Android is an operating system that powers mobile devices and is one of the most popular mobile platforms today. Android platform runs on hundreds of millions of mobile devices throughout the world. It's the largest installed operating system of any mobile operating system and growing rapidly day by day.

You can develop apps and games using Android and it gives you an open marketplace for distributing your apps and games instantly.

Android is the operating system for powering screens of all sizes. Android version is named after a dessert. The latest version of android is Android 9.0– Pie. Following table shows how the android platform evolves.

## HISTORY OF MOBILE APPLICATION DEVELOPMENT

To understand what makes Android so convincing, you must study how mobile development has evolved and how Android differs from other mobile platforms.

The Motorola DynaTAC 8000X was the first commercially available cell phone and it is of brick size. First-generation mobile were expensive, not particularly full featured and has Proprietary software.

As mobile phone prices dropped, batteries improved, and reception areas grew, more and more people began carrying these handy devices. Customers began pushing for more features and more games. They needed some way to provide a portal for entertainment and information services without allowing direct access to the handset.

Early phone have postage stamp-sized low-resolution screens and limited storage and processing power, these phones couldn't handle the data -intensive operations required by traditional web browsers. The bandwidth requirements for data transmission were also costly to the user.

**Wireless Application Protocol**

The Wireless Application Protocol (WAP) standard emerged to address above concerns. WAP was stripped-down version of HTTP. WAP browsers were designed to run within the memory and bandwidth constraints of the phone. Third-party WAP sites served up pages written in a mark-up language called Wireless Mark up Language (WML). The WAP solution was great for handset manufacturers and mobile operators. Phone users can access the news, stock market quotes, and sports scores on their phones.

WAP fell short of commercial expectations due to following reasons and Critics began to call WAP "Wait and Pay."

- Handset screens were too small for surfing.
- WAP browsers, especially in the early days, were slow and frustrating.
- Reading a sentence fragment at a time, and then waiting seconds for the next segment to download, ruined the user experience, especially because every second of downloading was often charged to the user.
- Mobile operators who provided the WAP portal often restricted which WAP sites were accessible.

## THE ANDROID PLATFORM

Andy Rubin has been credited as the father of the Android platform. His company, Android Inc., was acquired by Google in 2005. Working together, OHA members, including Google, began developing a non-proprietary open standard platform based upon technology developed at Android Inc. that would aim to solve the problems hindering the mobile community. The result is the Android project.

Most Android platform development is completed by Rubin's team at Google, where he acts as VP of Engineering and manages the Android platform roadmap. Google hosts the Android open source project and provides online Android documentation, tools, forums, and the Software Development Kit (SDK) for developers. All major Android news originates at Google.

**What is android?**

Android is called as "the first complete, open, and free mobile platform":

**Complete:** allows for rich application development opportunities.

**Open:** It is provided through open source licensing.

**Free:** Android applications are free to develop. Android applications can be distributed and commercialized in a variety of ways.

**Features of Android**

- Free and Open Source
- Familiar and inexpensive development tools
- Freely available SDK
- Familiar Language, Familiar Development Environments
- Reasonable learning curve for developers
- Enabling development of powerful applications
- Rich, secure application integration
- No costly obstacles to publication
- Free "Market" for application
- A new growing platform

**What it is not?**

Android is not:

- A Java ME implementation: Android applications are written in the Java language, but they are not run within a Java ME virtual machine, and Java-compiled classes and executable will not run natively in Android.
- Part of the Linux Phone Standards Forum or the Open Mobile Alliance: Android runs on an open-source Linux kernel, but, while their goals are similar, Android's complete software stack approach goes further than the focus of these standards-defining organizations.
- Simply an application layer (like UIQ or S60): While Android does include an application layer, ''Android'' also describes the entire software stack encompassing the underlying operating system, the API libraries, and the applications themselves.

- A mobile phone handset Android includes a reference design for mobile handset manufacturers, but there is no single ‘‘Android phone.’’ Instead, Android has been designed to support many alternative hardware devices.
- Google’s answer to the iPhone: The iPhone is a fully proprietary hardware and software platform released by a single company (Apple), while Android is an open-source software

## THE ANDROID VERSION

Android is the operating system for powering screens of all sizes. Android version is named after a dessert. The latest version of android is Android 9.0 – Pie. Following table shows how the android platform evolves.

ritics began to call WAP "Wait and Pay."

| Android Version | Name | Feature | API Level |
|---|---|---|---|
| **1.0** | Alpha | Web browser, Camera, Synchronization of Gmail, Contact and Calendar, Google Maps, Google Search, Google Talk, Instant Messaging, Text Messaging and MMS, Media Player, Notification, Voice Dialer, YouTube Video Player Other applications include: Alarm Clock, Calculator, Dialer (Phone), Home screen (Launcher), Pictures (Gallery), and Settings. | 1 |
| **1.1** | Beta | The update resolved bugs, changed the Android | 2 |

