Creating Android Applications

Working of Android Applications

- **WHAT MAKES AN ANDROID APPLICATION?**
- Android applications consist of loosely coupled components, bound by the application manifest that describes each component and how they interact.
- The manifest is also used to specify the application's metadata, its hardware and platform requirements, external libraries, and required permissions.

- The following components comprise the building blocks for all your Android applications:
- ‰ **Activities** — Your application's presentation layer. The UI of your application is built around one or more extensions of the Activity class. Activities use Fragments and Views to layout and display information, and to respond to user actions. Compared to desktop development, Activities are equivalent to Forms.
- ‰ **Services** — The invisible workers of your application. Service components run without a UI, updating your data sources and Activities, triggering Notifications, and broadcasting Intents. They're used to perform long running tasks, or those that require no user interaction (such as network lookups or tasks that need to continue even when your application's Activities aren't active or visible.)

- ‰ **Content Providers** — Shareable persistent data storage. Content Providers manage and persist application data and typically interact with SQL databases. They're also the preferred means to share data across application boundaries. You can configure your application's Content Providers to allow access from other applications, and you can access the Content Providers exposed by others. Android devices include several native Content Providers that expose useful databases such as the media store and contacts.
- ‰ **Intents** — A powerful interapplication message-passing framework. Intents are used extensively throughout Android. You can use Intents to start and stop Activities and Services, to broadcast messages system-wide or to an explicit Activity, Service, or Broadcast Receiver, or to request an action be performed on a particular piece of data.

- ‰ **Broadcast Receivers** — Intent listeners. Broadcast Receivers enable your application to listen for Intents that match the criteria you specify. Broadcast Receivers start your application to react to any received Intent, making them perfect for creating event-driven applications.
- ‰ **Widgets** — Visual application components that are typically added to the device home screen. A special variation of a Broadcast Receiver, widgets enable you to create dynamic, interactive application components for users to embed on their home screens.
- **Notifications** — Notifications enable you to alert users to application events without stealing focus or interrupting their current Activity. They're the preferred technique for getting a user's attention when your application is not visible or active, particularly from within a Service or Broadcast Receiver. For example, when a device receives a text message or an email, the messaging and Gmail applications use Notifications to alert you by flashing lights, playing sounds, displaying icons, and scrolling a text summary. You can trigger these notifications from your applications.

INTRODUCING THE APPLICATION MANIFEST FILE

- Each Android project includes a manifest fi le, AndroidManifest.xml, stored in the root of its project hierarchy. The manifest defines the structure and metadata of your application, its components, and its requirements.
- It includes nodes for each of the Activities, Services, Content Providers, and Broadcast Receivers that make up your application and, using Intent Filters and Permissions, determines how they interact with each other and with other applications.
- The manifest can also specify application metadata (such as its icon, version number, or theme), and additional top-level nodes can specify any required permissions, unit tests, and define hardware, screen, or platform requirements (as described next).
- The manifest is made up of a root manifest tag with a package attribute set to the project’s package. It should also include an xmlns:android attribute that supplies several system attributes used within the file.

- Use the versionCode attribute to define the current application version as an integer that increases with each version iteration, and use the versionName attribute to specify a public version that will be displayed to users.
- You can also specify whether to allow (or prefer) for your application be installed on external storage (usually an SD card) rather than internal storage using the installLocation attribute. To do this specify either preferExternal or auto, where the former installs to external storage whenever possible, and the latter asks the system to decide.

A Closer Look at the Application Manifest

- The following XML snippet shows a typical manifest node:
- <manifest xmlns:android=”http://schemas.android.com/apk/res/android”
- package=“com.paad.myapp“
- android:versionCode=“1“
- android:versionName=“0.9 Beta“
- android:installLocation=“preferExternal“>
- [... manifest nodes ...]
- </manifest>

- The manifest tag can include nodes that define the application components, security settings, test classes, and requirements that make up your application.
- The following list gives a summary of the available manifest sub-node tags and provides an XML snippet demonstrating how each tag is used:

- ‰ uses-sdk — This node enables you to define a minimum and maximum SDK version that must be available on a device for your application to function properly, and target SDK for which it has been designed using a combination of minSDKVersion, maxSDKVersion, and targetSDKVersion attributes, respectively.

- <uses-sdk android:minSdkVersion=”6”
- android:targetSdkVersion=”15”/>

- ‰ uses-configuration — The uses-configuration nodes specify each combination of input mechanisms are supported by your application. You shouldn’t normally need to include this node, though it can be useful for games that require particular input controls.

- You can specify any combination of input devices that include the following:
- ‰ reqFiveWayNav — Specify true for this attribute if you require an input device capable of navigating up, down, left, and right and of clicking the current selection. This includes both trackballs and directional pads (D-pads).
- ‰ reqHardKeyboard — If your application requires a hardware keyboard, specify true.
- ‰ reqKeyboardType — Lets you specify the keyboard type as one of nokeys, qwerty, twelvekey, or undefined.
- ‰ reqNavigation — Specify the attribute value as one of nonav, dpad, trackball, wheel, or undefined as a required navigation device.
- ‰ reqTouchScreen — Select one of notouch, stylus, finger, or undefined to specify the required touchscreen input.

- You can specify multiple supported confi gurations, for example, a device with a finger touchscreen, a trackball, and either a QUERTY or a twelve-key hardware keyboard, as shown here:
- <uses-configuration android:reqTouchScreen=”finger”
- android:reqNavigation=”trackball”
- android:reqHardKeyboard=”true”
- android:reqKeyboardType=”qwerty”/>
- <uses-configuration android:reqTouchScreen=”finger”
- android:reqNavigation=”trackball”
- android:reqHardKeyboard=”true”
- android:reqKeyboardType=”twelvekey”/>

- ‰ uses-feature — Android is available on a wide variety of hardware platforms. Use multiple uses-feature nodes to specify which hardware features your application requires.
- This prevents your application from being installed on a device that does not include a required piece of hardware, such as NFC hardware, as follows:
- <uses-feature android:name=”android.hardware.nfc” />

- You can require support for any hardware that is optional on a compatible device. Currently, optional hardware features include the following:
- ‰ **Audio** — For applications that requires a low-latency audio pipeline. Note that at the time of writing this book, no Android devices satisfi ed this requirement.
- ‰ **Bluetooth** — Where a Bluetooth radio is required.

- ‰ **Camera** — For applications that require a camera. You can also require (or set as options) autofocus, fl ash, or a front-facing camera.
- ‰ **Location** — If you require location-based services. You can also specify either network or GPS support explicitly.
- ‰ **Microphone** — For applications that require audio input.
- ‰ **NFC** — Requires NFC (near-fi eld communications) support.
- ‰ **Sensors** — Enables you to specify a requirement for any of the potentially available hardware sensors.
- ‰ **Telephony** — Specify that either telephony in general, or a specifi c telephony radio (GSM or CDMA) is required.
- ‰ **Touchscreen** — To specify the type of touch-screen your application requires.
- ‰ **USB** — For applications that require either USB host or accessory mode support.
- ‰ **Wi-Fi** — Where Wi-Fi networking support is required.

- As the variety of platforms on which Android is available increases, so too will the optional hardware. You can find a full list of uses-feature hardware at
- http://developer.android.com/guide/topics/manifest/uses-feature-element.html#features-reference.

- To ensure compatibility, requiring some permissions implies a feature requirement. In particular, requesting permission to access Bluetooth, the camera, any of the location service permissions, audio recording, Wi-Fi, and telephony-related permissions implies the corresponding hardware features. You can override these implied requirements by adding a required attribute and setting it to false — for example, a note-taking application that supports recording an audio note:
- <uses-feature android:name=”android.hardware.microphone”
- android:required=”false” />

- The camera hardware also represents a special case. For compatibility reasons requesting permission to use the camera, or adding a uses-feature node requiring it, implies a requirement for the camera to support autofocus. You can specify it as optional as appropriate:
- <uses-feature android:name=”android.hardware.camera” />
- <uses-feature android:name=”android.hardware.camera.autofocus”
- android:required=”false” />
- <uses-feature android:name=”android.hardware.camera.flash”
- android:required=”false” />

- ‰ supports-screens — The first Android devices were limited to 3.2" HVGA hardware. Since then, hundreds of new Android devices have been launched including tiny 2.55“ QVGA phones, 10.1" tablets, and 42" HD televisions. The supports-screen node enables you to specify the screen sizes your application has been designed and tested to. On devices with supported screens, your application is laid out normally using the scaling properties associated with the layout files you supply. On unsupported devices the system may apply a “compatibility mode,” such as pixel scaling to display your application. It’s best practice to create scalable layouts that adapt to all screen dimensions. You can use two sets of attributes when describing your screen support. The first set is used primarily for devices running Android versions prior to Honeycomb MR2 (API level 13). Each attribute takes a Boolean specifying support. As of SDK 1.6 (API level 4), the default value for each attribute is true, so use this node to specify the screen sizes you do not support.
- ‰ smallScreens — Screens with a resolution smaller than traditional HVGA (typically, QVGA screens).
- ‰ normalScreens — Used to specify typical mobile phone screens of at least HVGA, including WVGA and WQVGA.
- ‰ largeScreens — Screens larger than normal. In this instance a large screen is considered to be signifi cantly larger than a mobile phone display.
- ‰ xlargeScreens — Screens larger than large-typically tablet devices.

- ‰ requiresSmallestWidthDp — Enables you to specify a minimum supported screen width in device independent pixels. The smallest screen width for a device is the lower dimension of its screen height and width. This attribute can potentially be used to filter applications from the Google Play Store for devices with unsupported screens, so when used it should specify the absolute minimum number of pixels required for your layouts to provide a useable user experience.
- ‰ compatibleWidthLimitDp — Specifies the upper bound beyond which your application may not scale. This can cause the system to enable a compatibility mode on devices with screen resolutions larger than you specify.
- ‰ largestWidthLimitDp — Specifi es the absolute upper bound beyond which you know your application will not scale appropriately. Typically this results in the system forcing the application to run in compatibility mode (without the ability for users to disable it) on devices with screen resolutions larger than that specified.

