that best fits the given criteria.

Using LocationProvider, application can now get Location object by:

1) Using *getLocation(int timeout)* method to get single Location Object.

2) Registering listener to get Location at periodic intervals.

Location object contains location information and coordinates. Location information includes accuracy, speed, course, and information about the positioning method used for the location, plus an optional textual address.

Coordinates are represented by either of two classes:

- Coordinates object represents a point's latitude and longitude in degrees, and altitude in meters.
- QualifiedCoordinates object contains latitude, longitude, altitude and an indication of their accuracy, represented as the radius of an area.

Textual address of the location is represented by AddressInfo object. AddressInfo contains the textual information about the location. It has getter and setter methods for setting and retrieving data based on the constant fields. Example: Country, Country Code, Phone number, State, Street etc.

1) Using *getLocation()* method

The following code snippet shows how to obtain location object using *getLocation()* method:

```
…
/* Set criteria for selecting a location provider:
 accurate to 500 meters horizontally*/
Criteria cr= new Criteria();
cr.setHorizontalAccuracy(500);

/* Get an instance of the provider*/
LocationProvider lp= LocationProvider.getInstance(cr);

/* Request the location, setting a one-minute timeout*/
Location l = lp.getLocation(60);
Coordinates c = l.getQualifiedCoordinates();

if(c != null ) {
```

```
 /* Use coordinate information*/
 double lat = c.getLatitude();
 double lon = c.getLongitude();
}
…
```

2) Using Listener

LocationProvider class has two listener methods to register a listener to get periodically updated Location objects via

1. **LocationListener**

```
setLocationListener(LocationListener listener, int interval, int timeout, int maxAge)
```

2. **ProximityListener**

```
addProximityListener(ProximityListener listener, Coordinates coordinates, float
proximityRadius)
```

LocationListener

LocationListener gives regular position updates at defined interval.

LocationListener has two methods

```
locationUpdated(LocationProvider provider,Location location)
```

locationupdated() method gives location updates at regular intervals.

```
providerStateChanged(LocationProvider provider,int newState)
```

providerStateChanged() gives information of LocationProvider state i.e. AVAILABLE, OUT_OF_SERVICE or TEMPORARILY_UNAVAILABLE

Following code snippet shows the implementation of the above methods.

```
…
public void locationUpdated(LocationProvider provider, Location location){
   if (location != null && location.isValid()) {
     QualifiedCoordinates qc = location.getQualifiedCoordinates();
     form.append("Lat: "+qc.getLatitude()+"Lon: "+qc.getLongitude()+     "Alt: " +
qc.getAltitude()”     );
   }
 }

public void providerStateChanged(LocationProvider provider,int newState){
form.append(“newState ”+newState);
}
…
```

ProximityListener

Application can also add ProximityListener that notifies when proximity to registered coordinates is detected. The listener is called when the terminal enters the proximity of registered coordinates.

ProximityListener has two methods:

```
proximityEvent(Coordinates coordinates,Location location)
```

proximityEvent is called when user comes in the proximity of a particular location.

```
monitoringStateChanged(boolean isMonitoringActive)
```

monitoringStateChanged is called when the state of the proximity monitoring is changed. isMonitoringActive boolean indicates the new state of the proximity monitoring.

true - indicates that the proximity monitoring is active.

false - indicates that the proximity monitoring cannot be done currently.

Following code snippet shows the implementation of the above methods:

```
…
try {
        Criteria cr = new Criteria();
        provider = LocationProvider.getInstance(cr);
        /* coordinates around which proximity has to be found*/
        Coordinates proximityCoordinates = new Coordinates(34.359766723, 60.495850625, 410);
        /*Listener registration for above coordinates for proximity of 2000 meters.*/
        LocationProvider.addProximityListener(this, proximityCoordinates, 2000.0f);

        } catch (LocationException e) {

                e.printStackTrace();
        }

/*ProxmityListener methods*/

public void proximityEvent(Coordinates arg0, Location arg1) {
form.append("You are presently within 2000 meters radius from Location Lat: 34.359766723
Long: 60.495850625" );
}

public void monitoringStateChanged(boolean isActive)
{
  if(isActive)
```

```
        form.append("Monitoring is Active");
 else
        form.append("Monitoring is currently not Active");

}
…
```

Landmark

Landmark is a physical location with a name that represents as a location to the end user. Location API allows user to create new landmark, add, store, retrieve, and delete landmarks. Landmark and LandmarkStore class provides this functionality.