| Android Version | Name | Feature | API Level |
|---|---|---|---|
| | | API and added a number of features such as Details and reviews available when a user searches for businesses on Maps, Ability to show/hide dial pad and save attachments in messages. | |
| **1.5** | Cupcake | Virtual keyboards with text prediction and user dictionary for custom words, widgets, video recording and playback, Bluetooth, Copy and Paste, animated screen transition, auto rotation, upload video on YouTube, upload photo to Picasa. | 3 |
| **1.6** | Donut | Voice and text entry search, Multi-lingual speech synthesis, updated technology support for CDMA/EVDO, 802.1x, VPNs, and a text-to-speech engine, WVGA screen resolutions, Expanded Gesture framework and new Gesture Builder development tool | 4 |
| **2.0**<br>**2.0.1**<br>**2.1** | Éclair | Customize your home screen just the way you want it. Arrange apps and widgets across multiple screens and in folders. Stunning live wallpapers respond to your touch. | 5<br>6<br>7 |
| **2.2-2.2.3** | Froyo | Voice Typing lets you input text, while Voice Actions allow you to control your phone, just by speaking. | 8 |
| **2.3** | Gingerbread | New sensors make Android great for gaming – so you can touch, tap, tilt and play away. | 9-10 |
| **3.0** | Honeycomb | Optimized for tablets. | 11-13 |
| **4.0** | Ice Cream Sandwich | A new, refined design. Simple, beautiful and beyond smart. | 14-15 |
| **4.1-4.3** | Jelly Bean | Fast and smooth with slick graphics. With Google Now, you get just the right information | 16-18 |

| Android Version | Name | Feature | API Level |
|---|---|---|---|
| | | at the right time. | |
| **4.4** | Kit Kat | A more polished design, improved performance and new features. | 19-20 |
| **5.0** | Lolipop | Get the smarts of Android on screens big and small with the right information at the right moment. | 21-22 |
| **6.0** | Marshmallow | New App Drawer, Doze mode, Native finger print support, Android pay, USB type-C and USB 3.1 support, Direct share. | 23 |
| **7.0** | Nougat | Revamped notification, Split-screen use, file based encryption, direct boot, data saver | 24-25 |
| **8.0** | Oreo | Picture in picture, Google play protect, emoji | 26-27 |
| **9.0** | Pie | Adaptive Battery, adaptive brightness, intuitive navigation, dashboard, App timers, Wind down and do not disturb, Digital wellbeing. | 28 |

**Table-1 Android Versions**

## NATIVE ANDROID APPLICATIONS

Android phones will normally come with a suite of generic preinstalled applications that are part of the Android Open Source Project (AOSP), including, but not necessarily limited to:

- An e-mail client
- An SMS management application
- A full PIM (personal information management) suite including a calendar and contacts list
- A Web Kit-based web browser
- A music player and picture gallery
- A camera and video recording application
- A calculator

- The home screen
- An alarm clock

In many cases Android devices will also ship with the following proprietary Google mobile applications:

- The Android Market client for downloading third-party Android applications
- A fully-featured mobile Google Maps application including StreetView, driving directions and turn-by-turn navigation, satellite view, and traffic conditions
- The Gmail mail client
- The Google Talk instant-messaging client
- The YouTube video player

## ANDROID SDK FEATURES

The true appeal of Android as a development environment lays in the APIs it provides. As an application-neutral platform, Android gives you the opportunity to create applications that are as much a part of the phone as anything provided out of the box. The following list highlights some of the most noteworthy Android features:

- No licensing, distribution, or development fees or release approval processes
- Wi-Fi hardware access
- GSM, EDGE, and 3G networks for telephony or data transfer, enabling you to make or receive calls or SMS messages, or to send and retrieve data across mobile networks
- Comprehensive APIs for location-based services such as GPS
- Full multimedia hardware control, including playback and recording with the camera and microphone
- APIs for using sensor hardware, including accelerometers and the compass
  - Libraries for using Bluetooth for peer-to-peer data transfer
  - IPC message passing
  - Shared data stores
  - Background applications and processes
  - Home-screen Widgets, Live Folders, and Live Wallpaper
  - The ability to integrate application search results into the system search
  - An integrated open-source HTML5WebKit-based browser

- Full support for applications that integrate map controls as part of their user interface
- Mobile-optimized hardware-accelerated graphics, including a path-based 2D graphics library and support for 3D graphics using OpenGL ES 2.0
- Media libraries for playing and recording a variety of audio/video or still image formats
- Localization through a dynamic resource framework
- An application framework that encourages reuse of application components and the replacement of native applications

## ANDROID ARCHITECTURE

Android is an open source, Linux-based software stack created for a wide array of devices and form factors. The following diagram shows the major components of the Android platform.

**The Linux Kernel**

The foundation of the Android platform is the Linux kernel. For example, the Android Runtime (ART) relies on the Linux kernel for underlying functionalities such as threading and low-level memory management. Using a Linux kernel allows Android to take advantage of key security features and allows device manufacturers to develop hardware drivers for a well-known kernel.

**Hardware Abstraction Layer (HAL)**

The hardware abstraction layer (HAL) provides standard interfaces that expose device hardware capabilities to the higher-level Java API framework. The HAL consists of multiple library modules, each of which implements an interface for a

specific type of hardware component, such as the camera or Bluetooth module. When a framework API makes a call to access device hardware, the Android system loads the library module for that hardware component.

**Android Runtime**

For devices running Android version 5.0 (API level 21) or higher, each app runs in its own process and with its own instance of the Android Runtime (ART). ART is written to run multiple virtual machines on low-memory devices by executing DEX files, a byte code format designed especially for Android that's optimized for minimal memory footprint. Build tool chains, such as Jack, compile Java sources into DEX byte code, which can run on the Android platform.

Some of the major features of ART include the following:

- Ahead-of-time (AOT) and just-in-time (JIT) compilation
- Optimized garbage collection (GC)
- On Android 9 (API level 28) and higher, conversion of an app package's Dalvik Executable format (DEX) files to more compact machine code.
- Better debugging support, including a dedicated sampling profiler, detailed diagnostic exceptions and crash reporting, and the ability to set watch points to monitor specific fields

Prior to Android version 5.0 (API level 21), Dalvik was the Android runtime. If your app runs well on ART, then it should work on Dalvik as well, but the reverse may not be true. Android also includes a set of core runtime libraries that provide most of the functionality of the Java programming language, including some Java 8 language features that the Java API framework uses.

