MODULE – 3: ANDROID UI DESIGN AND LOCATION BASED SERVICES

Syllabus: Views and View Groups – Basic Views – Fragments – Displaying Maps – Getting Location Data – Preparing for Publishing – Deploying APK Files

3.1 VIEWS

A view is a widget that has an appearance on screen. It is **derived from** the base class `android.view.View`. There are three types viz.

- Basic views
- Picker views
- List views

Each of these are discussed in the following sections.

3.1.1 Basic Views

Some of the basic views that can be used to design UI of Android application are:

- TextView
- EditText
- Button
- ImageButton
- CheckBox
- ToggleButton
- RadioButton
- RadioGroup

These basic views enable you to display text information, as well as perform some basic selection.

- **TextView View:** The TextView view is used to display text to the user. When we create a new Android project, Eclipse always creates one `<TextView>` element in `activity_main.xml` file, to display Hello World as shown below –

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      >
      <TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="@string/hello"
      />
</LinearLayout>
```

- **Button** — Represents a push-button widget
- **ImageButton** — Similar to the Button view, but it also displays an image
- **EditText** — A subclass of the TextView view, but it allows users to edit its text content
- **CheckBox** — A special type of button that has two states: checked or unchecked
- **RadioGroup** and **RadioButton** — The RadioButton has two states: either checked or unchecked. Once a RadioButton is checked, it cannot be unchecked. A RadioGroup is used to group together one or more RadioButton views, thereby allowing only one RadioButton to be checked within the RadioGroup.
- **ToggleButton** — Displays checked/unchecked states using a light indicator

To understand the behavior of these views, create a new android project and place the following code in `activity_main.xml` file, without disturbing existing code.

```
<Button android:id="@+id/btnSave"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="Save" />
<Button android:id="@+id/btnOpen"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Open" />
<ImageButton android:id="@+id/btnImg1"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:src="@drawable/icon" />
<EditText android:id="@+id/txtName"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content" />
<CheckBox android:id="@+id/chkAutosave"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="Autosave" />
<CheckBox android:id="@+id/star"
      style="?android:attr/starStyle"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content" />
<RadioGroup android:id="@+id/rdbGp1"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:orientation="vertical" >
<RadioButton android:id="@+id/rdb1"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="Option 1" />
<RadioButton android:id="@+id/rdb2"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="Option 2" />
</RadioGroup>
```

```
<ToggleButton android:id="@+id/toggle1"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content" />
```

After running the application, the output will be displayed as shown in the following diagram. Click on each of these views, to observe their default behavior.

3.1.2 Picker Views

Selecting the date and time is one of the common tasks you need to perform in a mobile application. Android supports this functionality through the `TimePicker` and `DatePicker` views. The `TimePicker` view enables users to select a time of the day, in either 24-hour mode or AM/PM mode. Using the `DatePicker`, you can enable users to select a particular date on the activity.

To add `TimePicker` in the android application, use the following code:

```
<TimePicker android:id="@+id/timePicker"
android:layout_width="wrap_content"
android:layout_height="wrap_content" />
```

To add `DatePicker` in the android application, use the following code:

```
<DatePicker android:id="@+id/datePicker"
android:layout_width="wrap_content"
android:layout_height="wrap_content" />
```

NOTE: A complete program to demonstrate `TimePicker` view is **Lab Program 10.**

3.1.3 List Views

List views are views that enable you to display a long list of items. In Android, there are two types of list views: `ListView` and `SpinnerView`. Both are useful for displaying long lists of items. The `ListView` displays a list of items in a vertically scrolling list.

To add `SpinnerView` in the android application, use the following code:

```
<Spinner android:id="@+id/spinner1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:drawSelectorOnTop="true" />
```

NOTE: Example for `ListView` view is **Lab Program 1.**

3.2 VIEW GROUPS

One or more views can be grouped together into a `ViewGroup`. A `ViewGroup` (which is itself a special type of view) provides the layout in which you can order the appearance and sequence of views. Examples of ViewGroups include `LinearLayout` and `FrameLayout`. A `ViewGroup` derives from the base class `android.view.ViewGroup`.

Each View and ViewGroup has a set of common attributes as shown in the following table.

ATTRIBUTE	DESCRIPTION
layout_width	Specifies the width of the View or ViewGroup
layout_height	Specifies the height of the View or ViewGroup
layout_marginTop	Specifies extra space on the top side of the View or ViewGroup
layout_marginBottom	Specifies extra space on the bottom side of the View or ViewGroup
layout_marginLeft	Specifies extra space on the left side of the View or ViewGroup
layout_marginRight	Specifies extra space on the right side of the View or ViewGroup
layout_gravity	Specifies how child Views are positioned
layout_weight	Specifies how much of the extra space in the layout should be allocated to the View
layout_x	Specifies the x-coordinate of the View or ViewGroup
layout_y	Specifies the y-coordinate of the View or ViewGroup

Some of these attributes are applicable only when a View is in a specific ViewGroup. For example, the `layout_weight` and `layout_gravity` attributes are applicable only when a View is in either a `LinearLayout` or a `TableLayout`.

Android supports the following ViewGroups:

- LinearLayout
- AbsoluteLayout
- TableLayout
- RelativeLayout
- FrameLayout
- ScrollView

Each of these ViewGroups are explained hereunder.

3.2.1 LinerLayout

- The LinearLayout arranges views in a single column or a single row.
- Child views can be arranged either vertically or horizontally.
- To see how LinearLayout works, consider the following elements typically contained in the activity_main.xml file:

```
<?xml version="1.0" encoding="Utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:orientation="Vertical" >
  <TextView
     android:layout_width="fill_parent"
     android:layout_height="wrap_content"
     android:text="string/hello" />
</LinearLayout>
```

- In the main.xml file, observe that the root element is <LinearLayout> and
- It has a <TextView> element contained within it.
- The <LinearLayout> element controls the order in which the views contained within it appear.
- Observe the following diagram.

3.2.2 AbsoluteLayout

The `AbsoluteLayout` enables you to specify the exact location of its children. Consider the following UI defined in main.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     xmlns:android=http://schemas.android.com/apk/res/android>
  <Button
     android:layout_width="188dp"
     android:layout_height="wrap_content"
     android:text="Button"
     android:layout_x="126px"
     android:layout_y="361px"/>
  <Button
     android:layout_width="113dp"
     android:layout_height="wrap_content"
     android:text="Button"
     android:layout_x="12px"
     android:layout_y="361px"/>
</AbsoluteLayout>
```

The above code will result into:

There is a problem with absolute layout when the activity is viewed on a high resolution screen. For this reason the AbslouteLayout has been deprecated since Android 1.5. It is not guaranteed to be supported in future version of android.

3.2.3 TableLayout

- The TableLayout groups views into rows and columns.
- <TableRow> element to designate a row in the table. Each row can contain one or more views.
- Each view placed within a row forms a cell.
- The width of each column is determined by the largest width of each cell in that column.

Consider the content of main.xml shown here:

```
<TableLayout
     xmlns:android=”http://schemas.android.com/apk/res/android”
     android:layout_height=”fill_parent”
     android:layout_width=”fill_parent”>
  <TableRow>
<TextView
     android:text=”User Name:”
     android:width =”120px” />
<EditText
     android:id=”@+id/txtUserName”
     android:width=”200px” />
</TableRow>
<TableRow>
<TextView
     android:text=”Password:” />
<EditText
     android:id=”@+id/txtPassword”
     android:password=”true” />
</TableRow>
<TableRow>
<TextView />
<CheckBox android:id=”@+id/chkRememberPassword”
     android:layout_width=”fill_parent”
     android:layout_height=”wrap_content”
     android:text=”Remember Password”/>
</TableRow>
<TableRow>
<Button
     android:id=”@+id/buttonSignIn”
     android:text=”Log In” />
</TableRow>
</TableLayout>
```

The above code result into following GUI:

3.2.4 RelativeLayout

The RelativeLayout enables you to specify how child views are positioned relative to each other. Consider the following main.xml file:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
android:id="@+id/RLayout"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
xmlns:android="http://schemas.android.com/apk/res/android"
>
<TextView
android:id="@+id/lblComments"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Comments"
android:layout_alignParentTop="true"
android:layout_alignParentLeft="true"
/>
<EditText
android:id="@+id/txtComments"
android:layout_width="fill_parent"
```

```
android:layout_height="170px"
android:textSize="18sp"
android:layout_alignLeft="@+id/lblComments"
android:layout_below="@+id/lblComments"
android:layout_centerHorizontal="true"
/>
<Button
android:id="@+id/btnSave"
android:layout_width="125px"
android:layout_height="wrap_content"
android:text="Save"
android:layout_below="@+id/txtComments"
android:layout_alignRight="@+id/txtComments"
/>
<Button
android:id="@+id/btnCancel"
android:layout_width="124px"
android:layout_height="wrap_content"
android:text="Cancel"
android:layout_below="@+id/txtComments"
android:layout_alignLeft="@+id/txtComments"
/>
</RelativeLayout>
```

The UI of the above code would look like –

- Each view is embedded within the relative layout has attributes that enable it to align with another view.
- The value for each of these attributes is the **ID** for the view that you are **referencing.**
- These **attributes** are as follows:
    - layout_alignParentTop
    - layout_alignParentLeft
    - layout_alignLeft

- layout_alignRight
- layout_below
- layout_centerHorizontal

3.2.5 FrameLayout

The FrameLayout is a placeholder on screen that you can use to **display** a single view. Views that you add to a FrameLayout are **always anchored** to the top left of the layout.

```
<RelativeLayout
android:id="@+id/RLayout"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
xmlns:android="http://schemas.android.com/apk/res/android"
>
<TextView
android:id="@+id/lblComments"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="This is my lovely dog, Ookii"
android:layout_alignParentTop="true"
android:layout_alignParentLeft="true"
/>
<FrameLayout
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignLeft="@+id/lblComments"
android:layout_below="@+id/lblComments"
android:layout_centerHorizontal="true"
>
<ImageView
android:src = "@drawable/ookii"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
/>
</FrameLayout>
</RelativeLayout>
```

Here, you have a FrameLayout within a RelativeLayout. Within the FrameLayout, you embed an ImageView. If you add another view (such as a Button view) within the FrameLayout, the view will overlap the previous view.

3.2.6 ScrollView

A ScrollView is a special type of FrameLayout in that it enables users to scroll through a list of views that occupy more space than the physical display. The ScrollView can contain only one child view or ViewGroup, which normally is a LinearLayout.

NOTE:

- Do not use a ListView together with the ScrollView.
- The ListView is designed for showing a list of related information and is optimized for dealing with large lists

