MCA253 - Mobile Applications

Unit:4
Integration with Other Applications

Dr. Siddesha S MCA, M.Sc Tech (by Research) , Ph.D.
Asst. Professor,
Dept. of Computer Applications,
JSS Science and Technology University
Mysuru – 570 006

SMS Messaging

- SMS messaging is one of the **main functions** on a mobile phone today—for some users, it's as necessary as the device itself.
- Android comes with a built-in SMS application that enables you to **send and receive SMS messages.**
- However, in some cases, you might want to integrate SMS capabilities into your Android application.
- For example, you might want to write an application that automatically sends an SMS message at regular time intervals.
- For example, this would be useful if you wanted to track the location of your kids—simply give them an Android device that sends out an SMS message containing its geographical location every 30 minutes.

SMS Messaging

- The good news for Android developers is that you don't need a real device to test SMS messaging: The free Android emulator provides that capability.
- In fact, when looking at your emulator window, the four-digit number that appears above your emulator is its "phonenumber."
- The first emulator session that you open is typically 5554, with each subsequent session being incremented by 1.

Sending SMS Messages Programmatically:

Program Name: SendSMS.

- Android uses a permissions-based policy whereby all the permissions needed by an application must be specified in the AndroidManifest.xml file.
- This ensures that when the application is installed, the user knows exactly which access permissions it requires.

Sending SMS Messages Programmatically

- To send an SMS message programmatically, you use the SmsManager class.
- Unlike other classes, do not directly instantiate this class.
- Instead, it call the getDefault() static method to obtain an SmsManager object.
- Then send the SMS message using the sendTextMessage() method.

//---sends an SMS message---

```
private void sendSMS(String phoneNumber, String message) {
SmsManager sms = SmsManager.getDefault();
sms.sendTextMessage(phoneNumber, null, message, null, null);
}
```

Sending SMS Messages Programmatically

- Following are the five arguments to the sendTextMessage() method;
  - **destinationAddress**—Phone number of the recipient
  - **scAddress**—Service center address; use null for default SMSC
  - **text**—Content of the SMS message
  - **sentIntent** —Pending intent to invoke when the message is sent
  - **deliveryIntent**—Pending intent to invoke when the message has been delivered

Sending SMS Messages Using Intent

- Using the SmsManager class, you can send SMS messages from within your application without the need to involve the built-in Messaging application.
- However, sometimes it would be easier if you could simply invoke the built-in Messaging application and let it handle sending the message.
- To activate the built-in Messaging application from within your application, you can use an Intent object with the MIME (Multipurpose Internet Mail Exchange) type "vnd.android-dir/mms-sms", as shown

```
Intent i = new
Intent(android.content.Intent.ACTION_VIEW);
i.putExtra("address", "5556; 5558; 5560");
i.putExtra("sms_body", "Hello my friends!");
i.setType("vnd.android-dir/mms-sms");
startActivity(i);
```

Sending SMS Messages Using Intent

- This code invokes the Messaging application directly.
- Note that you can send your SMS to multiple recipients by separating each phone number with a semicolon (in the putExtra() method).
- The numbers are separated using commas in the Messaging application.

Receiving SMS Messages

- Besides sending SMS messages from your Android applications, you can also receive incoming SMS messages from within your applications by using a BroadcastReceiver object.
- This is useful when you want your application to perform an action when a certain SMS message is received.
- For example, you might want to track the location of your phone in case it is lost or stolen.
- In this case, you can write an application that automatically listens for SMS messages containing some secret code.
- When that message is received, you can then send an SMS message containing the location's coordinates back to the sender.

Program name: **ReceiveSMS**

Receiving SMS Messages

- To listen for incoming SMS messages, you create a BroadcastReceiver class.
- The BroadcastReceiver class enables your application to receive intents sent by other applications using the sendBroadcast() method.
- Essentially, it enables your application to handle events raised by other applications.
- When an intent is received, the onReceive() method is called, which needs to be overridden.
- When an incoming SMS message is received, the onReceive() method is fired.
- The SMS message is contained in the Intent object (intent, which is the second parameter in the onReceive() method) via a Bundle object.