- mode. Wherever possible, ensure that your layouts scale in a way that makes them usable on larger devices.
- <supports-screens android:smallScreens=”false”
- android:normalScreens=”true”
- android:largeScreens=”true”
- android:xlargeScreens=”true”
- android:requiresSmallestWidthDp=”480”
- android:compatibleWidthLimitDp=”600”
- android:largestWidthLimitDp=”720”/>

- ‰ supports-gl-texture — Declares that the application is capable of providing texture assets that are compressed using a particular GL texture compression format. You must use multiple supports-gl-texture elements if your application is capable of supporting multiple texture compression formats. You can find the most up-to-date list of supported GL texture compression format values at http://developer.android.com/guide/topics/manifest/
- supports-gl-texture-element.html.
- <supports-gl-texture android:name=”GL_OES_compressed_ETC1_RGB8_texture” />

- ‰ uses-permission — As part of the security model, uses-permission tags declare the user permissions your application requires. Each permission you specify will be presented to the user before the application is installed. Permissions are required for many APIs and method calls, generally those with an associated cost or security implication (such as dialing, receiving
- SMS, or using the location-based services).
- <uses-permission android:name=”android.permission.ACCESS_FINE_LOCATION”/>

- ‰ permission — Your application components can also create permissions to restrict access to shared application components. You can use the existing platform permissions for this purpose or define your own permissions in the manifest.
- To do this, use the permission tag to create a permission definition. Within the permission tag, you can specify the level of access the permission permits (normal, dangerous, signature, signatureOrSystem), a label, and an external resource containing the description that explains the risks of granting the specified permission.
- <permission android:name=”com.paad.DETONATE_DEVICE”
- android:protectionLevel=”dangerous”
- android:label=”Self Destruct”
- android:description=”@string/detonate_description”>
- </permission>

- ‰ instrumentation — Instrumentation classes provide a test framework for your application components at run time. They provide hooks to monitor your application and its interaction with the system resources. Create a new node for each of the test classes you've created for your application.
- <instrumentation android:label=”My Test”
- android:name=”.MyTestClass”
- android:targetPackage=”com.paad.apackage”>
- </instrumentation>

- ‰ application — A manifest can contain only one application node. It uses attributes to specify the metadata for your application (including its title, icon, and theme). During development you should include a debuggable attribute set to true to enable debugging, then be sure to disable it for your release builds.

- The application node also acts as a container for the Activity, Service, Content Provider, and Broadcast Receiver nodes that specify the application components.

- <application android:icon=”@drawable/icon”
- android:logo=”@drawable/logo”
- android:theme=”@android:style/Theme.Light”
- android:name=”.MyApplicationClass”
- android:debuggable=”true”>
- [... application nodes ...]
- </application>

- ‰ activity — An activity tag is required for every Activity within your application. Use the android:name attribute to specify the Activity class name. You must include the main launch Activity and any other Activity that may be displayed. Trying to start an Activity that’s not defined in the manifest will throw a runtime exception. Each Activity node supports intent-filter child tags that define the Intents that can be used to start the Activity.
- <activity android:name=”.MyActivity” android:label=”@string/app_name”>
- <intent-filter>
- <action android:name=”android.intent.action.MAIN” />
- <category android:name=»android.intent.category.LAUNCHER» />
- </intent-filter>
- </activity>

- ‰ service — As with the activity tag, add a service tag for each Service class used in your application. Service tags also support intent-filter child tags to allow late runtime binding.
- <service android:name=”.MyService”>
- </service>

- ‰ provider — Provider tags specify each of your application’s Content Providers. Content Providers are used to manage database access and sharing.
- <provider android:name=”.MyContentProvider”
- android:authorities=”com.paad.myapp.MyContentProvider”/>

- ‰ receiver — By adding a receiver tag, you can register a Broadcast Receiver without having to launch your application first. Broadcast Receivers are like global event listeners that, when registered, will execute whenever a matching Intent is broadcast by the system or an application. By registering a Broadcast Receiver in the manifest you can make this process entirely autonomous. If a matching Intent is broadcast, your application will be started automatically and the registered Broadcast Receiver will be executed. Each receiver node supports intent-filter child tags that defi ne the Intents that can be used to trigger the receiver:
- <receiver android:name=”.MyIntentReceiver”>
- <intent-filter>
- <action android:name=”com.paad.mybroadcastaction” />
- </intent-filter>
- </receiver>

- ‰ uses-library — Used to specify a shared library that this application requires. Maps, Geocoding, and Location-Based Services,” are packaged as a separate library that is not automatically linked.
- You can specify that a particular package is required — which prevents the application from being installed on devices without the specified library — or optional, in which case your application must use reflection to check for the library before attempting to make use of it.
- <uses-library android:name=”com.google.android.maps”
- android:required=”false”/>

Creating Resources

- Application resources are stored under the res folder in your project hierarchy. Each of the available resource types is stored in subfolders, grouped by resource type.
- Each resource type is stored in a different folder: simple values, Drawables, colors, layouts, animations, styles, menus, XML fi les (including searchables), and raw resources. When your application is built, these resources will be compiled and compressed as efficiently as possible and included in your application package.
- This process also generates an R class fi le that contains references to each of the resources you include in your project. This enables you to reference the resources in your code, with the advantage of design-time syntax checking.