Landmark class represents Landmark information. Landmark information includes name, description, address information and qualified coordinates. Address information is represented by AddressInfo object.

```
Landmark(String name, String description, QualifiedCoordinates coordinates, AddressInfo
addressInfo)
```

Following code snippet shows how to create Landmark

```
...
AddressInfo textAddress = new AddressInfo();
textAddress.setField(AddressInfo.COUNTRY , "UK");
textAddress.setField(AddressInfo.CITY , "London");

Landmark landmark = new Landmark("My Restaurant", " My Restaurant best in  the world",
new QualifiedCoordinates(11.289496608768690, 34.59678880927362,460, 31.32, 45.000) ,
textAddress);
...
```

LandmarkStore is a shared persistent area to store, modify, and delete landmarks. Landmark information can be stored in this data store and can be used later by the application as and when required.

Landmark can also be categorized in LandmarkStore. The Landmark has a name and may be placed in a category or several categories. The category is intended to group landmarks that are of similar type to the end user, e.g. Restaurants, Museums, etc.

All Landmark store database must be shared between all Java ME Applications and may be shared with native applications.

Following code snippet shows creating LandmarkStore and LandmarkStore Category:

```
…
/*Check if the Landmark Store already exists*/
String allLandmarkStores[] = LandmarkStore.listLandmarkStores();
boolean isStoreExist=false;
for(int i=0; i< allLandmarkStores.length; i++) {
        if(allLandmarkStores [i].equals("MyLandmarks")) {
        isStoreExist = true;
        }
}

/*Create a new Lanmark Store if the store does not exist*/
if(!isStoreExist){
        LandmarkStore.createLandmarkStore("MyLandmarkStore");
}
myLmStore = LandmarkStore.getInstance("MyLandmarkStore");
/*Add category*/
myLmStore.addCategory("Restaurant");
...
```

Adding Landmark into LandmarkStore

LandmarkStore class provides *addLandmark(Landmark landmark, String category)* to add Landmark object into LandmarkStore. Category could be Restaurants, Home etc.

Following code snippet shows how to add Landmark into LandmarkStore:

```
…
Criteria cr = new Criteria();
/*Accuracy set to 100 Meters*/
cr.setHorizontalAccuracy(100);
/*Get LocationProvider*/
LocationProvider provider = LocationProvider.getInstance(cr);

/*180 secs time out*/
Location location = provider.getLocation(180);
QualifiedCoordinates c = location.getQualifiedCoordinates();
double latitude = c.getLatitude();
double longitude = c.getLongitude();
float altitude = c.getAltitude();
float hAccuracy = c.getHorizontalAccuracy();
float vAccuracy = c.getVerticalAccuracy();

/* Same Address Info for all landmarks. */
AddressInfo textAddress = new AddressInfo();
textAddress.setField(AddressInfo.COUNTRY , "MyCountry");
```

```
textAddress.setField(AddressInfo.STATE , "My State");
textAddress.setField(AddressInfo.CITY , "MyCity");
textAddress.setField(AddressInfo.STREET , "MyStreet");

Landmark landmark = new Landmark(name,description,new QualifiedCoordinates(latitude,
longitude, altitude, hAccuracy, vAccuracy) ,textAddress);
…
```

Retrieving Landmark from LandmarkStore

Similarly, Landmark objects can be retrieved using *getLandmarks()*. The Landmark objects returned from the *getLandmarks()* guarantee that the application can read a consistent set of the landmark data valid at the time of obtaining the object instance, even if the landmark information in the store is modified subsequently by this or some other application. *getLandmarks()* returns an enumeration of all Landmarks.

```
…
try {
  Enumeration landmarkEnum = MyLandmarkStore.getLandmarks();
  while (landmarkEnum.hasMoreElements()) {
    Landmark myLandmark = (Landmark)landmarkEnum.nextElement();
    double lmLatitude = myLandmark.getQualifiedCoordinates().getLatitude();
    double lmLongitude =
        myLandmark.getQualifiedCoordinates().getLatitude();
    String landmarkName = myLandmark.getName();
    String lmCountry=
        landmark.getAddressInfo().getField(AddressInfo.COUNTRY);

    form.append("landmarkName  "+landmarkName + "Latitude: " + lmLatitude + " Longitude: " +
lmLongitude + " Country: " +lmCountry);
}

}catch(Exception e){
          form.append("No Landmarks in MyLandmarkStore");
}
…
```

Deleting Landmark from LandmarkStore

Landmark objects can be deleted from LandmarkStore database by using *deleteLandmarkStore(String storeName)*.