**Native C/C++ Libraries**

Many core Android system components and services, such as ART and HAL, are built from native code that requires native libraries written in C and C++. The Android platform provides Java framework APIs to expose the functionality of some of these native libraries to apps. For example, you can access OpenGL ES through the Android framework's Java OpenGL API to add support for drawing and manipulating 2D and 3D graphics in your app. If you are developing an app that requires C or C++

code, you can use the Android NDK to access some of these native platform libraries directly from your native code.

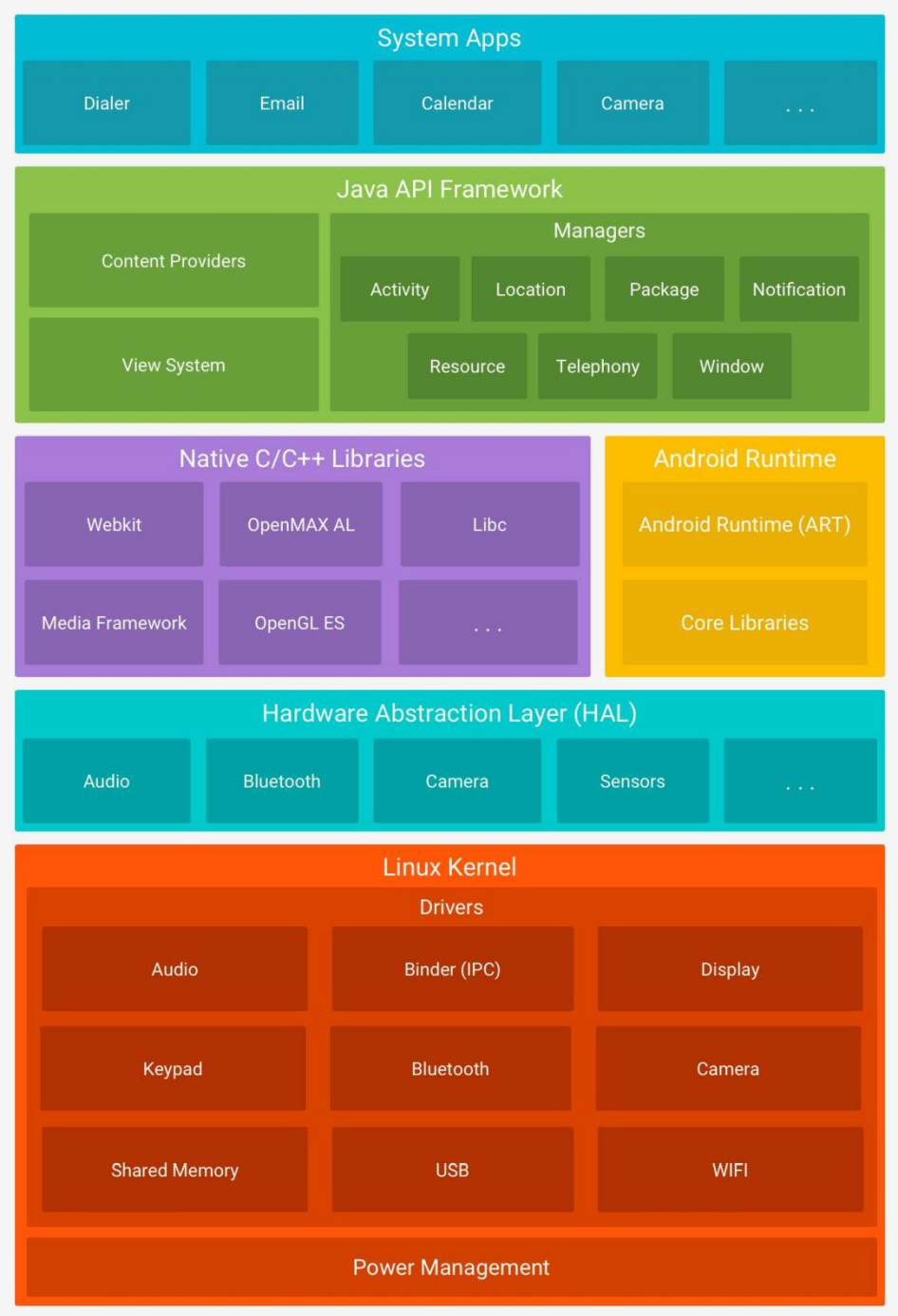



**Figure-1: Android Software Stack**

### Java API Framework

The entire feature-set of the Android OS is available to you through APIs written in the Java language. These APIs form the building blocks you need to create Android

apps by simplifying the reuse of core, modular system components and services, which include the following:

- A rich and extensible View System you can use to build an app's UI, including lists, grids, text boxes, buttons, and even an embeddable web browser
- A Resource Manager, providing access to non-code resources such as localized strings, graphics, and layout files
- A Notification Manager that enables all apps to display custom alerts in the status bar
- An Activity Manager that manages the lifecycle of apps and provides a common navigation back stack
- Content Providers that enable apps to access data from other apps, such as the Contacts app, or to share their own data

Developers have full access to the same framework APIs that Android system apps use.

**System Apps**

Android comes with a set of core apps for email, SMS messaging, calendars, internet browsing, contacts, and more. Apps included with the platform have no special status among the apps the user chooses to install. So a third-party app can become the user's default web browser, SMS messenger, or even the default keyboard (some exceptions apply, such as the system's Settings app).