Simple Values

- Supported simple values include strings, colors, dimensions, styles, and string or integer arrays.
- All simple values are stored within XML fi les in the res/values folder.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="app_name">To Do List</string>
  <plurals name="androidPlural">
    <item quantity="one">One android</item>
    <item quantity="other">%d androids</item>
  </plurals>
  <color name="app_background">#FF0000FF</color>
  <dimen name="default_border">5px</dimen>
  <string-array name="string_array">
    <item>Item 1</item>
    <item>Item 2</item>
    <item>Item 3</item>
  </string-array>
  <array name="integer_array">
    <item>3</item>
    <item>2</item>
    <item>1</item>
  </array>
</resources>
```

- **Strings**
- Externalizing your strings helps maintain consistency within your application and makes it much easier to internationalize them.
- String resources are specified with the string tag, as shown in the following XML snippet:
- <string name=”stop_message”>Stop.</string>
- Android supports simple text styling, so you can use the HTML tags , <i>, and <u> to apply bold, italics, or underlining, respectively, to parts of your text strings, as shown in the following
- example:
- <string name=”stop_message”>Stop.</string>

- **Colors**
- Use the color tag to define a new color resource. Specify the color value using a # symbol followed by the (optional) alpha channel, and then the red, green, and blue values using one or two hexadecimal numbers with any of the following notations:
- ‰ #RGB
- ‰ #RRGGBB
- ‰ #ARGB
- ‰ #AARRGGBB
- The following example shows how to specify a fully opaque blue and a partially transparent green:
- <color name=”opaque_blue”>#00F</color>
- <color name=”transparent_green”>#7700FF00</color>

- **Dimensions**
- Dimensions are most commonly referenced within style and layout resources. They’re useful for creating layout constants, such as borders and font heights.
- To specify a dimension resource, use the dimen tag, specifying the dimension value, followed by an identifier describing the scale of your dimension:
- ‰ px (screen pixels)
- ‰ in (physical inches)
- ‰ pt (physical points)
- ‰ mm (physical millimeters)
- ‰ dp (density-independent pixels)
- ‰ sp (scale-independent pixels)

- **Styles and Themes**
- Style resources let your applications maintain a consistent look and feel by enabling you to specify the attribute values used by Views. The most common use of themes and styles is to store the colors and fonts for an application.
- To create a style, use a style tag that includes a name attribute and contains one or more item tags. Each item tag should include a name attribute used to specify the attribute (such as font size or color) being defined. The tag itself should then contain the value, as shown in the following skeleton code.
- <?xml version=”1.0” encoding=”utf-8”?>
- <resources>
- <style name=”base_text”>
- <item name=”android:textSize”>14sp</item>
- <item name=”android:textColor”>#111</item>
- </style>
- </resources>

- **Drawables**
- Drawable resources include bitmaps and NinePatches (stretchable PNG images). They also include complex composite Drawables, such as LevelListDrawables and StateListDrawables, that can be defined in XML.

- **Layouts**
- Layout resources enable you to decouple your presentation layer from your business logic by designing UI layouts in XML rather than constructing them in code.
- You can use layouts to define the UI for any visual component, including Activities, Fragments, and Widgets. Once defined in XML, the layout must be "inflated" into the user interface. Within an Activity this is done using setContentView (usually within the onCreate method), whereas Fragment Views are inflated using the inflate method from the Inflator object passed in to the Fragment's onCreateView handler.

- **Menus**
- Create menu resources to design your menu layouts in XML, rather than constructing them in code.
- You can use menu resources to define both Activity and context menus within your applications, and provide the same options you would have when constructing your menus in code.
- When defined in XML, a menu is inflated within your application via the inflate method of the MenuInflator Service, usually within the onCreateOptionsMenu method.

Using Resources in Code

- Access resources in code using the static R class. R is a generated class based on your external resources, and created when your project is compiled. The R class contains static subclasses for each of the resource types for which you've defined at least one resource. For example, the default new project includes the R.string and R.drawable subclasses.

- Each of the subclasses within R exposes its associated resources as variables, with the variable names matching the resource identifi ers — for example, R.string.app_name or R.drawable.icon.

- The value of these variables is an integer that represents each resource’s location in the resource table, *not* an instance of the resource itself.
- Where a constructor or method, such as setContentView, accepts a resource identifier, you can pass in the resource variable, as shown in the following code snippet:
- // Inflate a layout resource.
- setContentView(R.layout.main);
- // Display a transient dialog box that displays the
- // error message string resource.
- Toast.makeText(this, R.string.app_error, Toast.LENGTH_LONG).show();

Referencing Resources Within Resources

- You can also use resource references as attribute values in other XML resources. This is particularly useful for layouts and styles, letting you create specialized variations on themes and localized strings and image assets. It’s also a useful way to support different images and spacing for a layout to ensure that it’s optimized for different screen sizes and resolutions.
- To reference one resource from another, use the @ notation, as shown in the following snippet:
- attribute=”@[packagename:]resourcetype/resourceidentifier”

- <?xml version=”1.0” encoding=”utf-8”?>
- <LinearLayout
- xmlns:android=”http://schemas.android.com/apk/res/android”
- android:orientation=”vertical”
- android:layout_width=”match_parent”
- android:layout_height=”match_parent”
- **android:padding=”@dimen/standard_border”>**
- <EditText
- android:id=”@+id/myEditText”
- android:layout_width=”match_parent”
- android:layout_height=”wrap_content”
- **android:text=”@string/stop_message”**
- **android:textColor=”@color/opaque_blue”**
- />
- </LinearLayout>

Using System Resources

- The Android framework makes many native resources available, providing you with various strings, images, animations, styles, and layouts to use in your applications.
- Accessing the system resources in code is similar to using your own resources. The difference is that you use the native Android resource classes available from android.R, rather than the applicationspecific R class.
- The following code snippet uses the getString method available in the application context to retrieve an error message available from the system resources:
- CharSequence httpError = getString(android.R.string.httpErrorBadUrl);

- To access system resources in XML, specify android as the package name, as shown in this XML snippet:
- <EditText
- android:id=”@+id/myEditText”
- android:layout_width=”match_parent”
- android:layout_height=”wrap_content”
- **android:text=”@android:string/httpErrorBadUrl”**
- **android:textColor=”@android:color/darker_gray”**
- />

Runtime Confi guration Changes

- Android handles runtime changes to the language, location, and hardware by terminating and restarting the active Activity. This forces the resource resolution for the Activity to be reevaluated and the most appropriate resource values for the new configuration to be selected.
- To have an Activity listen for runtime configuration changes, add an android:configChanges attribute to its manifest node, specifying the configuration changes you want to handle.
- The following list describes some of the configuration changes you can specify:
- ‰ mcc and mnc — A SIM has been detected and the mobile country or network code (respectively) has changed.
- ‰ locale — The user has changed the device’s language settings.

- ‰ keyboardHidden — The keyboard, d-pad, or other input mechanism has been exposed or hidden.
- ‰ keyboard — The type of keyboard has changed; for example, the phone may have a 12-key keypad that flips out to reveal a full keyboard, or an external keyboard might have been plugged in.
- ‰ fontScale — The user has changed the preferred font size.
- ‰ uiMode — The global UI mode has changed. This typically occurs if you switch between car mode, day or night mode, and so on.
- ‰ orientation — The screen has been rotated between portrait and landscape.
- ‰ screenLayout — The screen layout has changed; typically occurs if a different screen has been activated.
- ‰ screenSize — Introduced in Honeycomb MR2 (API level 12), occurs when the available screen size has changed, for example a change in orientation between landscape and portrait.
- ‰ smallestScreenSize — Introduced in Honeycomb MR2 (API level 12), occurs when the physical screen size has changed, such as when a device has been connected to an external display.

- <activity
- android:name=”.MyActivity”
- android:label=”@string/app_name”
- android:configChanges=”screenSize|orientation|keyboardHidden”>
- <intent-filter >
- <action android:name=”android.intent.action.MAIN” />
- <category android:name=”android.intent.category.LAUNCHER” />
- </intent-filter>
- </activity>

- Adding an android:configChanges attribute suppresses the restart for the specified configuration changes, instead triggering the onConfigurationChanged handler in the associated Activity.
- Override this method to handle the configuration changes yourself, using the passed-in Configuration object to determine the new configuration values, as shown in Listing 3-6. Be sure to call back to the superclass and reload any resource values that the Activity uses, in case they've changed.

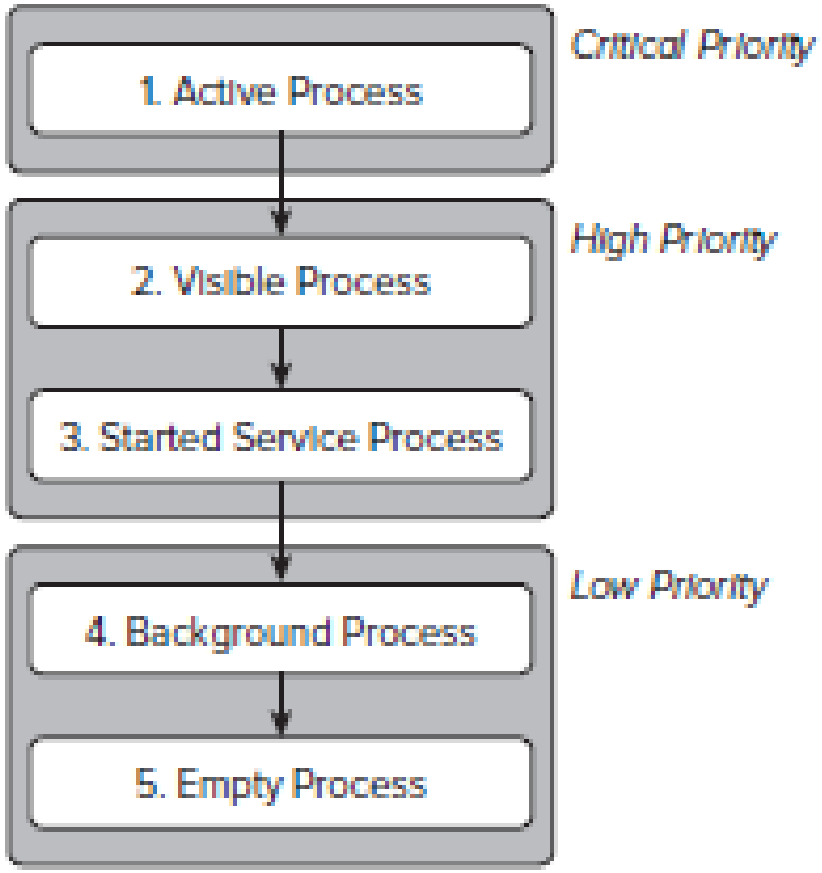
LISTING 3-6: Handling configuration changes in code

```
@Override
public void onConfigurationChanged(Configuration newConfig) {
  super.onConfigurationChanged(newConfig);

  // [ ... Update any UI based on resource values ... ]

  if (newConfig.orientation == Configuration.ORIENTATION_LANDSCAPE) {
    // [ ... React to different orientation ... ]
  }