All landmarks and categories defined in this store are removed. Similarly, *deleteLandmark (Landmark lm)* deletes landmark from current LandmarkStore.

This method removes the landmark from all categories and deletes the information from this LandmarkStore.

```
…
try{
/*Check if the Landmark Store already exists*/
String allLandmarkStores[] = LandmarkStore.listLandmarkStores();
boolean isStoreExist=false;
for(int i=0; i< allLandmarkStores.length; i++) {
            if(allLandmarkStores [i].equals("MyLandmarks")) {
            isStoreExist = true;
            }
}
if(isStoreExist)
LandmarkStore .deleteLandmarkStore(“MyLandmarks”);
}catch(Exception e)
{
form.append(“Exception at deleting Landmark”+e);
}
…
```

Orientation

This feature is useful for navigational purpose. Orientation class represents the physical orientation of the device (compass orientation). Application may be able to determine not only device location but also its orientation, if the device has a compass.

Orientation is described by azimuth to north (the horizontal pointing direction), pitch (the vertical elevation angle) and roll (the rotation of the terminal around its own longitudinal axis).

To get Orientation object use getOrientation() method. Following code snippet shows orientation

```
…
Orientation orientation = Orientation.getOrientation();
float azimuth = orientation.getCompassAzimuth();
float pitch = orientation.getPitch();
float roll = orientation.getRoll();
boolean isOrientationMagnetic = orientation.isOrientationMagnetic();
form.append(“azimuth ”+azimuth+” pitch ”+pitch+” roll ”+roll+” isOrientationMagnetic ”+
isOrientationMagnetic);
…
```

isOrientationMagnetic() returns a boolean value that indicates whether this Orientation is relative to the magnetic field of the Earth or relative to true north and gravity.

Detecting Location API presence

To check whether the handset support Location API,

```
System.getProperty("microedition.location.version").
```

can be used. If it is supported, the location version is returned else null will be returned.

Security & Permissions

Some methods in this API are defined to throw a SecurityException if the user does not have the permissions needed to perform the action.

Following table shows the permissions associated with Location API.

Table 4: Permissions

Permissions name	Methods protected by this Permission
javax.microedition.location.Location	LocationProvider.getLocation() LocationProvider.setLocationListener()
javax.microedition.location.Orientation	Orientation.getOrientation()
javax.microedition.location.ProximityListener	LocationProvider.addProximityListener()
javax.microedition.location.LandmarkStore.read	LandmarkStore.getInstance() LandmarkStore.listLandmarkStores()
javax.microedition.location.LandmarkStore.write	LandmarkStore.addLandmark() LandmarkStore.deleteLandmark() LandmarkStore.removeLandmarkFromCategory() LandmarkStore.updateLandmark()
javax.microedition.location.LandmarkStore.category	LandmarkStore.addCategory(), LandmarkStore.deleteCategory()
javax.microedition.location.LandmarkStore.management	LandmarkStore.createLandmarkStore(), LandmarkStore.deleteLandmarkStore()

Location Example

This sample example shows how to use Location API to get location information.