Receiving SMS Messages

- Note that each SMS message received invokes the onReceive() method. If your device receives five SMS messages then the onReceive() method is called five times.
- Each SMS message is stored in an Object array.
- If the SMS message is fewer than 160 characters then the array has one element.
- If an SMS message contains more than 160 characters the message is split into multiple smaller messages and is stored as multiple elements in the array.
- To extract the content of each message, use the static getMessagesFromIntent() method from Telephony.Sms .
- The phone number of the sender is obtained via the getOriginatingAddress() method.

Receiving SMS Messages

- If you need to send an autoreply to the sender, use the getOriginating-Address() method to obtain the sender's phone number.
- To extract the body of the message, use the getMessageBody() method.
- One interesting characteristic of the BroadcastReceiver is that your application continues to listen for incoming SMS messages even if it is not running.
- As long as the application is installed on the device, any incoming SMS messages are received by the application.

Preventing the Messaging Application from Receiving a Message

- When an SMS message is received, all applications (including the Messaging application) on the Android device take turns handling the incoming message.
- Sometimes, however, this is not the behavior you want.
- For example, you might want your application to receive the message and prevent it from being sent to other applications.
- This is very useful, especially if you are building some kind of tracking application.
- The solution is very simple. To prevent an incoming message from being handled by the built-in Messaging application, your application needs to handle the message before the Messaging app has the chance to do it.

Preventing the Messaging Application from Receiving a Message

- To do this, add the android:priority attribute to the <intent-filter> element, like this:

```
<receiver android:name=".SMSReceiver">
    <intent-filter android:priority="100">
        <action android:name=
        "android.provider.Telephony.SMS_RECEIVED" />
    </intent-filter>
</receiver>
```

- Set this attribute to a high number, such as 100.
- The higher the number, the earlier Android executes your application.
- When an incoming message is received, your application executes first, and you can decide what to do with the message.
- To prevent other applications from seeing the message, simply call the **abortBroadcast()** method in your **BroadcastReceiver** class

Updating an Activity from a BroadcastReceiver

- The previous section demonstrates how you can use a BroadcastReceiver class to listen for incoming SMS messages and then use the Toast class to display the received SMS message.
- Often, you'll want to send the SMS message back to the main activity of your application.
- For example, you might want to display the message in a TextView.

Program name: ActivityBroadcastReceiver

Caveats and Warnings

- Although the capability to send and receive SMS messages makes Android a very compelling platform for developing sophisticated applications, this flexibility comes with a price.
- A seemingly innocent application might send SMS messages behind the scene without the user knowing, as demonstrated by a recent case of an SMS-based Trojan Android application.

https://www.tripwire.com/state-of-security/security-data-protection/android-malware-sms/

- The app claims to be a media player, but when it's installed it sends SMS messages to a premium rate number, resulting in huge phone bills for the user.
- The user needs to explicitly give permissions (such as accessing the Internet, sending and receiving SMS messages, and so on) to your application; however, the request for permissions is shown only at installation time.

Caveats and Warnings

- If the user clicks the Install button, he or she is considered to have granted the application permission to send and receive SMS messages.
- This is dangerous because after the application is installed it can send and receive SMS messages without ever prompting the user again.
- In addition to this, the application also can “sniff” for incoming SMS messages. For example, based on the techniques you learned from the previous section, you can easily write an application that checks for certain keywords in the SMS message.
- When an SMS message contains the keyword you are looking for, you can then use the Location Manager to obtain your geographical location and then send the coordinates back to the sender of the SMS message
- The sender could then easily track your location.

Caveats and Warnings

- All these tasks can be done easily without the user knowing it!
- That said, users should try to avoid installing Android applications that come from dubious sources, such as from unknown websites or strangers.

Sending Email

- Like SMS messaging, Android also supports email.
- The Gmail/Email application on Android enables you to configure an email account using POP3 or IMAP.
- Besides sending and receiving emails using the Gmail/Email application, you can also send email messages programmatically from within your Android application.
- You must configure the Email app on your emulator.
- Simply click the Email app on the emulator and follow the on-screen prompts to set up the application.

Program name: sendEmail

Location-Based Services

- One category of apps that is very popular is Location-Based Services, commonly known as LBS.
- LBS apps track your location, and might offer additional services such as locating amenities nearby, offering suggestions for route planning, and so on.
- Of course, one of the key ingredients in an LBS app is maps, which present a visual representation of your location.