  if (newConfig.keyboardHidden == Configuration.KEYBOARDHIDDEN_NO) {
    // [ ... React to changed keyboard visibility ... ]
  }
}
```

UNDERSTANDING AN APPLICATION'S PRIORITY AND ITS PROCESS' STATES

- The order in which processes are killed to reclaim resources is determined by the priority of their hosted applications. An application's priority is equal to that of its highest-priority component.
- If two applications have the same priority, the process that has been at that priority longest will be killed first. Process priority is also affected by interprocess dependencies; if an application has a dependency on a Service or Content Provider supplied by a second application, the secondary application has at least as high a priority as the application it supports.

- The following list details each of the application states shown in Figure , explaining how the state is determined by the application components of which it comprises:
- ‰ **Active processes** — Active (foreground) processes have application components the user is interacting with. These are the processes Android tries to keep responsive by reclaiming resources from other applications. There are generally very few of these processes, and they will be killed only as a last resort.



- Active processes include the following:
- ‰ Activities in an active state — that is, those in the foreground responding to user events.
- ‰ Broadcast Receivers executing onReceive event handlers.
- ‰ Services executing onStart, onCreate, or onDestroy event handlers.
- ‰ Running Services that have been fl agged to run in the foreground.

- ‰ **Visible processes** — Visible but inactive processes are those hosting “visible” Activities. As the name suggests, visible Activities are visible, but they aren’t in the foreground or responding to user events. This happens when an Activity is only partially obscured (by a non-full-screen or transparent Activity). There are generally very few visible processes, and they’ll be killed only under extreme circumstances to allow active processes to continue.

- ‰ **Started Service processes** — Processes hosting Services that have been started. Because these Services don’t interact directly with the user, they receive a slightly lower priority than visible Activities or foreground Services. Applications with running Services are still considered foreground processes and won’t be killed unless resources are needed for active or visible processes. When the system terminates a running Service it will attempt to restart them (unless you specify that it shouldn’t) when resources become available.
- ‰ **Background processes** — Processes hosting Activities that aren’t visible and that don’t have any running Services. There will generally be a large number of background processes that Android will kill using a last-seen-first-killed pattern in order to obtain resources for foreground processes.
- ‰ **Empty processes** — To improve overall system performance, Android will often retain an application in memory after it has reached the end of its lifetime. Android maintains this cache to improve the start-up time of applications when they’re relaunched. These processes are routinely killed, as required.

INTRODUCING THE ANDROID APPLICATION CLASS

- Your application’s Application object remains instantiated whenever your application runs. Unlike Activities, the Application is not restarted as a result of configuration changes.
- Extending the Application class with your own implementation enables you to do three things:
- ‰ Respond to application level events broadcast by the Android run time such as low memory conditions.
- ‰ Transfer objects between application components.
- ‰ Manage and maintain resources used by several application components.

- When your Application implementation is registered in the manifest, it will be instantiated when your application process is created. As a result, your Application implementation is by nature a singleton and should be implemented as such to provide access to its methods and member variables.

Overriding the Application Lifecycle Events

- The Application class provides event handlers for application creation and termination, low memory conditions, and configuration changes (as described in the previous section).
- By overriding these methods, you can implement your own application-specific behavior for each of these circumstances:
- ‰ onCreate — Called when the application is created. Override this method to initialize your application singleton and create and initialize any application state variables or shared resources.

- ‰ onLowMemory — Provides an opportunity for well-behaved applications to free additional memory when the system is running low on resources. This will generally only be called when background processes have already been terminated and the current foreground applications are still low on memory. Override this handler to clear caches or release unnecessary resources.
- ‰ onTrimMemory — An application specific alternative to the onLowMemory handler introduced in Android 4.0 (API level 13). Called when the run time determines that the current application should attempt to trim its memory overhead – typically when it moves to the background. It includes a level parameter that provides the context around the request.
- ‰ onConfigurationChanged — Unlike Activities Application objects are not restarted due to configuration changes. If your application uses values dependent on specific configurations, override this handler to reload those values and otherwise handle configuration changes at an application level.

- Overriding the Application Lifecycle Handlers
- public class MyApplication extends Application {
- private static MyApplication singleton;
- // Returns the application instance
- public static MyApplication getInstance() {
- return singleton;
- }
- **@Override**
- **public final void onCreate() {**
- **super.onCreate();**
- **singleton = this;**
- **}**
- **@Override**
- **public final void onLowMemory() {**
- **super.onLowMemory();**
- **}**
- **@Override**
- **public final void onTrimMemory(int level) {**
- **super.onTrimMemory(level);**
- **}**
- **@Override**
- **public final void onConfigurationChanged(Configuration newConfig) {**
- **super.onConfigurationChanged(newConfig);**
- **}**
- }

Building User Interfaces

- **FUNDAMENTAL ANDROID UI DESIGN**
- User interface (UI) design, user experience (UX), human computer interaction (HCI), and usability are huge topics.
- Android introduces some new terminology for familiar programming metaphors that will be explored in detail in the following sections:
- ‰ **Views** — Views are the base class for all visual interface elements (commonly known as *controls* or *widgets*). All UI controls, including the layout classes, are derived from View.

- ‰ **View Groups** — View Groups are extensions of the View class that can contain multiple child Views. Extend the ViewGroup class to create compound controls made up of interconnected child Views. The ViewGroup class is also extended to provide the Layout Managers that help you lay out controls within your Activities.

- ‰ **Fragments** — Fragments, introduced in Android 3.0 (API level 11), are used to encapsulate portions of your UI. This encapsulation makes Fragments particularly useful when optimizing your UI layouts for different screen sizes and creating reusable UI elements. Each Fragment includes its own UI layout and receives the related input events but is tightly bound to the Activity into which each must be embedded. Fragments are similar to UI View Controllers in iPhone development.

- ‰ **Activities** — Activities, described in detail in the previous chapter, represent the window, or screen, being displayed. Activities are the Android equivalent of Forms in traditional Windows desktop development. To display a UI, you assign a View (usually a layout or Fragment) to an Activity.

ANDROID USER INTERFACE FUNDAMENTALS

- All visual components in Android descend from the View class and are referred to generically as *Views*.

- The ViewGroup class is an extension of View designed to contain multiple Views. View Groups are used most commonly to manage the layout of child Views, but they can also be used to create atomic reusable components. View Groups that perform the former function are generally referred to as *layouts*.

Assigning User Interfaces to Activities

- A new Activity starts with a temptingly empty screen onto which you place your UI. To do so, call setContentView, passing in the View instance, or layout resource, to display. Because empty screens aren't particularly inspiring, you will almost always use setContentView to assign an Activity's UI when overriding its onCreate handler.

- The setContentView method accepts either a layout's resource ID or a single View instance. This lets you define your UI either in code or using the preferred technique of external layout resources.
- @Override
- public void onCreate(Bundle savedInstanceState) {
- super.onCreate(savedInstanceState);
- **setContentView(R.layout.main);**
- }

- You can obtain a reference to each of the Views within a layout using the findViewById method:
- **TextView myTextView = (TextView)findViewById(R.id.myTextView);**

- If you prefer the more *traditional* approach, you can construct the UI in code:
- @Override
- public void onCreate(Bundle savedInstanceState) {
- super.onCreate(savedInstanceState);
- **TextView myTextView = new TextView(this);**
- **setContentView(myTextView);**
- myTextView.setText(“Hello, Android”);
- }
- The setContentView method accepts a single View instance; as a result, you use layouts to add multiple controls to your Activity.

Android Layout Types

There are number of Layouts provided by Android which you will use in almost all the Android applications to provide different view, look and feel.

S.N.	Layout & Description
1	Linear Layout LinearLayout is a view group that aligns all children in a single direction, vertically or horizontally.
2	Relative Layout RelativeLayout is a view group that displays child views in relative positions.
3	Table Layout TableLayout is a view that groups views into rows and columns.
4	Absolute Layout AbsoluteLayout enables you to specify the exact location of its children.
5	Frame Layout The FrameLayout is a placeholder on screen that you can use to display a single view.
6	List View ListView is a view group that displays a list of scrollable items.
7	Grid View GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid.

Linear Layout

Android LinearLayout is a view group that aligns all children in a single direction, *vertically* or*horizontally*.

LinearLayout Attributes

Following are the important attributes specific to LinearLayout:

Attribute	Description
android:id	This is the ID which uniquely identifies the layout.
android:baselineAligned	This must be a boolean value, either "true" or "false" and prevents the layout from aligning its children's baselines.
android:divider	This is drawable to use as a vertical divider between buttons. You use a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".
android:gravity	This specifies how an object should position its content, on both the X and Y axes. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.
android:orientation	This specifies the direction of arrangmet and you will use "horizontal" for a row, "vertical" for a column. The default is horizontal.

```
<?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" >

   <Button android:id="@+id/btnStartService"
   android:layout_width="150px"
   android:layout_height="wrap_content"
   android:text="@string/start_service"

   <Button android:id="@+id/btnPauseService"
   android:layout_width="150px"
   android:layout_height="wrap_content"
   android:text="@string/pause_service"

   <Button android:id="@+id/btnStopService"
   android:layout_width="150px"
   android:layout_height="wrap_content"
   android:text="@string/stop_service"

</LinearLayout>
```


5556:VirtualAVD
3G
8:20
HelloWorld
Start
Pause
Stop
Basic Controls
Hardware Buttons
MENU
DPAD not enabled in AVD
Hardware Keyboard
Use your physical keyboard to provide input


5556:VirtualAVD
3G
8:17
HelloWorld
Start
Pause
Stop
Basic Controls
Hardware Buttons
MENU
DPAD not enabled in AVD
Hardware Keyboard
Use your physical keyboard to provide input

Relative Layout

Android RelativeLayout enables you to specify how child views are positioned relative to each other. The position of each view can be specified as relative to sibling elements or relative to the parent.

RelativeLayout Attributes

Following are the important attributes specific to RelativeLayout:

Attribute	Description
android:id	This is the ID which uniquely identifies the layout.
android:gravity	This specifies how an object should position its content, on both the X and Y axes. Possible values are top, bottom, left, right, center, center vertical, center horizontal etc.
android:ignoreGravity	This indicates what view should not be affected by gravity.

Using RelativeLayout, you can align two elements by right border, or make one below another, centered in the screen, centered left, and so on. By default, all child views are drawn at the top-left of the layout, so you must define the position of each view using the various layout properties available from**RelativeLayout.LayoutParams** and few of the important attributes are given below:

Attribute	Description
android:layout_above	Positions the bottom edge of this view above the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name"
android:layout_alignBottom	Makes the bottom edge of this view match the bottom edge of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
android:layout_alignLeft	Makes the left edge of this view match the left edge of the given anchor view ID and must be a reference to another resource, in the form

	"@[+][package:]type:name".
android:layout_alignParentBottom	If true, makes the bottom edge of this view match the bottom edge of the parent. Must be a boolean value, either "true" or "false".
android:layout_alignParentEnd	If true, makes the end edge of this view match the end edge of the parent. Must be a boolean value, either "true" or "false".
android:layout_alignParentLeft	If true, makes the left edge of this view match the left edge of the parent. Must be a boolean value, either "true" or "false".
android:layout_alignParentRight	If true, makes the right edge of this view match the right edge of the parent. Must be a boolean value, either "true" or "false".
android:layout_alignParentStart	If true, makes the start edge of this view match the start edge of the parent. Must be a boolean value, either "true" or "false".
android:layout_alignParentTop	If true, makes the top edge of this view match the top edge of the parent. Must be a boolean value, either "true" or "false".
android:layout_alignRight	Makes the right edge of this view match the right edge of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
android:layout_alignStart	Makes the start edge of this view match the start edge of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
android:layout_alignTop	Makes the top edge of this view match the top edge of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
android:layout_below	Positions the top edge of this view below the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
android:layout_centerHorizontal	If true, centers this child horizontally within its parent. Must be a boolean value, either "true" or "false".
android:layout_centerInParent	If true, centers this child horizontally and vertically within its parent. Must be a boolean value, either "true" or "false".
android:layout_centerVertical	If true, centers this child vertically within its parent. Must be a boolean value, either "true" or "false".
android:layout_toEndOf	Positions the start edge of this view to the end of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
android:layout_toLeftOf	Positions the right edge of this view to the left of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
android:layout_toRightOf	Positions the left edge of this view to the right of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".
android:layout_toStartOf	Positions the end edge of this view to the start of the given anchor view ID and must be a reference to another resource, in the form "@[+][package:]type:name".

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:paddingLeft="16dp"
   android:paddingRight="16dp" >
   <EditText
      android:id="@+id/name"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:hint="@string/reminder" />
   <TextView
      android:id="@+id/dates"
      android:layout_width="0dp"
      android:layout_height="wrap_content"
      android:layout_below="@id/name"
      android:layout_alignParentLeft="true"
```