The system apps function both as apps for users and to provide key capabilities that developers can access from their own app. For example, if your app would like to deliver an SMS message, you don't need to build that functionality you—you can instead invoke whichever SMS app is already installed to deliver a message to the recipient you specify.

## Factors that affect Mobile Application development

You should keep in mind the following factors while developing mobile application:

- Low processing power
- Limited RAM

- Limited permanent storage capacity
- Small screens with low resolution
- High costs associated with data transfer
- Slow data transfer rates with high latency
- Unreliable data connections
- Limited battery life

**Following are some of the factors that affect app development time:**

- User Interface & User Experience.
- Custom application
- Resource availability
- App security and publishing the app
- App designing
- Number of screens/devices/platforms/operating systems
- Third party integration
- Features
- Understanding the business logic
- Complexity of the App

## Introduction

For developing application for android platform, you will require Integrated Development Environment (IDE). Android Studio is the official IDE for Android application development. Android Studio provides everything you need to start developing apps for Android, including the Android Studio IDE and the Android SDK tools. First we discuss what the system requirements for android studio are and how to install and configure android studio.

## System Requirements for Android Studio

**Windows**

- Microsoft® Windows® 8/7/Vista/2003 (32 or 64-bit)
-

GB RAM minimum, 4 GB RAM recommended

- 400 MB hard disk space
- At least 1 GB for Android SDK, emulator system images, and caches
- 1280 x 800 minimum screen resolution
- Java Development Kit (JDK) 7
- Optional for accelerated emulator: Intel® processor with support for Intel® VT-x, Intel® EM64T (Intel® 64), and Execute Disable (XD) Bit functionality

## Downloading Android Studio

Download android studio from http://developer.android.com/sdk/index.html. It will open following web page.

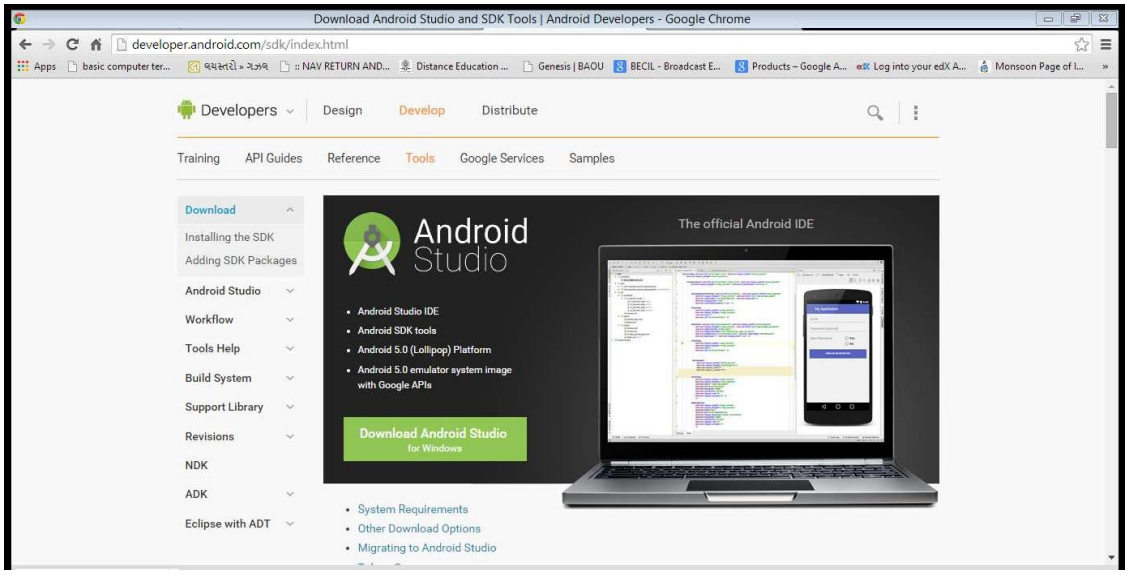



**Figure-2**

Click green button "Download Android Studio for Windows" it will open following page.

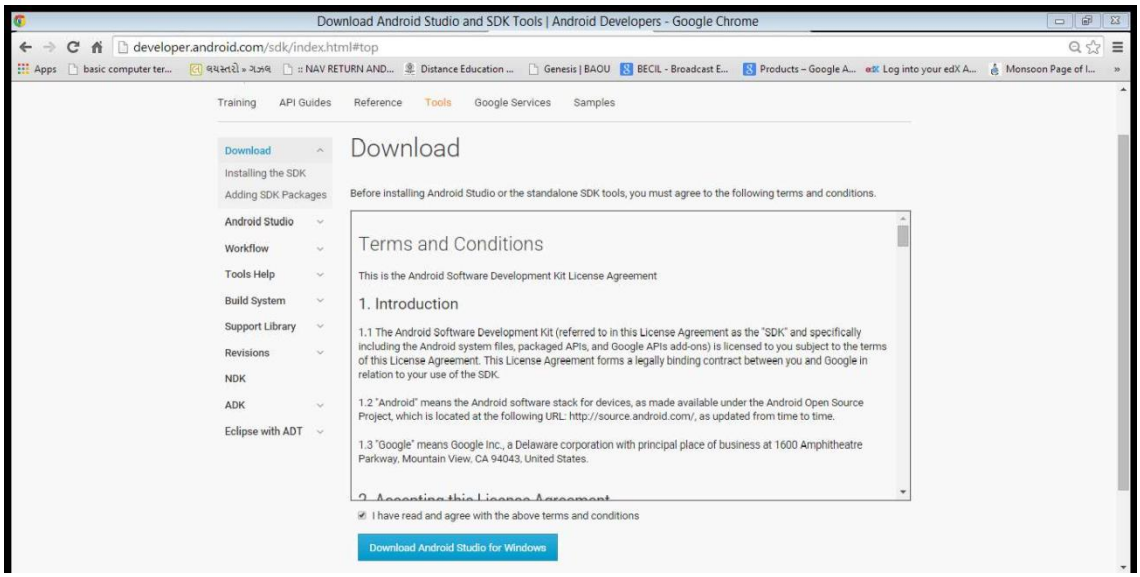



**Figure-3**

Accept term and condition at the bottom and press blue button "Download Android Studio for Windows". It will start downloading android studio for windows. Do not start installation before downloading and installing JDK 1.7 is over.