```
<ScrollView
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     xmlns:android="http://schemas.android.com/apk/res/android" >
<LinearLayout
     android:layout_width="fill_parent"
     android:layout_height="wrap_content"
     android:orientation="Vertical">
</LinearLayout>
</ScrollView>
```

3.3 LOCATION BASED SERVICES

We have all seen the explosive growth of mobile apps in recent years. One category of apps that is very popular is location-based services, commonly known as LBS. LBS apps track your location, and may offer additional services such as locating amenities nearby, as well as offering suggestions for route planning, and so on. Of course, one of the key ingredients in a LBS app is maps, which present a visual representation of your location.

Here, will learn how to make use of the Google Maps in your Android application, and how to manipulate it programmatically. In addition, you will learn how to obtain your geographical location using the `LocationManager` class available in the Android SDK.

3.3.1 Displaying Maps

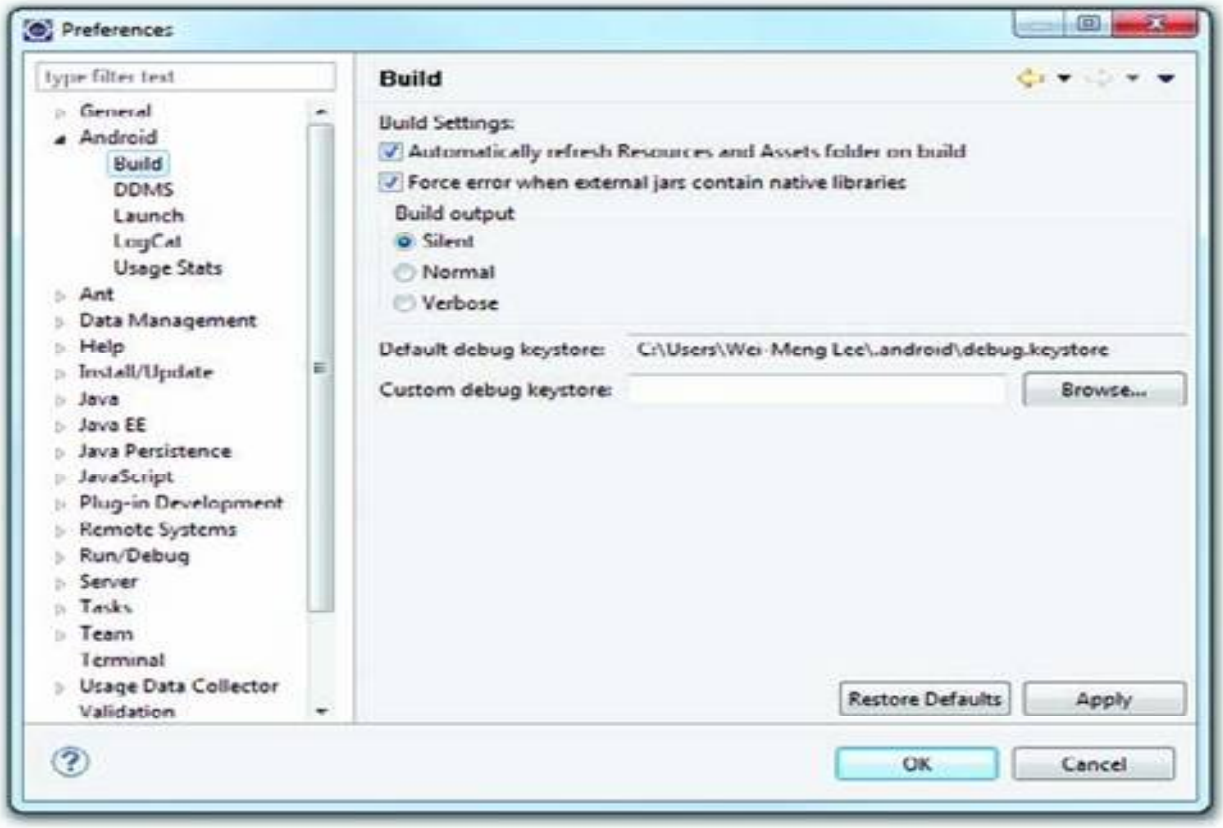
Google Maps is one of the many applications bundled with the Android platform. In addition to simply using the Maps application, you can also embed it into your own applications and make it do some very cool things. This section describes how to use Google Maps in your Android applications and programmatically 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).
- Add markers to Google Maps.

We will discuss how to build a project using maps.

Creating the Project: Create a new android project. In order to use Google Maps in your Android application, you need to ensure that you check the Google APIs as your build target. Google Maps is not part of the standard Android SDK, so you need to find it in the Google APIs add-on. If LBS is the name of your project, then you can see the additional JAR file (maps.jar) located under the Google APIs folder as below–

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.



First, if you are testing the application on the Android Emulator or an Android device directly connected to your development machine, locate the SDK debug certificate located in the default folder (C:\Users\<username>\.android for Windows 7 users). You can verify the existence of the debug certificate by going to Eclipse and selecting Window ➪

Preferences. Expand the Android item and select Build (as shown in figure above). On the right side of the window, you will be able to see the debug certificate's location.

The filename of the debug keystore is debug.keystore. This is the certificate that Eclipse uses to sign your application so that it may be run on the Android Emulator or devices.

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 this via the Geocoder class. You can retrieve the address of a location just touched using the getFromLocation() method. The Geocoder object converts the latitude and longitude into an address using the getFromLocation() method. Once the address is obtained, you display it using the Toast class.

NOTE: Lab Program 7 is the example for location based services.

3.3.2 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.
    - The advantage of cell tower triangulation is that it works indoors, without the need to obtain information from satellites.
    - 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

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