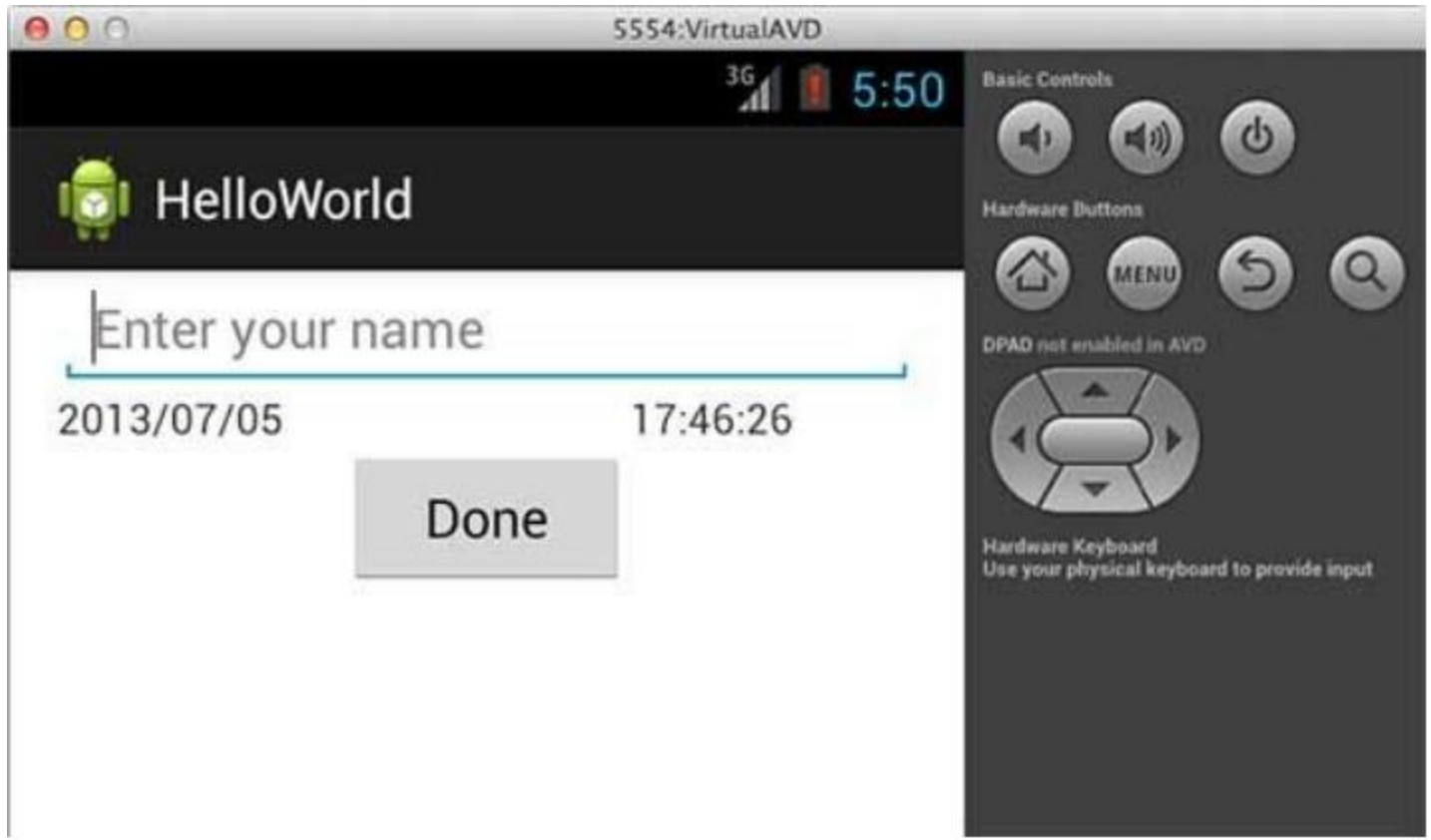

5554:VirtualAVD
3G
5:50
HelloWorld
Enter your name
2013/07/05
17:46:26
Done
Basic Controls
Hardware Buttons
MENU
DPAD not enabled in AVD
Hardware Keyboard
Use your physical keyboard to provide input

Table Layout

Android TableLayout groups views into rows and columns. You will use the <TableRow> element to build a row in the table. Each row has zero or more cells; each cell can hold one View object.

TableLayout containers do not display border lines for their rows, columns, or cells.

TableLayout Attributes

Following are the important attributes specific to TableLayout:

Attribute	Description
android:id	This is the ID which uniquely identifies the layout.
android:collapseColumns	This specifies the zero-based index of the columns to collapse. The column indices must be separated by a comma: 1, 2, 5.
android:collapseColumns	The zero-based index of the columns to shrink. The column indices must be separated by a comma: 1, 2, 5.
android:stretchColumns	The zero-based index of the columns to stretch. The column indices must be separated by a comma: 1, 2, 5.

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent">
   <TableRow>
      <Button
         android:id="@+id/backbutton"
         android:text="Back"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content" />
   </TableRow>
   <TableRow>
      <TextView
         android:text="First Name"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_column="1" />
         <EditText
         android:width="100px"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content" />
   </TableRow>
   <TableRow>
      <TextView
         android:text="Last Name"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:layout_column="1" />
```


5554:VirtualAVD
3G
6:18
HelloWorld
Back
First Name
Last Name
Basic Controls
Hardware Buttons
MENU
DPAD not enabled in AVD
Hardware Keyboard
Use your physical keyboard to provide input

Absolute Layout

An Absolute Layout lets you specify exact locations (x/y coordinates) of its children. Absolute layouts are less flexible and harder to maintain than other types of layouts without absolute positioning.

AbsoluteLayout Attributes

Following are the important attributes specific to AbsoluteLayout:

Attribute	Description
android:id	This is the ID which uniquely identifies the layout.
android:layout_x	This specifies the x-coordinate of the view.
android:layout_y	This specifies the y-coordinate of the view.

```
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent">

   <Button
      android:layout_width="100dp"
      android:layout_height="wrap_content"
      android:text="OK"
      android:layout_x="50px"
      android:layout_y="361px" />
```


5554:VirtualAVD
3G
6:35
HelloWorld
OK
Cancel
Basic Controls
Hardware Buttons
MENU
DPAD not enabled in AVD
Hardware Keyboard
Use your physical keyboard to provide input

Frame Layout

Frame Layout is designed to block out an area on the screen to display a single item. Generally, FrameLayout should be used to hold a single child view, because it can be difficult to organize child views in a way that's scalable to different screen sizes without the children overlapping each other.

You can, however, add multiple children to a FrameLayout and control their position within the FrameLayout by assigning gravity to each child, using the android:layout_gravity attribute.

FrameLayout Attributes

Following are the important attributes specific to FrameLayout:

Attribute	Description
android:id	This is the ID which uniquely identifies the layout.
android:foreground	This defines the drawable to draw over the content and possible values may be a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".
android:foregroundGravity	Defines the gravity to apply to the foreground drawable. The gravity defaults to fill. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.
android:measureAllChildren	Determines whether to measure all children or just those in the VISIBLE or INVISIBLE state when measuring. Defaults to false.

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent">

   <ImageView
   android:src="@drawable/ic_launcher"
   android:scaleType="fitCenter"
   android:layout_height="250px"
   android:layout_width="250px"/>

   <TextView
   android:text="Frame Demo"
   android:textSize="30px"
   android:textStyle="bold"
   android:layout_height="fill_parent"
   android:layout_width="fill_parent"
   android:gravity="center"/>
</FrameLayout>
```


5554:VirtualAVD
3G
7:18
HelloWorld
FarmeLayout Demo
Basic Controls
Hardware Buttons
MENU
DPAD not enabled in AVD
Hardware Keyboard
Use your physical keyboard to provide input

List View

Android **ListView** is a view which groups several items and display them in vertical scrollable list. The list items are automatically inserted to the list using an **Adapter** that pulls content from a source such as an array or database.

An adapter actually bridges between UI components and the data source that fill data into UI Component. Adapter can be used to supply the data to like spinner, list view, grid view etc.

The **ListView** and **GridView** are subclasses of **AdapterView** and they can be populated by binding them to an **Adapter**, which retrieves data from an external source and creates a View that represents each data entry.
Android provides several subclasses of Adapter that are useful for retrieving different kinds of data and building views for an AdapterView (ie. ListView or GridView). The two most common adapters are*ArrayAdapter* and *SimpleCursorAdapter*. We will see separate examples for both the adapters.

ListView Attributes

Following are the important attributes specific to GridView:

Attribute	Description
android:id	This is the ID which uniquely identifies the layout.
android:divider	This is drawable or color to draw between list items. .
android:dividerHeight	This specifies height of the divider. This could be in px, dp, sp, in, or mm.
android:entries	Specifies the reference to an array resource that will populate the ListView.
android:footerDividersEnabled	When set to false, the ListView will not draw the divider before each footer view. The default value is true.
android:headerDividersEnabled	When set to false, the ListView will not draw the divider after each header view. The default value is true.

ArrayAdapter

You can use this adapter when your data source is an array. By default, ArrayAdapter creates a view for each array item by calling toString() on each item and placing the contents in a **TextView**. Consider you have an array of strings you want to display in a ListView, initialize a new **ArrayAdapter** using a constructor to specify the layout for each string and the string array:

```
ArrayAdapter adapter = new ArrayAdapter<String>(this,
          R.layout.ListView,
          StringArray);
```

Here are arguments for this constructor:

- First argument **this** is the application context. Most of the case, keep it **this**.
- Second argument will be layout defined in XML file and having **TextView** for each string in the array.
- Final argument is an array of strings which will be populated in the text view.