## Downloading JDK

To download JDK 1.7 type following URL in browser. http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html. It will open page shown below.

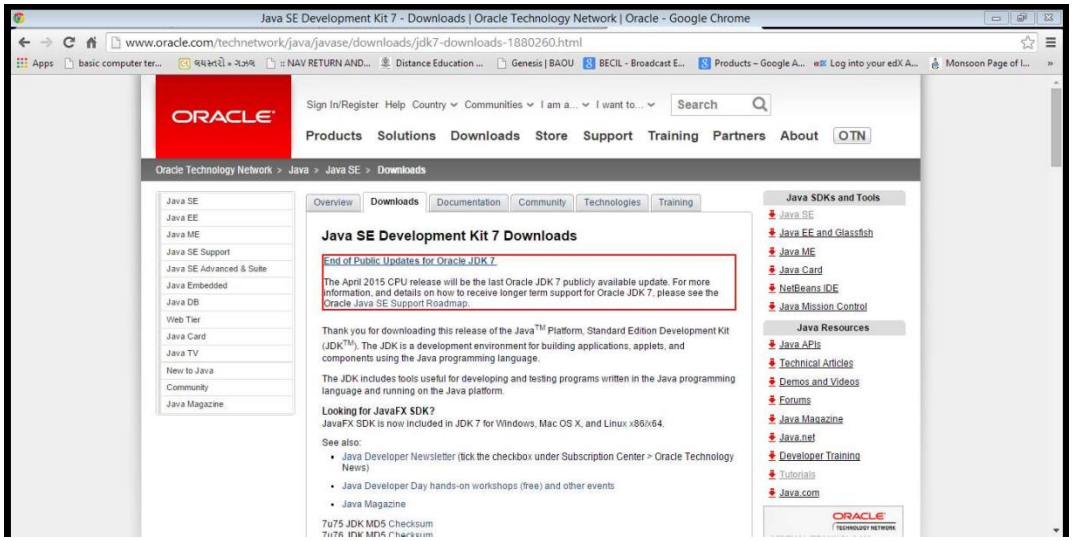



**Figure-4**

Go to bottom of the page and accept License agreement and download JDK for Windows x86 or Windows x64 for windows 32 bit and windows 64 bit respectively.

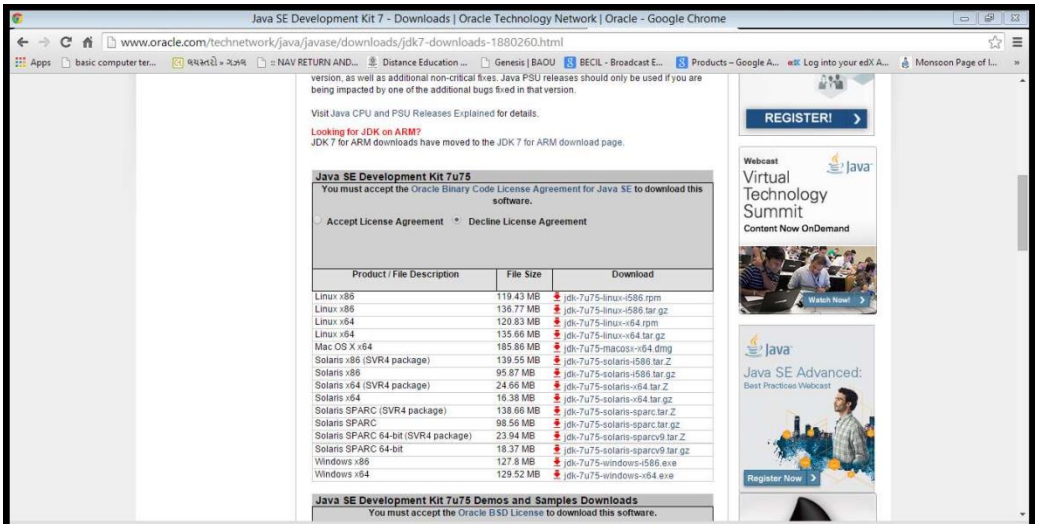



**Figure-5**

## Installing JDK

Double click downloaded JDK Installation file and follow instructions on screen.