Displaying Maps

- Google Maps is one of the many applications bundled with the Android platform.
- In addition to simply using the Maps application, you can also embed it into your own applications and make it do some very cool things.
- This section describes how to use Google Maps in your Android applications and programmatically perform the following:
  - Change the views of Google Maps.
  - Obtain the latitude and longitude of locations in Google Maps.
  - Perform geocoding and reverse geocoding (translating an address to latitude and longitude and vice versa).

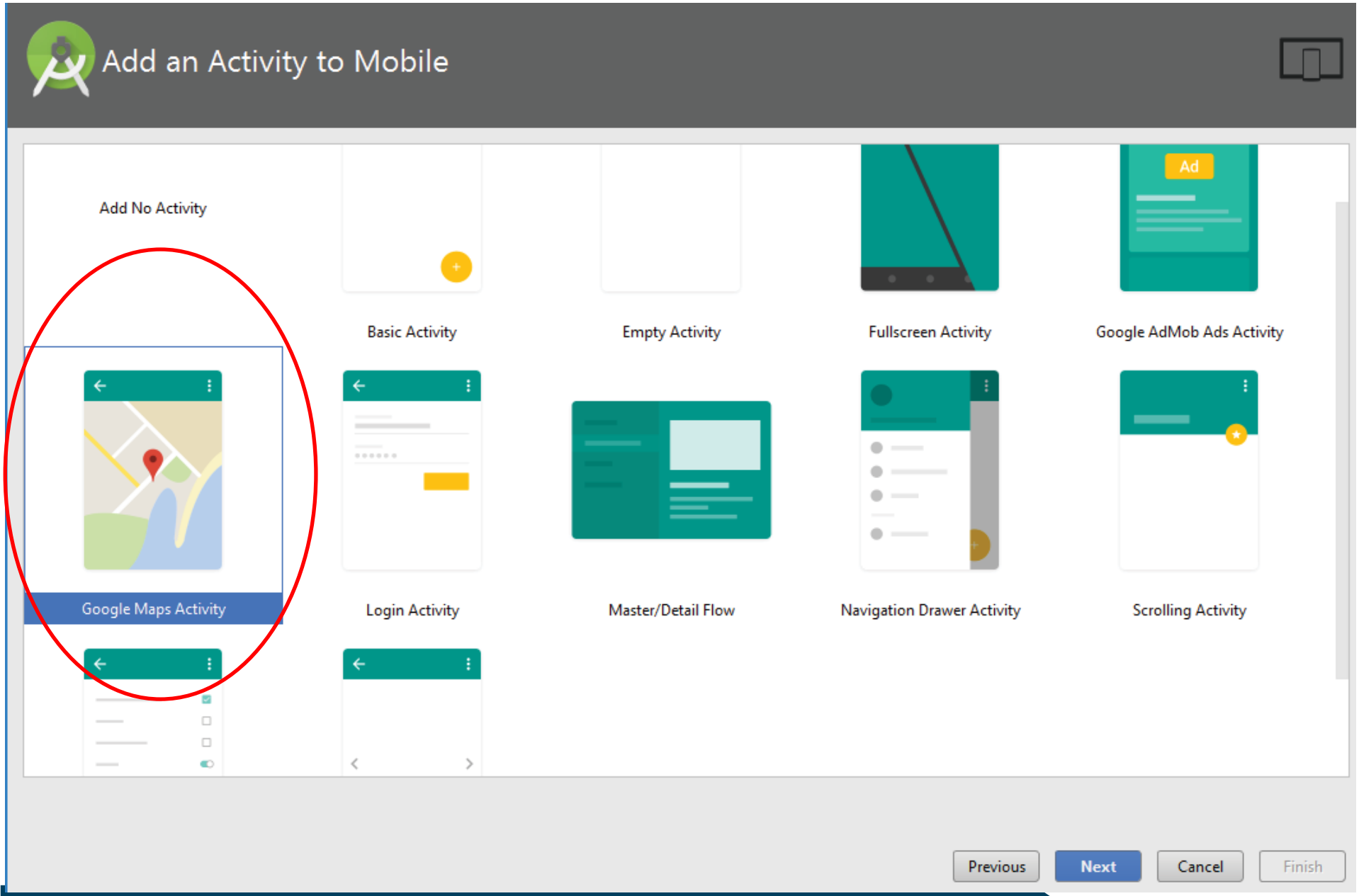
Creating the Project

- To get started, you need to first create an Android project so that you can display Google Maps in your activity:
    1. Using Android Studio, create an Android project and name it LBS.
    2. From the Create New Project Wizard, select Google Maps Activity as shown in Figure (next slide).