Once you have array adaptor created, then simply call **setAdapter()** on your **ListView** object as follows:

```
ListView listView = (ListView) findViewById(R.id.listview);
listView.setAdapter(adapter);
```

You will define your list view under res/layout directory in an XML file. For our example we are going to using activity_main.xml file.

```
package com.example.helloworld;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.widget.ArrayAdapter;
import android.widget.ListView;

public class MainActivity extends Activity {

   // Array of strings...
    String[] countryArray = {"India", "Pakistan", "USA", "UK"};

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      ArrayAdapter adapter = new ArrayAdapter<String>(this,
      R.layout.activity_listview, countryArray);

      ListView listView = (ListView) findViewById(R.id.country_list);
      listView.setAdapter(adapter);
   }
}
```

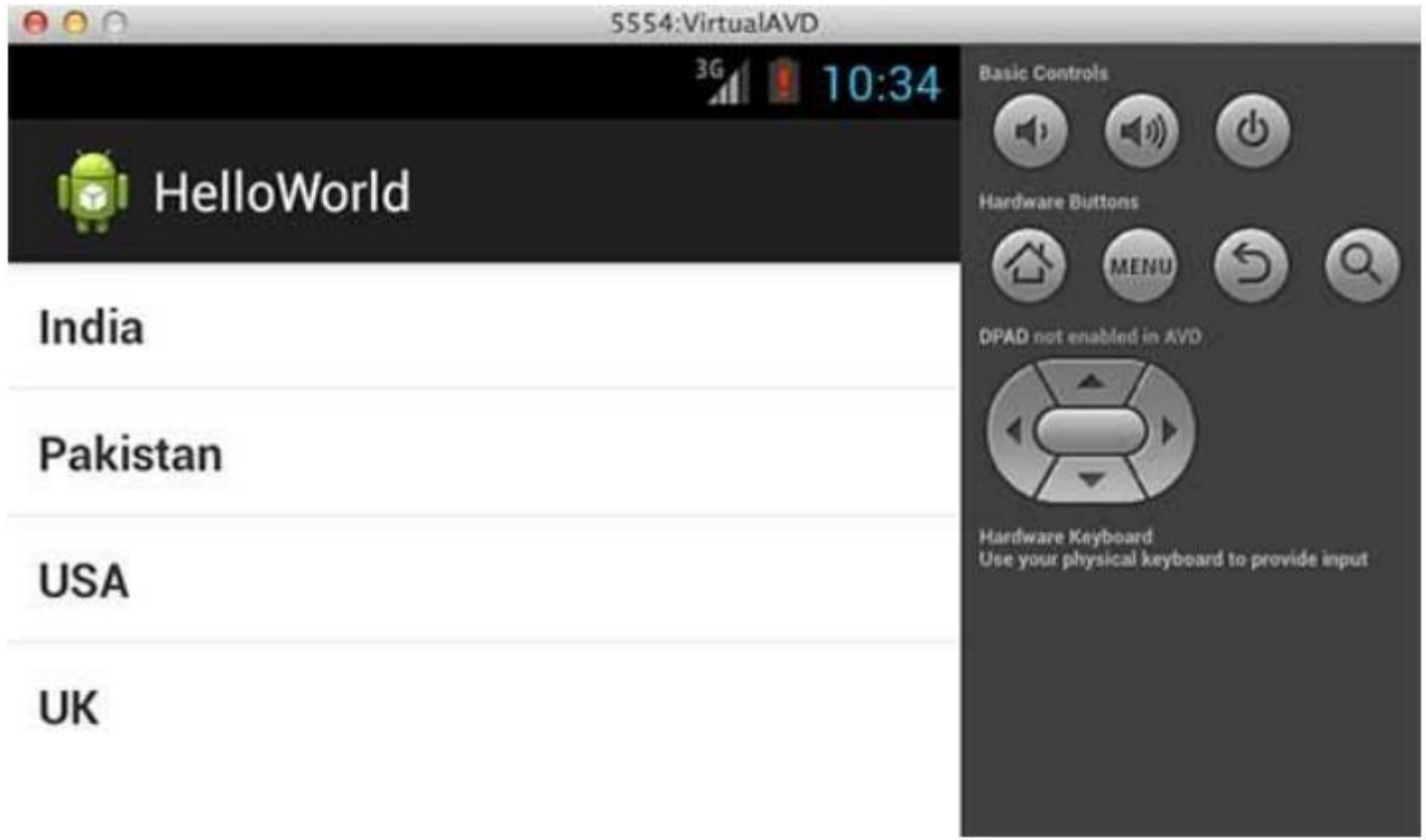
Following will be the content of **res/layout/activity_main.xml** file:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout width="match parent"
android:layout_height="match_parent"
android:orientation="vertical"
tools:context=".ListActivity" >

   <ListView
```

```
   android:id="@+id/country_list"
   android:layout_width="match_parent"
   android:layout_height="wrap_content" >
   </ListView>

</LinearLayout>
```

Following will be the content of **res/layout/activity_listview.xml** file:

```
<?xml version="1.0" encoding="utf-8"?>
<!--  Single List Item Design -->
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/label"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:padding="10dip"
        android:textSize="16dip"
        android:textStyle="bold" >
</TextView>
```


5554:VirtualAVD
3G
10:34
HelloWorld
India
Pakistan
USA
UK
Basic Controls
Hardware Buttons
MENU
DPAD not enabled in AVD
Hardware Keyboard
Use your physical keyboard to provide input

SimpleCursorAdapter

You can use this adapter when your data source is a database Cursor. When using*SimpleCursorAdapter*, you must specify a layout to use for each row in the **Cursor** and which columns in the Cursor should be inserted into which views of the layout.

For example, if you want to create a list of people's names and phone numbers, you can perform a query that returns a Cursor containing a row for each person and columns for the names and numbers. You then create a string array specifying which columns from the Cursor you want in the layout for each result and an integer array specifying the corresponding views that each column should be placed:

```
String[] fromColumns = {ContactsContract.Data.DISPLAY_NAME,
                        ContactsContract.CommonDataKinds.Phone.NUMBER};
int[] toViews = {R.id.display_name, R.id.phone_number};
```

When you instantiate the SimpleCursorAdapter, pass the layout to use for each result, the Cursor containing the results, and these two arrays:

```
SimpleCursorAdapter adapter = new SimpleCursorAdapter(this,
        R.layout.person_name_and_number, cursor, fromColumns, toViews, 0);

ListView listView = getListView();
listView.setAdapter(adapter);
```

The SimpleCursorAdapter then creates a view for each row in the Cursor using the provided layout by inserting each fromColumns item into the corresponding **toViews** view.

GridView

Android **GridView** shows items in two-dimensional scrolling grid (rows & columns) and the grid items are not necessarily predetermined but they automatically inserted to the layout using a **ListAdapter**

An adapter actually bridges between UI components and the data source that fill data into UI Component. Adapter can be used to supply the data to like spinner, list view, grid view etc.

The **ListView** and **GridView** are subclasses of **AdapterView** and they can be populated by binding them to an **Adapter**, which retrieves data from an external source and creates a View that represents each data entry.

GridView Attributes

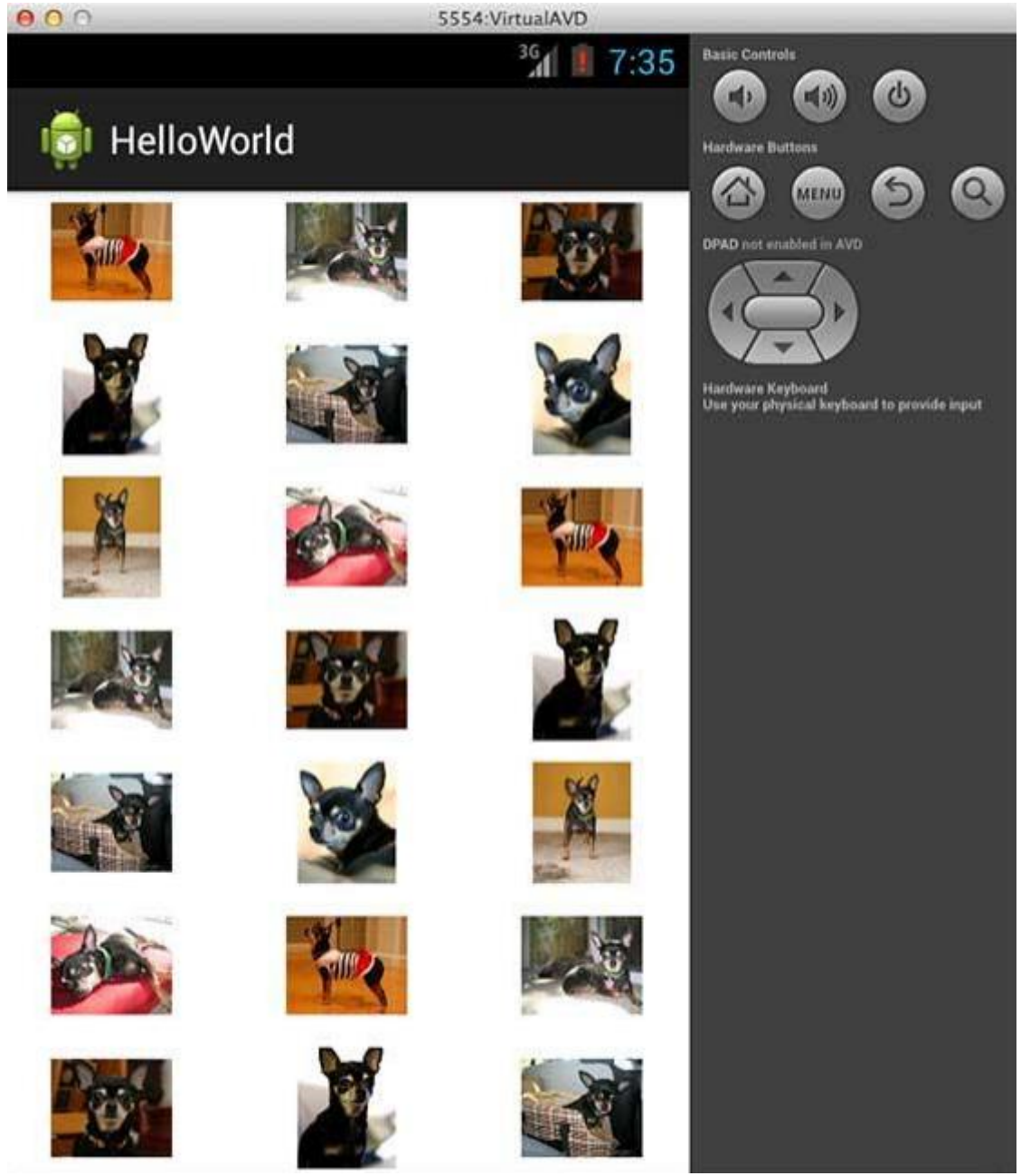
Following are the important attributes specific to GridView:

Attribute	Description
android:id	This is the ID which uniquely identifies the layout.
android:columnWidth	This specifies the fixed width for each column. This could be in px, dp, sp, in, or mm.
android:gravity	Specifies the gravity within each cell. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.
android:horizontalSpacing	Defines the default horizontal spacing between columns. This could be in px, dp, sp, in, or mm.
android:numColumns	Defines how many columns to show. May be an integer value, such as "100" or auto_fit which means display as many columns as possible to fill the available space.

android:stretchMode	Defines how columns should stretch to fill the available empty space, if any. This must be either of the values: none: Stretching is disabled. spacingWidth: The spacing between each column is stretched. columnWidth: Each column is stretched equally. spacingWidthUniform: The spacing between each column is uniformly stretched..
android:verticalSpacing	Defines the default vertical spacing between rows. This could be in px, dp, sp, in, or mm.