**Figure-6**

Press Next Button

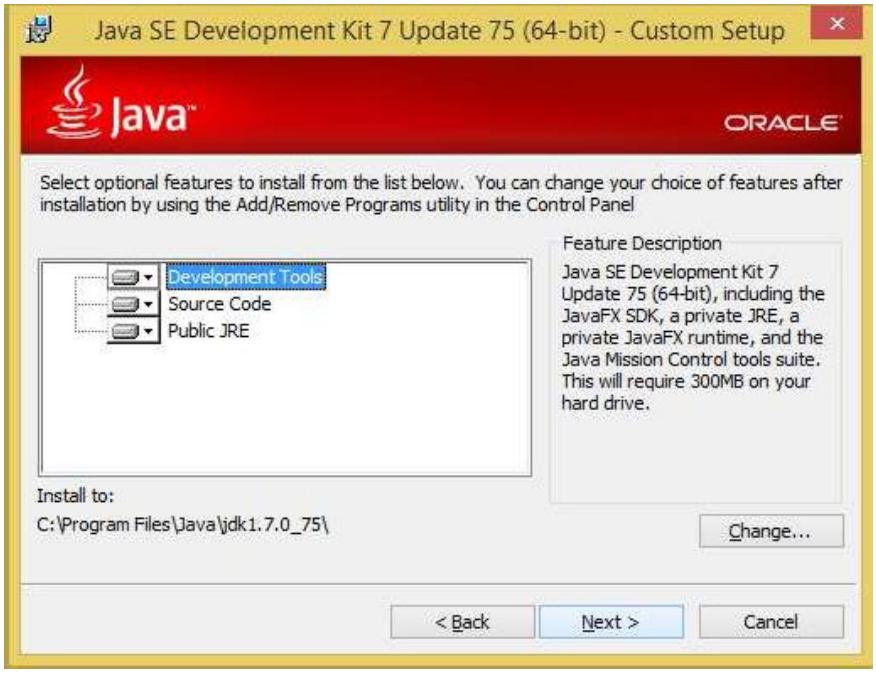



**Figure-7**

Press Next Button this will start installation as shown below.


**Figure-8**

After installation starts it will asks for Java Runtime Environment (JRE) Installation location.


**Figure-9**

Press Next Button it will start installing JRE as shown below.


**Figure-10**

Once Java SDK installation is finished it will display following screen.


**Figure-11**

Press close button to finish Java SDK installation.

To ensure that JDK is properly installed, open a terminal and type javac –version and press enter as shown below.

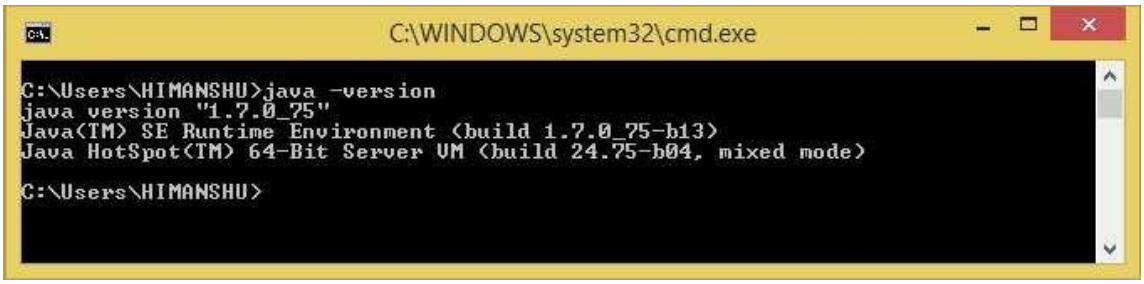



**Figure-12**

## Installing Android Studio

Launch the .exe file you just downloaded. It will open following screen.


**Figure-13**

Click on Next button.


**Figure-14**

Click Next button.


**Figure-15**

Click on I Agree button.

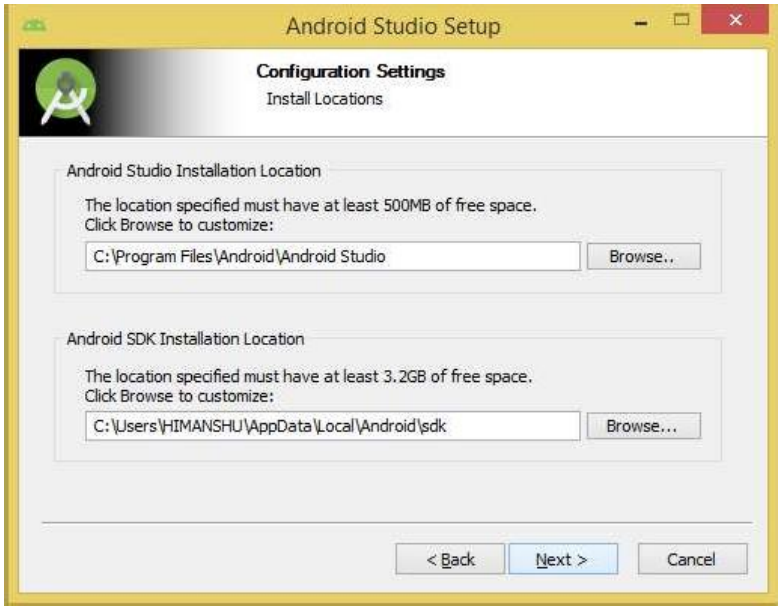



**Figure-16**

Specify path for Android Studio Installation and Android SDK Installation or use default and press Next button.