Obtaining the Maps API Key

- Beginning with the Android SDK release v1.0, you need to apply for a free Google Maps API key before you can integrate Google Maps into your Android application. When you apply for the key, you must also agree to Google's terms of use, so be sure to read them carefully.
- To get a Google Maps key, open the google_maps_api.xml file that was created in your LBS project. Within this file is a link to create a new Google Maps key. Simply copy and paste the link into your browser and follow the instructions. Make note of the key that Google gives you because you need it later in this project.

Creating the Project



Creating the Project

- Replace the YOUR_KEY_HERE placeholder in the google_maps_api.xml with your Google Maps key.
- To display Google Maps in your application, you first need the ACCESS_FINE_LOCATION permission in your manifest file. This is created for you automatically when you selected to set up a Google Maps
- Activity. You can see the line if you open the AndroidManifest.xml.

```
<uses-permission
 android:name="android.permission.ACCESS_FINE_LOCATION" />
```

- In order to test your application on the Android emulator, be sure to create an Emulator with an SDK version that includes Google Play Services as the selected target.

Program name: LBS

Displaying the Zoom Control

- The previous section showed how you can display Google Maps in your Android application.
- You can pan the map to any desired location and it updates on-the-fly.
- However, there is no way to use the emulator to zoom in or out from a particular location (on a real Android device you can pinch the map to zoom it).
- Thus, in this section, you find out how you can enable users to zoom in or out of the map using the built-in zoom controls.
- To display the built-in zoom controls, you must add a parameter to activity_maps.xml that sets the uiZoomControls to true:

```
map:uiZoomControls="true"
```

Displaying the Zoom Control

- Besides displaying the zoom controls, you can also programmatically zoom in or out of the map using the animateCamera() method of the GoogleMap class.

Program name: LBS.

- To handle key presses on your activity, you handle the onKeyDown event,

```
public boolean onKeyDown(int keyCode, KeyEvent event) {
switch (keyCode) {
      case KeyEvent.KEYCODE_3:
            mMap.animateCamera(CameraUpdateFactory.zoomIn());
            break;
      case KeyEvent.KEYCODE_1:
           mMap.animateCamera(CameraUpdateFactory.zoomOut());
           break;
      }
    return super.onKeyDown(keyCode, event);
}
```

Changing Views

- By default, Google Maps is displayed in map view, which is basically drawings of streets and places of interest.
- You can also set Google Maps to display in satellite view using the setMapType() method of the GoogleMap class:
- Remove Commented last line in onMapReady() method

Program name: LBS

Navigating to a Specific Location

- By default, Google Maps displays the map of Australia when it is first loaded.
- However, you can set Google Maps to display a particular location.
- To do so, you can use the moveCamera() method of the GoogleMap class.

```
  LatLng mysore = new LatLng(12.311827,   76.652985);
mMap.addMarker(new MarkerOptions().position(mysore).title("Marker
in Mysore"));
mMap.moveCamera(CameraUpdateFactory.newLatLng(mysore));
```

- In the preceding code, a LatLng object is created and set to the new coordinates of 12.311827 N, 76.652985 E (the coordinates of Mysore).
- To represent either W (west) or S (south) you use the negative value of the coordinate.

Program name: LBS

Getting the Location That Was Touched

- After using Google Maps for a while, you might want to know the latitude and longitude of a location corresponding to the position on the screen that was just touched.
- Knowing this information is very useful because you can determine a location's address—a process known as *reverse geocoding.*
- To get the latitude and longitude of a point on the Google Map that was touched, you must set a onMapClickListener.

Program name: LBS

Geocoding and Reverse Geocoding

- If you know the latitude and longitude of a location, you can find out its address using a process known as reverse geocoding.
- Google Maps in Android supports reverse geocoding via the Geocoder class.

Program name: LBSNew

- The Geocoder object converts the latitude and longitude into an address using the getFromLocation() method.
- After the address is obtained, you display it using the Toast class.
- We are only getting the address of a location that you touch.
- If you know the address of a location but want to know its latitude and longitude, you can do so via geocoding.
- Again, you can use the Geocoder class for this purpose.