Class: LocationMidlet.java

```
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.location.Criteria;
import javax.microedition.location.Location;
import javax.microedition.location.LocationException;
import javax.microedition.location.LocationProvider;

import javax.microedition.location.QualifiedCoordinates;
import javax.microedition.midlet.MIDlet;

public class LocationMidlet extends MIDlet implements CommandListener,Runnable{

   /* Define String Constants */
   private final String locate_String = "Locate";
   private final String exit_String = "Exit";
   private final String done_String = "Done";
   private final String locationForm_Title_String = "Location Demo";
   private final String locationInfo_String =
                        "Press Locate Command to start Application Demo \n";
   private final String exitInfo_String = "Press Exit Command to exit Application \n";
   /*LocationMidlet Display */
   private Display display;

   /*LocationMidlet Commands*/
   private Command cmd_Locate, cmd_Exit, cmd_Done;

   /*LocationMidlet Form*/
   private Form locationForm;

   /** location provider */
   private LocationProvider locationProvider = null;
   private Location location;
   private QualifiedCoordinates coordinates;

   private Thread locationThread;
```

```
/*LocationMidlet Constructor*/
public LocationMidlet()
{
   /*Intialize Commands*/
   cmd_Locate = new Command(locate_String, Command.OK,1);
   cmd_Done   = new Command(done_String, Command.OK,1);
   cmd_Exit   = new Command(exit_String, Command.EXIT,2);

   /*Create Form*/
   locationForm = new Form(locationForm_Title_String);

   /*Appending location and Exit Information*/
   locationForm.append(locationInfo_String);
   locationForm.append(exitInfo_String);

   /*Adding commands to Location Form*/
   locationForm.addCommand(cmd_Exit);
   locationForm.addCommand(cmd_Locate);
   locationForm.setCommandListener(this);
   /*Intialize display*/
   display = Display.getDisplay(this);

   /*Create LocationProvider Instance*/
   locationProvider = null;
   createLocationProvider();
   if(locationProvider == null)
   {
      displayError();
   }
}

public void startApp() {
   /*display Form*/
   display.setCurrent(locationForm);
}

public void pauseApp() {

   /*your code to handle interrupt*/
}

public void destroyApp(boolean unconditional) {
   /*your code before notifyDestroyed method*/
   notifyDestroyed();
}

private void exitMidlet()
```

```
{
    destroyApp(true);
}

private void displayError()
{
    locationForm.deleteAll();
    locationForm.append("Unable to create LocationProvider");
    locationForm.addCommand(cmd_Exit);
}

/*Initialize LocationProvider using Default Criteria*/
private void createLocationProvider() {
    if (locationProvider == null) {
        /*Constructs a Criteria object. All the fields are set to the default values*/
        Criteria criteria = new Criteria();
        /* criteria Fields which you can set
        criteria.setCostAllowed(true); //default value
        criteria.setSpeedAndCourseRequired(true);
        criteria.setHorizontalAccuracy(500);
        criteria.setAltitudeRequired(true);
        criteria.setPreferredPowerConsumption(Criteria.POWER_USAGE_LOW);
        */

        try {
            locationProvider = LocationProvider.getInstance(criteria);
        } catch (LocationException le) {
            /*Unable to create Location Provider using this criteria*/
            le.printStackTrace();
        }
    }
}

private void createLocation()
{
    try {
        // Get a location fix for 30 seconds timeout
        location = locationProvider.getLocation(30);
    } catch (LocationException ex) {
        ex.printStackTrace();
    } catch (InterruptedException ex) {
        ex.printStackTrace();
    }
}

private void getLocationResults()
{
```

```
    coordinates = location.getQualifiedCoordinates();
}

private void displayLocationResults()
{
   /*
    locationForm.append("Course"+loc.getCourse());
    locationForm.append("Speed"+loc.getSpeed());
    */
   locationForm.append("Altitude:"+coordinates.getAltitude() +"\n");
   locationForm.append("Latitude:"+coordinates.getLatitude() +"\n");
   locationForm.append("Longitude:"+coordinates.getLongitude() +"\n");
}

public void commandAction(Command cmd, Displayable disp) {
   if(cmd == cmd_Locate)
   {
      /*Deleting previous data displayed*/
      locationForm.deleteAll();
      /*Removing exit and locate command*/
      locationForm.removeCommand(cmd_Exit);
      locationForm.removeCommand(cmd_Locate);
      /*Adding done command*/
      locationForm.addCommand(cmd_Done);
      /*Create Thread to avoid deadlock*/
      locationThread = new Thread(this);
      locationThread.start();
   }else
   if(cmd == cmd_Exit)
   {
      exitMidlet();
   }else
   if(cmd == cmd_Done)
   {
      locationForm.deleteAll();

      /*Remove done command*/
      locationForm.removeCommand(cmd_Done);

      /*Adding commands to Location Form*/
      locationForm.addCommand(cmd_Exit);
      locationForm.addCommand(cmd_Locate);

      /*Appending location and Exit Information*/
      locationForm.append(locationInfo_String);
      locationForm.append(exitInfo_String);
```

```
        }
    }

    public void run() {
        /*Get Location*/
        createLocation();
        /*Get Location Results*/
        getLocationResults();
        /*Display Results*/
        displayLocationResults();
    }
}
```

Landmark Example

This example shows how to add, store, retrieve, delete landmark.