```
package com.example.helloworld;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.widget.GridView;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        GridView gridview = (GridView) findViewById(R.id.gridview);
        gridview.setAdapter(new ImageAdapter(this));
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }

}
```

```
<?xml version="1.0" encoding="utf-8"?>
<GridView xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/gridview"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:columnWidth="90dp"
   android:numColumns="auto_fit"
   android:verticalSpacing="10dp"
   android:horizontalSpacing="10dp"
   android:stretchMode="columnWidth"
   android:gravity="center"
/>
```


5554:VirtualAVD
3G
7:35
HelloWorld
Basic Controls
Hardware Buttons
MENU
DPAD not enabled in AVD
Hardware Keyboard
Use your physical keyboard to provide input

UI Controls

An Android application user interface is everything that the user can see and interact with. You have learned about the various layouts that you can use to position your views in an activity. This chapter will give you detail on various views.

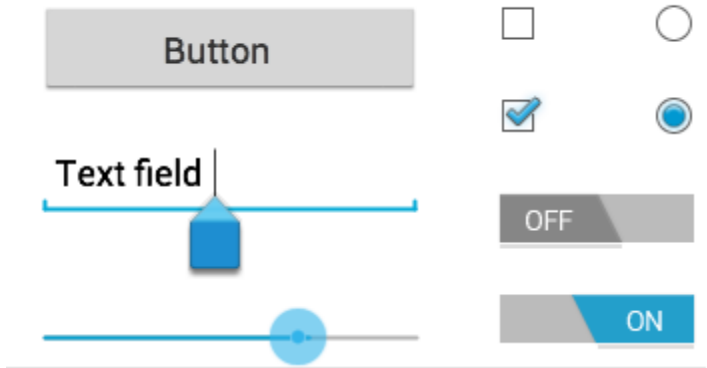
A **View** is an object that draws something on the screen that the user can interact with and a **ViewGroup** is an object that holds other View (and ViewGroup) objects in order to define the layout of the user interface.

Android UI Controls

There are number of UI controls provided by Android that allow you to build the graphical user interface for your app.

S.N.	UI Control & Description
1	TextView This control is used to display text to the user.
2	EditText EditText is a predefined subclass of TextView that includes rich editing capabilities.
3	AutoCompleteTextView The AutoCompleteTextView is a view that is similar to EditText, except that it shows a list of completion

	suggestions automatically while the user is typing.
4	Button A push-button that can be pressed, or clicked, by the user to perform an action.
5	ImageButton AbsoluteLayout enables you to specify the exact location of its children.
6	CheckBox An on/off switch that can be toggled by the user. You should use checkboxes when presenting users with a group of selectable options that are not mutually exclusive.
7	ToggleButton An on/off button with a light indicator.
8	RadioButton The RadioButton has two states: either checked or unchecked.
9	RadioGroup A RadioGroup is used to group together one or more RadioButtons.
10	ProgressBar The ProgressBar view provides visual feedback about some ongoing tasks, such as when you are performing a task in the background.
11	Spinner A drop-down list that allows users to select one value from a set.
12	TimePicker The TimePicker view enables users to select a time of the day, in either 24-hour mode or AM/PM mode.
13	DatePicker The DatePicker view enables users to select a date of the day.



Control Type	Description	Related Classes
Button	A push-button that can be pressed, or clicked, by the user to perform an action.	`Button`
Text field	An editable text field. You can use the `AutoCompleteTextView` widget to create a text entry widget that provides auto-complete suggestions	`EditText`, `AutoCompleteTextView`
Checkbox	An on/off switch that can be toggled by the user. You should use checkboxes when presenting users with a group of selectable options that are not mutually exclusive.	`CheckBox`
Radio button	Similar to checkboxes, except that only one option can be selected in the group.	`RadioGroup` `RadioButton`
Toggle button	An on/off button with a light indicator.	`ToggleButton`
Spinner	A drop-down list that allows users to select one value from a set.	`Spinner`
Pickers	A dialog for users to select a single value for a set by using up/down buttons or via a swipe gesture. Use a `DatePicker`code> widget to enter the values for the date (month, day, year) or a `TimePicker` widget to enter the values for a time (hour, minute, AM/PM), which will be formatted automatically for the user's locale.	`DatePicker`, `TimePicker`

TextView

A TextView displays text to the user and optionally allows them to edit it. A TextView is a complete text editor, however the basic class is configured to not allow editing.

TextView Attributes

Following are the important attributes related to TextView control. You can check Android official documentation for complete list of attributes and related methods which you can use to change these attributes are run time.

Attribute	Description
android:id	This is the ID which uniquely identifies the control.
android:capitalize	If set, specifies that this TextView has a textual input method and should automatically capitalize what the user types. Don't automatically capitalize anything - 0 Capitalize the first word of each sentence - 1 Capitalize the first letter of every word - 2

	Capitalize every character - 3
android:cursorVisible	Makes the cursor visible (the default) or invisible. Default is false.
android:editable	If set to true, specifies that this TextView has an input method.
android:fontFamily	Font family (named by string) for the text.
android:gravity	Specifies how to align the text by the view's x- and/or y-axis when the text is smaller than the view.
android:hint	Hint text to display when the text is empty.
android:inputType	The type of data being placed in a text field. Phone, Date, Time, Number, Password etc.
android:maxHeight	Makes the TextView be at most this many pixels tall.
android:maxWidth	Makes the TextView be at most this many pixels wide.
android:minHeight	Makes the TextView be at least this many pixels tall.
android:minWidth	Makes the TextView be at least this many pixels wide.
android:password	Whether the characters of the field are displayed as password dots instead of themselves. Possible value either "true" or "false".

android:phoneNumber	If set, specifies that this TextView has a phone number input method. Possible value either "true" or "false".
android:text	Text to display.
android:textAllCaps	Present the text in ALL CAPS. Possible value either "true" or "false".
android:textColor	Text color. May be a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".
android:textColorHighlight	Color of the text selection highlight.
android:textColorHint	Color of the hint text. May be a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".
android:textIsSelectable	Indicates that the content of a non-editable text can be selected. Possible value either "true" or "false".
android:textSize	Size of the text. Recommended dimension type for text is "sp" for scaled-pixels (example: 15sp).
android:textStyle	Style (bold, italic, bolditalic) for the text. You can use or more of the following values separated by '|'. normal - 0 bold - 1 italic - 2
android:typeface	Typeface (normal, sans, serif, monospace) for the text. You can use or more of the following values separated by '|'.

	normal - 0 sans - 1 serif - 2 monospace – 3

EditText

A EditText is an overlay over TextView that configures itself to be editable. It is the predefined subclass of TextView that includes rich editing capabilities.

EditText Attributes

Following are the important attributes related to EditText control. You can check Android official documentation for complete list of attributes and related methods which you can use to change these attributes are run time.

Inherited from **android.widget.TextView** Class:

Attribute	Description
android:autoText	If set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors.
android:drawableBottom	This is the drawable to be drawn below the text.
android:drawableRight	This is the drawable to be drawn to the right of the text.
android:editable	If set, specifies that this TextView has an input method.
android:text	This is the Text to display.

Inherited from **android.view.View** Class:

Attribute	Description
android:background	This is a drawable to use as the background.
android:contentDescription	This defines text that briefly describes content of the view.
android:id	This supplies an identifier name for this view,
android:onClick	This is the name of the method in this View's context to invoke when the view is clicked.
android:visibility	This controls the initial visibility of the view.

AutoCompleteTextView

A AutoCompleteTextView is a view that is similar to EditText, except that it shows a list of completion suggestions automatically while the user is typing. The list of suggestions is displayed in drop down menu. The user can choose an item from there to replace the content of edit box with.

AutoCompleteTextView Attributes

Following are the important attributes related to AutoCompleteTextView control. You can check Android official documentation for complete list of attributes and related methods which you can use to change these attributes are run time.

Attribute	Description
android:completionHint	This defines the hint displayed in the drop down menu.
android:completionHintView	This defines the hint view displayed in the drop down menu.
android:completionThreshold	This defines the number of characters that the user must type before completion suggestions are displayed in a drop down menu.
android:dropDownAnchor	This is the View to anchor the auto-complete dropdown to.
android:dropDownHeight	This specifies the basic height of the dropdown.
android:dropDownHorizontalOffset	The amount of pixels by which the drop down should be offset horizontally.
android:dropDownSelector	This is the selector in a drop down list.
android:dropDownVerticalOffset	The amount of pixels by which the drop down should be offset vertically.
android:dropDownWidth	This specifies the basic width of the dropdown.
android:popupBackground	This sets the background.