Geocoding and Reverse Geocoding

- You can find the exact location of the Empire State Building by using the getFromLocationName().

```
Geocoder geoCoder = new Geocoder( getBaseContext(),
Locale.getDefault());
try {List<Address> addresses = geoCoder.getFromLocationName(
"empire state building", 5);
if (addresses.size() > 0) {
LatLng p = new LatLng(
(int) (addresses.get(0).getLatitude()),
(int) (addresses.get(0).getLongitude()));
mMap.moveCamera(CameraUpdateFactory.newLatLng(p));
}
} catch (IOException e) {
e.printStackTrace(); }
```

Getting Location Data

- Nowadays, mobile devices are commonly equipped with GPS receivers.
- Because of the many satellites orbiting the earth, you can use a GPS receiver to find your location easily.
- However, GPS requires a clear sky to work and hence does not always work indoors or where satellites can't penetrate (such as a tunnel through a mountain).
- Another effective way to locate your position is through *cell tower triangulation*.
- When a mobile phone is switched on, it is constantly in contact with base stations surrounding it.
- By knowing the identity of cell towers, it is possible to translate this information into a physical location through the use of various databases containing the cell towers' identities and their exact geographical locations.

Getting Location Data

- The advantage of cell tower triangulation is that it works indoors, without the need to obtain information from satellites.
- However, it is not as precise as GPS because its accuracy depends on overlapping signal coverage, which varies quite a bit.
- Cell tower triangulation works best in densely populated areas where the cell towers are closely located.
- A third method of locating your position is to rely on Wi-Fi triangulation.
- Rather than connect to cell towers, the device connects to a Wi-Fi network and checks the service provider against databases to determine the location serviced by the provider.
- Of the three methods described here, Wi-Fi triangulation is the least accurate.

Getting Location Data

- On the Android platform, the SDK provides the LocationManager class to help your device determine the user's physical location.

Program name: GmapSpecific

- In Android, location-based services are provided by the LocationManager class, located in the android.location package.
- Using the LocationManager class, your application can obtain periodic updates of the device's geographical locations, as well as fire an intent when it enters the proximity of a certain location.
- In the MapsActivity.java file, you first check for permission to use the Course Locations.
- Then you obtain a reference to the LocationManager class using the getSystemService() method.
- You do this in the onCreate() method of the LBSActivity:

Getting Location Data

```
//---use the LocationManager class to obtain locations data---
lm = (LocationManager)
getSystemService(Context.LOCATION_SERVICE);
locationListener = new MyLocationListener();
```

- Next, you create an instance of the MyLocationListener class, which you define later in the class.
- The MyLocationListener class implements the LocationListener abstract class. You need to override four methods in this implementation:
  - **onLocationChanged(Location location)—**Called when the location has changed
  - **onProviderDisabled(String provider)—**Called when the provider is disabled by the user
  - **onProviderEnabled(String provider)—**Called when the provider is enabled by the user
  - **onStatusChanged(String provider, int status, Bundle extras)—**Called when the provider status changes

Getting Location Data

- To be notified whenever there is a change in location, you needed to register a request for location changes so that your program can be notified periodically.
- You do this via the requestLocation- Updates() method:

```
if(permissionGranted) {
lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0,
locationListener);
}
```

- The requestLocationUpdates() method takes four arguments:

- **provider—**The name of the provider with which you register. In this case, you are using GPS to obtain your geographical location data.
- **minTime—**The minimum time interval for notifications, in milliseconds. 0 indicates that you want to be continually informed of location changes.

Getting Location Data

- **minDistance**—The minimum distance interval for notifications, in meters. 0 indicates that you want to be continually informed of location changes.
- **listener**—An object whose onLocationChanged() method will be called for each location update

❑ Finally, in the **onPause()** method, you remove the listener when the activity is destroyed or goes into the background (so that the application no longer listens for changes in location, thereby saving the battery of the device). You do that using the **removeUpdates()** method

Monitoring a Location

- One very cool feature of the LocationManager class is its ability to monitor a specific location.
- This is achieved using the addProximityAlert() method.
- The following code snippet shows how to monitor a particular location such that if the user is within a five-meter radius from that location, your application will fire an intent to launch the web browser.

- The addProximityAlert() method takes five arguments:
- Latitude
- Longitude
- Radius (in meters)
- Expiration (duration for which the proximity alert is valid, after which it is deleted; –1 for no expiration)
- Pending intent

Monitoring a Location

- Note that if the Android device's screen goes to sleep, the proximity is also checked once every four minutes in order to preserve the battery life of the device.