```
lm.requestLocationUpdates( LocationManager.GPS_PROVIDER, 0, 0,
locationListener);
```

This method takes four parameters:

1. Provider -The name of the provider with which you register. In this case, you are using GPS to obtain your geographical location data.
2. minTime - The minimum time interval for notifications, in milliseconds.
3. minDistance - The minimum distance interval for notifications, in meters.
4. Listener - An object whose onLocationChanged() method will be called for each location update

3.4 PREPARING FOR PUBLISHING

Google has made it relatively easy to publish your Android application so that it can be quickly distributed to end users. The steps to publishing your Android application generally involve the following:

- Export your application as an APK (Android Package) file.
- Generate your own self-signed certificate and digitally sign your application with it.
- Deploy the signed application.
- Use the Android Market for hosting and selling your application.

Here, you will learn how to prepare your application for signing, and then learn about the various ways to deploy your applications.

- **Versioning:** Beginning with version 1.0 of the Android SDK, the `AndroidManifest.xml` file of every Android application includes the `android:versionCode` and `android:versionName` attributes:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="net.learn2develop.LBS"
        android:versionCode="1"
        android:versionName="1.0">
```

  The `android:versionCode` attribute represents the version number of your application. For every revision you make to the application, you should increment this value by 1 so that you can programmatically differentiate the newest version from the previous one. This value is never used by the Android system, but is useful for developers as a means to obtain the version number of an application. However, the `android:versionCode` attribute is used by Android Market to determine if there is a newer version of your application available.

  The `android:versionName` attribute contains versioning information that is visible to the users. If you are planning to publish your application on the Android Market (means, playstore), the `AndroidManifest.xml` file must have the following attributes:
    - android:versionCode (within the <manifest> element)
    - android:versionName (within the <manifest> element)
    - android:icon (within the <application> element)
    - android:label (within the <application> element)

  The `android:label` attribute specifies the name of your application. This name will be displayed in the Settings ➪ Applications ➪ Manage Applications section of your Android device.

  In addition, if your application needs a minimum version of the SDK, you can specify it in the AndroidManifest.xml file using the <uses-sdk> element:

```
<uses-sdk android:minSdkVersion="7" />
```

- **Digitally Signing your Android Application:** All Android applications must be digitally signed before they are allowed to be deployed onto a device (or emulator). Unlike some mobile platforms, you need not purchase digital certificates from a certificate authority (CA) to sign your applications. Instead, you can generate your own self-signed certificate and use it to sign your Android applications.

  When you use Eclipse to develop your Android application and then press F11 to deploy it to an emulator, Eclipse automatically signs it for you. You can verify this by going to Windows ➪ Preferences in Eclipse, expanding the Android item, and selecting Build. Eclipse uses a default debug keystore (appropriately named "debug.keystore") to sign your application. ***A keystore is commonly known as a digital certificate.***

If you are publishing an Android application, you must sign it with your own certificate. Applications signed with the debug certificate cannot be published. While you can manually generate your own certificates using the keytool.exe utility provided by the Java SDK, Eclipse has made it easy for you by including a wizard that walks you through the steps to generate a certificate. It will also sign your application with the generated certificate (which you can also sign manually using the `jarsigner.exe` tool from the Java SDK).

3.5 DEPLOYING APK FILES

Once you have signed your APK fi les, you need a way to get them onto your users' devices. Three methods are here:

- Deploying manually using the adb.exe tool
- Hosting the application on a web server
- Publishing through the Android Market

Besides the above methods, you can install your applications on users' devices through e-mails, SD card, etc. As long as you can transfer the APK file onto the user's device, you can install the application.