```
String[] arr = { "MS SQL SERVER", "MySQL", "Oracle" };

@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);

   autocompletetextview = (AutoCompleteTextView)
   findViewById(R.id.autoCompleteTextView1);

   ArrayAdapter adapter = new ArrayAdapter
   (this,android.R.layout.select_dialog_item, arr);

   autocompletetextview.setThreshold(1);
   autocompletetextview.setAdapter(adapter);
}
```

```
<AutoCompleteTextView
    android:id="@+id/autoCompleteTextView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/textView2"
    android:layout_below="@+id/textView2"
    android:layout_marginTop="54dp"
    android:ems="10" />
```

Button

A Button is a Push-button which can be pressed, or clicked, by the user to perform an action.

Button Attributes

Following are the important attributes related to Button control. You can check Android official documentation for complete list of attributes and related methods which you can use to change these attributes are run time.

Inherited from **android.widget.TextView** Class:

Attribute	Description
android:autoText	If set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors.
android:drawableBottom	This is the drawable to be drawn below the text.
android:drawableRight	This is the drawable to be drawn to the right of the text.
android:editable	If set, specifies that this TextView has an input method.
android:text	This is the Text to display.

Inherited from **android.view.View** Class:

Attribute	Description
android:background	This is a drawable to use as the background.
android:contentDescription	This defines text that briefly describes content of the view.
android:id	This supplies an identifier name for this view,
android:onClick	This is the name of the method in this View's context to invoke when the view is clicked.
android:visibility	This controls the initial visibility of the view.

Buttons

A button consists of text or an icon (or both text and an icon) that communicates what action occurs when the user touches it.

Depending on whether you want a button with text, an icon, or both, you can create the button in your layout in three ways:

- With text, using the `Button` class:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_text"
    ... />
```

- With an icon, using the `ImageButton` class:

```
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/button_icon"
    ... />
```

- With text and an icon, using the `Button` class with the `android:drawableLeft` attribute:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_text"
    android:drawableLeft="@drawable/button_icon"
    ... />
```

```
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/textView1"
    android:layout_below="@+id/edittext3"
    android:layout_marginTop="35dp"
    android:text="@string/click_button" />
```

Responding to Click Events

When the user clicks a button, the `Button` object receives an on-click event.

To define the click event handler for a button, add the `android:onClick` attribute to the `<Button>` element in your XML layout. The value for this attribute must be the name of the method you want to call in response to a click event. The `Activity` hosting the layout must then implement the corresponding method.

For example, here's a layout with a button using `android:onClick`:

```
<?xml version="1.0" encoding="utf-8"?>
<Button xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/button_send"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/button_send"
    android:onClick="sendMessage" />
```

Within the `Activity` that hosts this layout, the following method handles the click event:

```
/** Called when the user touches the button */
public void sendMessage(View view) {
    // Do something in response to button click
}
```

Using an OnClickListener

You can also declare the click event handler programmatically rather than in an XML layout. This might be necessary if you instantiate the `Button` at runtime or you need to declare the click behavior in a `Fragment` subclass.

To declare the event handler programmatically, create an `View.OnClickListener` object and assign it to the button by calling `setOnClickListener(View.OnClickListener)`. For example:

```
Button button = (Button) findViewById(R.id.button_send);
button.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        // Do something in response to button click
    }
});
```

ImageButton

A ImageButton is a AbsoluteLayout which enables you to specify the exact location of its children. This shows a button with an image (instead of text) that can be pressed or clicked by the user.

ImageButton Attributes

Following are the important attributes related to ImageButton control. You can check Android official documentation for complete list of attributes and related methods which you can use to change these attributes are run time.

Inherited from **android.widget.ImageView** Class:

Attribute	Description
android:adjustViewBounds	Set this to true if you want the ImageView to adjust its bounds to preserve the aspect ratio of its drawable.
android:baseline	This is the offset of the baseline within this view.
android:baselineAlignBottom	If true, the image view will be baseline aligned with based on its bottom edge.
android:cropToPadding	If true, the image will be cropped to fit within its padding.
android:src	This sets a drawable as the content of this ImageView.

Inherited from **android.view.View** Class:

Attribute	Description

android:background	This is a drawable to use as the background.
android:contentDescription	This defines text that briefly describes content of the view.
android:id	This supplies an identifier name for this view,
android:onClick	This is the name of the method in this View's context to invoke when the view is clicked.
android:visibility	This controls the initial visibility of the view.

```
<ImageButton
    android:id="@+id/imageButton1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignRight="@+id/textView1"
    android:layout_below="@+id/textView1"
    android:layout_marginRight="35dp"
    android:layout_marginTop="32dp"
    android:contentDescription=
    "@string/android_launcher_image"
    android:src="@drawable/ic_launcher" />
```

CheckBox Attributes

Following are the important attributes related to CheckBox control. You can check Android official documentation for complete list of attributes and related methods which you can use to change these attributes are run time.

Inherited from **android.widget.TextView** Class:

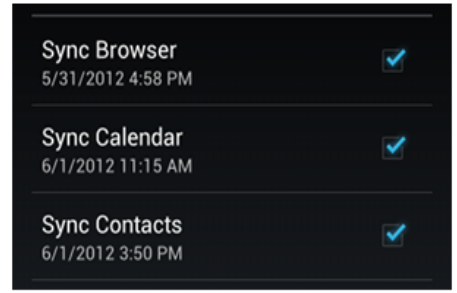
Attribute	Description
android:autoText	If set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors.
android:drawableBottom	This is the drawable to be drawn below the text.
android:drawableRight	This is the drawable to be drawn to the right of the text.
android:editable	If set, specifies that this TextView has an input method.
android:text	This is the Text to display.

Inherited from **android.view.View** Class:

Attribute	Description
android:background	This is a drawable to use as the background.
android:contentDescription	This defines text that briefly describes content of the view.
android:id	This supplies an identifier name for this view,
android:onClick	This is the name of the method in this View's context to invoke when the view is clicked.
android:visibility	This controls the initial visibility of the view.

Checkboxes

Checkboxes allow the user to select one or more options from a set. Typically, you should present each checkbox option in a vertical list.



To create each checkbox option, create a `CheckBox` in your layout. Because a set of checkbox options allows the user to select multiple items, each checkbox is managed separately and you must register a click listener for each one.

Responding to Click Events

When the user selects a checkbox, the `CheckBox` object receives an on-click event.

To define the click event handler for a checkbox, add the `android:onClick` attribute to the `<CheckBox>` element in your XML layout. The value for this attribute must be the name of the method you want to call in response to a click event. The `Activity` hosting the layout must then implement the corresponding method.

For example, here are a couple `CheckBox` objects in a list:

```
<?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">
    <CheckBox android:id="@+id/checkbox_meat"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/meat"
        android:onClick="onCheckboxClicked"/>
    <CheckBox android:id="@+id/checkbox_cheese"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/cheese"
        android:onClick="onCheckboxClicked"/>
</LinearLayout>
```

Within the `Activity` that hosts this layout, the following method handles the click event for both checkboxes:

```
public void onCheckboxClicked(View view) {
    // Is the view now checked?
    boolean checked = ((CheckBox) view).isChecked();

    // Check which checkbox was clicked
    switch(view.getId()) {
        case R.id.checkbox_meat:
            if (checked)
                // Put some meat on the sandwich
            else
                // Remove the meat
            break;
        case R.id.checkbox_cheese:
            if (checked)
                // Cheese me
            else
                // I'm lactose intolerant
            break;
        // TODO: Veggie sandwich
    }
}
```

ToggleButton

A ToggleButton displays checked/unchecked states as a button. It is basically an on/off button with a light indicator.

ToggleButton Attributes

Following are the important attributes related to ToggleButton control. You can check Android official documentation for complete list of attributes and related methods which you can use to change these attributes are run time.

Attribute	Description
android:disabledAlpha	This is the alpha to apply to the indicator when disabled.
android:textOff	This is the text for the button when it is not checked.
android:textOn	This is the text for the button when it is checked.

Inherited from **android.widget.TextView** Class:

Attribute	Description
android:autoText	If set, specifies that this TextView has a textual input method and automatically corrects some common spelling errors.
android:drawableBottom	This is the drawable to be drawn below the text.
android:drawableRight	This is the drawable to be drawn to the right of the text.

android:editable	If set, specifies that this TextView has an input method.
android:text	This is the Text to display.

Inherited from **android.view.View** Class:

Attribute	Description
android:background	This is a drawable to use as the background.
android:contentDescription	This defines text that briefly describes content of the view.
android:id	This supplies an identifier name for this view,
android:onClick	This is the name of the method in this View's context to invoke when the view is clicked.
android:visibility	This controls the initial visibility of the view.

Toggle Buttons

A toggle button allows the user to change a setting between two states.

You can add a basic toggle button to your layout with the `ToggleButton` object. Android 4.0 (API level 14) introduces another kind of toggle button called a switch that provides a slider control, which you can add with a `Switch` object.

If you need to change a button's state yourself, you can use the `CompoundButton.setChecked()` or `CompoundButton.toggle()` methods.

Toggle buttons

Switches (in Android 4.0+)

Responding to Button Presses

To detect when the user activates the button or switch, create an `CompoundButton.OnCheckedChangeListener` object and assign it to the button by calling `setOnCheckedChangeListener()`. For example:

```
ToggleButton toggle = (ToggleButton) findViewById(R.id.togglebutton);
toggle.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {
    public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
        if (isChecked) {
            // The toggle is enabled
        } else {
            // The toggle is disabled
        }
    }
});
```

```
<ToggleButton
    android:id="@+id/toggleButton1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/textView1"
    android:layout_below="@+id/textView1"
    android:layout_marginTop="24dp" />
```

```
private void addListenerOnToggleButton() {

  toggleBtn1 = (ToggleButton) findViewById
  (R.id.toggleButton1);
  toggleBtn2 = (ToggleButton) findViewById
  (R.id.toggleButton2);
  btResult = (Button) findViewById(R.id.button1);
  btResult.setOnClickListener(new OnClickListener() {

    @Override
    public void onClick(View v) {
      StringBuffer result = new StringBuffer();
      result.append("START Condition - ").append
      (toggleBtn1.getText());
      result.append("\nSTOP Condition - ").append
      (toggleBtn2.getText());
      Toast.makeText(MainActivity.this, result.toString(),
      Toast.LENGTH_SHORT).show();
    }
 });
}
```