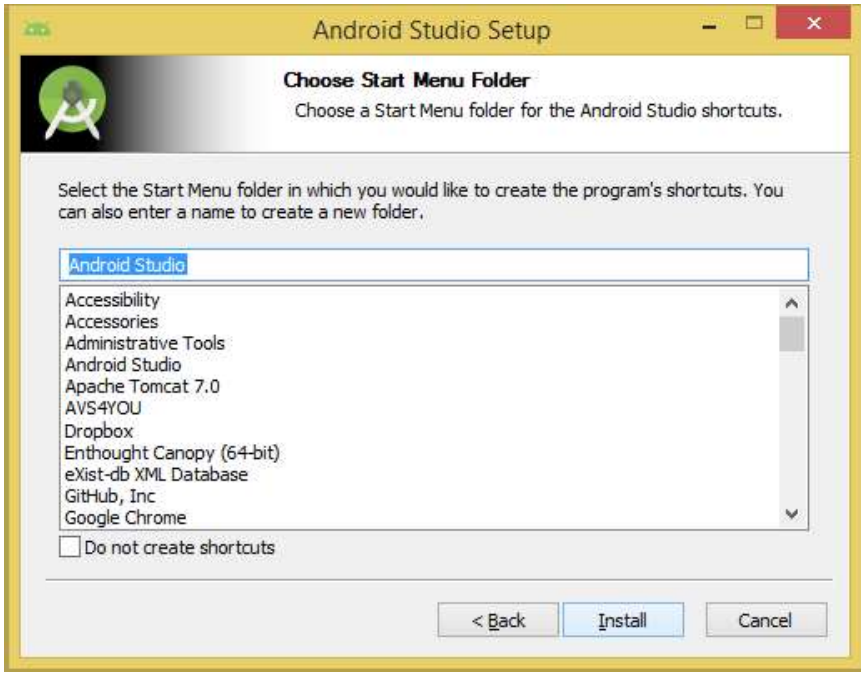



**Figure-17**

Press Install Button. This will start installation as shown below.

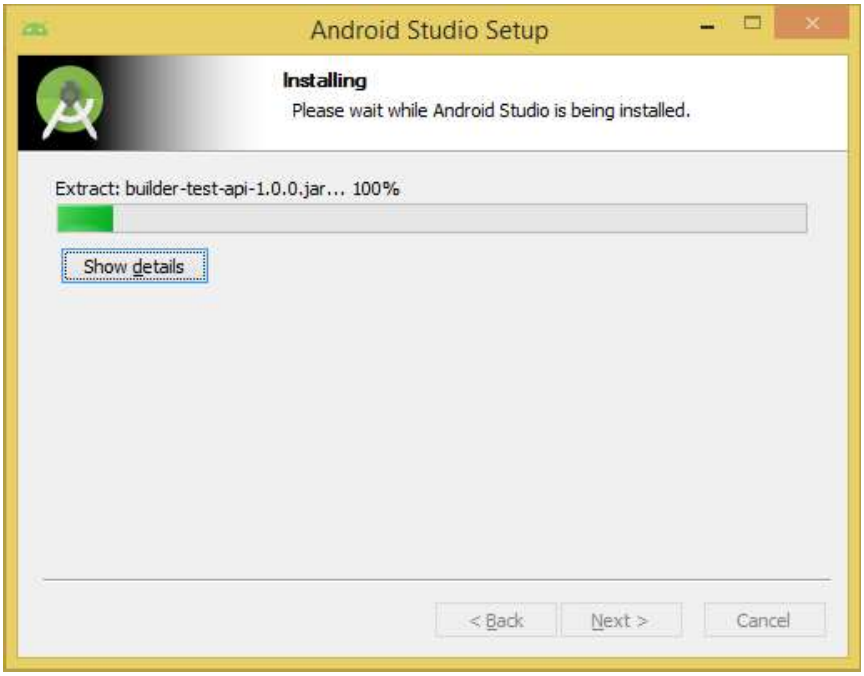



**Figure-18**

Once installation is finished completed message will be displayed as shown below.

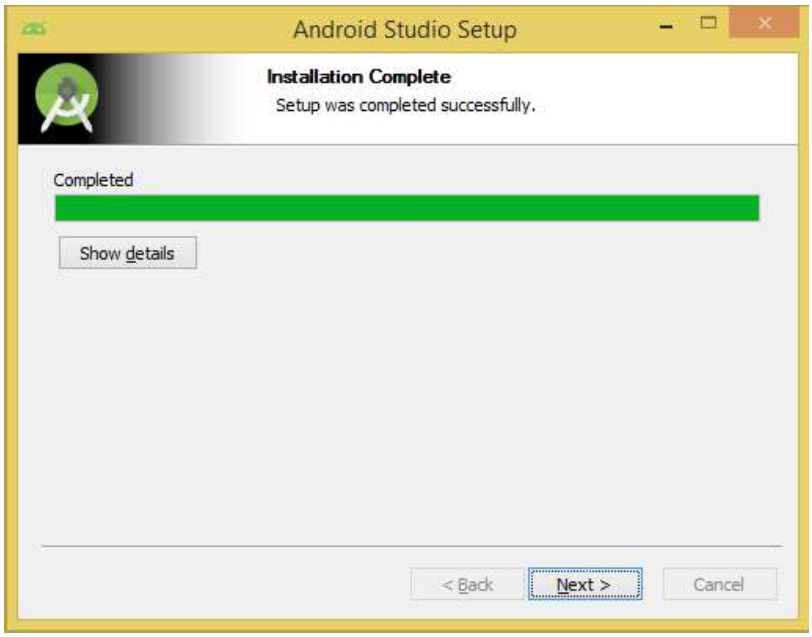



**Figure-19**

Press next button to finish installation and launch Android Studio.


**Figure-20**

**Important Note:** On some Windows systems, the launcher script does not find where Java is installed. If you encounter this problem, you need to set an environment variable indicating the correct location. Select Start menu > Computer > System Properties > Advanced System Properties. Then open Advanced tab > Environment Variables and add a new system variable JAVA_HOME that points to your JDK folder, for example C:\Program Files\Java\jdk1.7.0_XX.

## Launching Android Studio

Click on Finish button to launch Android Studio. It will display following screen


**Figure-21**

Select last option and press OK. It will download updates and create virtual device for you.