Using the adb.exe Tool: Once your Android application is signed, you can deploy it to emulators and devices using the adb.exe (Android Debug Bridge) tool (located in the platform-tools folder of the Android SDK). Using the command prompt in Windows, navigate to the "<Android_SDK>\platform-tools" folder. To install the application to an emulator/device (assuming the emulator is currently up and running or a device is currently connected), issue the following command:

```
adb install "C:\Users\Wei-Meng Lee\Desktop\LBS.apk"
```

(Note that, here, LBS is name of the project)

Besides using the adb.exe tool to install applications, you can also use it to remove an installed application. To do so, you can use the shell option to remove an application from its installed folder:

```
adb shell rm /data/app/net.learn2develop.LBS.apk
```

Another way to deploy an application is to use the DDMS tool in Eclipse. With an emulator (or device) selected, use the File Explorer in DDMS to go to the /data/app folder and use the "Push a file onto the device" button to copy the APK file onto the device.

Using a Web Server: If you wish to host your application on your own, you can use a web server to do that. This is ideal if you have your own web hosting services and want to provide the application free of charge to your users or you can restrict access to certain groups of people. Following are the steps involved:

- Copy the signed LBS.apk fi le to c:\inetpub\wwwroot\. In addition, create a new HTML file named Install.html with the following content:

```
<html>
<title>Where Am I application</title>
<body>
Download the Where Am I application <a href=”LBS.apk”>here</a>
</body>
</html>
```

- On your web server, you may need to register a new MIME type for the APK file. The MIME type for the .apk extension is `application/vnd.android.package-archive`.
- From the Application settings menu, check the “Unknown sources” item. You will be prompted with a warning message. Click OK. Checking this item will allow the Emulator/device to install applications from other non-Market sources (such as from a web server).
- To install the LBS.apk application from the IIS web server running on your computer, launch the Browser application on the Android Emulator/device and navigate to the URL pointing to the APK file. To refer to the computer running the emulator, you should use the special IP address of 10.0.2.2.
- Alternatively, you can also use the IP address of the host computer. Clicking the “here” link will download the APK file onto your device. Drag the notification bar down to reveal the download status. To install the downloaded application, simply tap on it and it will show the permission(s) required by this application.
- Click the Install button to proceed with the installation. When the application is installed, you can launch it by clicking the Open button.

Besides using a web server, you can also e-mail your application to users as an attachment; when the users receive the e-mail they can download the attachment and install the application directly onto their device.

Publishing on Android Market: It is always better to host your application on Android market (Google Playstore). Steps involved in doing so, are explained hereunder:

- **Creating a Developer Profile:**
  - Create a developer profile at http://market.android.com/publish/Home using a Google account.
  - Pay one-time registration fees.
  - Agree Android Market Developer Distribution Agreement

- **Submitting Your Apps:** If you intend to charge for your application, click the Setup Merchant Account link located at the bottom of the screen. Here you enter additional information such as bank account and tax ID. You will be asked to supply some details for your application. Following are the compulsory details to be provided:
  - The application in APK format
  - At least two screenshots. You can use the DDMS perspective in Eclipse to capture screenshots of your application running on the Emulator or real device.

- A high-resolution application icon. This size of this image must be 512×512 pixels.
- Provide the title of your application, its description and recent update details.
- Indicate whether your application employs copy protection, and specify a content rating.

When all these setup is done, click Publish to publish your application on the Android Market.

Question Bank:

1. Define a View. Explain different types of views.
2. Discuss Basic Views in Android with a suitable code segment.
3. Briefly explain List Views. Design an application that contains Phone Contacts in vertical linear manner. Selected contact appears at the top of the list with a large italicized font and a blue background.(Lab Program 1)
4. Write a note on
   a. TimePicker View
   b. DatePicker View
5. Define viewgroup. List out various viewgroups in Android.
6. List out comman attributes of viewgroups.
7. Explain various viewgroups with suitable code segments.
8. How do you display Google maps in Android application?
9. Develop a mobile application that uses GPS location information. (Lab Program 7).
10. What are the different methods for getting location data? Explain.
11. How do you publish Android application? Explain the steps involved.
12. Discuss APK file deployment in detail.