Class: LandmarkMidlet.java

```
import java.io.IOException;

import java.util.Enumeration;
import java.util.Vector;

import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.List;

import javax.microedition.location.AddressInfo;
import javax.microedition.location.Coordinates;
import javax.microedition.location.Landmark;
import javax.microedition.location.LandmarkException;
import javax.microedition.location.LandmarkStore;
import javax.microedition.location.QualifiedCoordinates;

import javax.microedition.midlet.MIDlet;

public class LandmarkMidlet extends MIDlet implements CommandListener, Runnable{
    private final String OFFICE_CATEGORY = "OFFICE";
    private LandmarkStore landmarkStore = null;

    private final int ADD_LANDMARK = 1;
    private final int SHOW_LANDMARK = ADD_LANDMARK + 1;
    private final int DELETE_LANDMARK = SHOW_LANDMARK + 1;
```

```
private int landmarkState;

private List landmarkList;
private Command cmd_Exit;
private Command cmd_showLandmark, cmd_deleteLandmark;
private Command cmd_addLandmark;
private Display display;

private Thread landmarkThread;
private Vector landmarkVector;

public LandmarkMidlet()
{
   landmarkList = new List("Landmark Demo", List.IMPLICIT);

   cmd_Exit = new Command("Exit", Command.EXIT, 1);
   /* cmd_addCategory = new Command("Add Category", Command.SCREEN, 1); */
   cmd_addLandmark = new Command("Add Landmark", Command.SCREEN, 1);
   /* cmd_showCategory = new Command("show Category", Command.SCREEN, 1); */
   cmd_showLandmark = new Command("show Landmark", Command.SCREEN, 1);
   cmd_deleteLandmark = new Command("delete Landmark", Command.SCREEN, 1);
   /* cmd_deleteCategory = new Command("delete Category", Command.SCREEN, 1);*/

   landmarkList.addCommand(cmd_Exit);
   landmarkList.addCommand(cmd_addLandmark);
   landmarkList.addCommand(cmd_showLandmark);
   landmarkList.addCommand(cmd_deleteLandmark);

   landmarkList.setCommandListener(this);

   landmarkVector = new Vector();

   display = Display.getDisplay(this);

   display.setCurrent(landmarkList);
}

public void startApp() {
   loadLandmarks();
}

public void pauseApp() {
   /*your code to handle interrupts*/
}

public void destroyApp(boolean unconditional) {
   /*your code before notifyDestroyed*/
```

```
    notifyDestroyed();
  }

  private void exitMidlet()
  {
    destroyApp(true);
  }

  private void loadLandmarks() {
    landmarkStore = LandmarkStore.getInstance(null);
    /*Added Office Category in LandmarkStore*/
    addLandmarkCategory();
  }

  private void addLandmarkCategory()
  {
    try {
      Enumeration c = landmarkStore.getCategories();
      boolean exists;
      for (exists = false; c.hasMoreElements() ;)
      {
        String category = (String)c.nextElement();
        if (category.equals(OFFICE_CATEGORY)) {
          exists = true;
          break;
        }
      }
      /*If Office Category doesn't exists then add Office Category
       in LandmarkStore*/
      if(!exists)
        landmarkStore.addCategory(OFFICE_CATEGORY);
    } catch (IOException ex) {
      ex.printStackTrace();
    } catch (LandmarkException ex) {
      ex.printStackTrace();
    }
  }

  private void addLandmark()
  {
    /*Latitude range -90 to 90 degrees*/
    /*Longitude range -180 to 180 degrees*/
    QualifiedCoordinates officeCoordinates = new QualifiedCoordinates(85.0, 85.0, 100, 500,
500);

    AddressInfo officeAddressInfo = new AddressInfo();
    officeAddressInfo.setField(AddressInfo.STREET, "OfficeStreet");
```

```
        officeAddressInfo.setField(AddressInfo.CITY, "OfficeCity");
        officeAddressInfo.setField(AddressInfo.POSTAL_CODE, "123456");
        officeAddressInfo.setField(AddressInfo.PHONE_NUMBER, "1234567890");

        Landmark landmark1 = new Landmark("My Office", "Office Description",
(QualifiedCoordinates) officeCoordinates, officeAddressInfo);

        try {
            /*Category needs to be present before adding Landmark
             * addLandmarkCategory() adds category
             */
            landmarkStore.addLandmark(landmark1, OFFICE_CATEGORY);
            /*Refresh List with newly added Landmark*/
            showLandmark();
        } catch (IOException ex) {
            /*Landmark could not be added to Office category*/
            ex.printStackTrace();
        }
    }

    private void showLandmark()
    {
        try {
            /*delete landmarks in list and vector*/
            landmarkVector.removeAllElements();
            landmarkList.deleteAll();
            Enumeration c = landmarkStore.getLandmarks();
            boolean exists;
            /*returns null if no landmark is present in landmarkStore Category*/
            for (exists = false; c!=null && c.hasMoreElements();) {
                Landmark landmark = (Landmark) c.nextElement ();
                landmarkVector.addElement(landmark);
                landmarkList.append(landmark.getName(), null);
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }

    private void deleteLandmark()
    {
        int landmarkIndex = landmarkList.getSelectedIndex();
        /*Check to avoid OutofBoundsException*/
        if(landmarkIndex>-1)
        {
            Landmark landmark = (Landmark)landmarkVector.elementAt(landmarkIndex);
            try {
```

```
            landmarkStore.deleteLandmark(landmark);
        } catch (IOException ex) {
            ex.printStackTrace();
        } catch (LandmarkException ex) {
            ex.printStackTrace();
        }
        /*Refresh List*/
        showLandmark();
    }
  }

  public void commandAction(Command cmd, Displayable disp) {
    if(cmd_Exit == cmd)
    {
      exitMidlet();
    }else
    if(cmd_addLandmark == cmd)
    {
      landmarkState  = ADD_LANDMARK;
      landmarkThread = new Thread(this);
      landmarkThread.start();
    }else
    if(cmd_showLandmark == cmd)
    {
      showLandmark();
    }else
    if(cmd_deleteLandmark == cmd)
    {
      landmarkState  = DELETE_LANDMARK;
      landmarkThread = new Thread(this);
      landmarkThread.start();
    }
  }

  public void run() {
    if(ADD_LANDMARK == landmarkState)
      addLandmark();
    else
    if(DELETE_LANDMARK == landmarkState)
      deleteLandmark();

    landmarkState = -1;
  }
}
```

Orientation Example

This example shows how to get Orientation.

Class: OrientationMidlet.java

```
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.location.Orientation;

public class OrientationMidlet extends MIDlet implements CommandListener {

   private Command exitCmd  = new Command("Exit", Command.EXIT, 1);
   private Display display;
   private Form mainForm;

   public OrientationMidlet(){
   }

   protected void destroyApp(boolean unconditional){

   }

   protected void pauseApp(){
   }

   protected void startApp()throws MIDletStateChangeException{
      display = Display.getDisplay(this);
      mainForm = new Form("OrientationMidlet");
      mainForm.addCommand(exitCmd);
      mainForm.setCommandListener(this);
      getOrientation();
      display.setCurrent(mainForm);
   }

   private void getOrientation()
   {
         try{
         Orientation orientation = Orientation.getOrientation();
                  float azimuth = orientation.getCompassAzimuth();
                  float pitch = orientation.getPitch();
                  float roll = orientation.getRoll();
                  boolean isOrientationMagnetic = orientation.isOrientationMagnetic();

         mainForm.append("azimuth "+azimuth);
         mainForm.append("pitch "+pitch);
         mainForm.append("roll "+roll);
```

```
            mainForm.append("isOrientationMagnetic "+isOrientationMagnetic);
            }catch(Exception e)
            {
                        mainForm.append("Exception "+e);
            }
    }

    public void commandAction(Command c, Displayable d) {
        if (c == exitCmd) {
            destroyApp(false);
            notifyDestroyed();
        }
    }

}
```