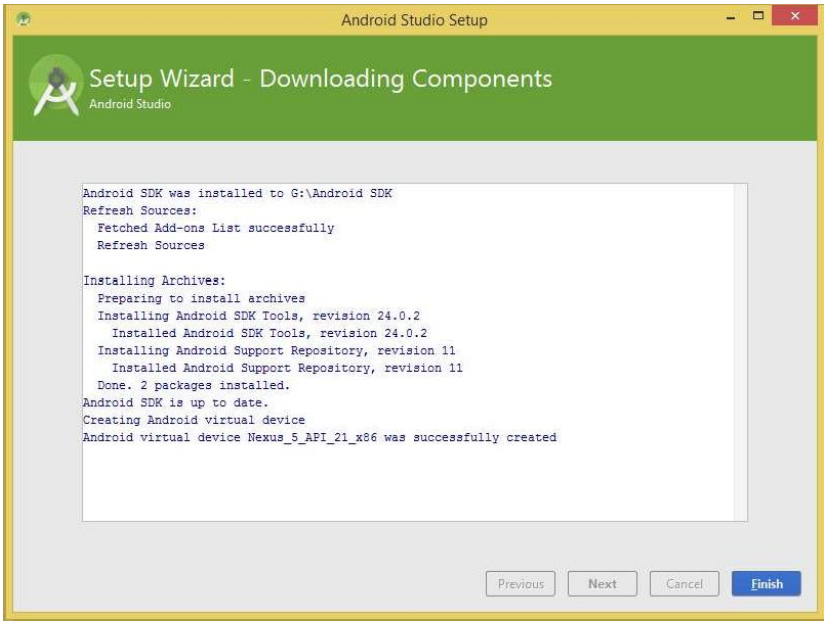



**Figure-22**

Press finish button to start Android Studio with following initial welcome screen.

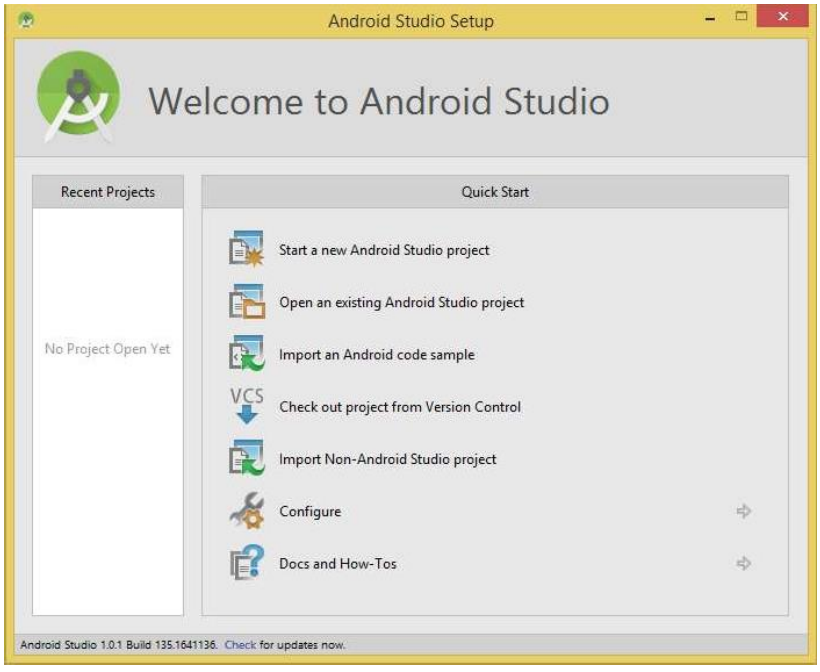



**Figure-23**

## Features of Android Studio

Android Studio has following salient features for android application development:

**Intelligent code editor:** Android Studio provides an intelligent code editor capable of advanced code completion, refactoring, and code analysis. The powerful code editor helps you be a more productive Android app developer.

**Project Wizard:** New project wizards make it easier than ever to start a new project. Start projects using template code for patterns such as navigation drawer and view pagers, and even import Google code samples from GitHub.

**Multi-screen app development:** Build apps for Android phones, tablets, Android Wear, Android TV, Android Auto and Google Glass. With the new Android Project

View and module support in Android Studio, it's easier to manage app projects and resources.

**Virtual devices for all shapes and sizes:** Android Studio comes pre-configured with an optimized emulator image. The updated and streamlined Virtual Device Manager provides pre-defined device profiles for common Android devices.

**Android builds evolved, with Gradle:** Create multiple APKs for your Android app with different features using the same project.

To develop apps for Android, you use a set of tools that are included in Android Studio. In addition to using the tools from Android Studio, you can also access most of the SDK tools from the command line.

## App Workflow

The basic steps for developing applications encompass four development phases, which include:

- **Environment Setup:** During this phase you install and set up your development environment. You also create Android Virtual Devices (AVDs) and connect hardware devices on which you can install your applications.
- **Project Setup and Development:** During this phase you set up and develop your Android Studio project and application modules, which contain all of the source code and resource files for your application.
- **Building, Debugging and Testing:** During this phase you build your project into a debuggable .apk package(s) that you can install and run on the emulator or an Android-powered device. Android Studio uses a build system based on Gradle that provides flexibility, customized build variants, dependency resolution, and much more. If you're using another IDE, you can build your project using Gradle and install it on a device using adb.

Next, with Android Studio you debug your application using the Android Debug Monitor and device log messages along with the IntelliJ IDEA intelligent coding features. You can also use a JDWP-compliant debugger along with the debugging and logging tools that are provided with the Android SDK.

Last, you test your application using various Android SDK testing tools.

- **Publishing:** During this phase you configure and build your application for release and distribute your application to users.

## Android Virtual Devices (AVD)

An Android Virtual Device (AVD) is an emulator configuration that lets you model an actual device by defining hardware and software options to be emulated by the Android Emulator. An AVD consists of:

**A hardware profile:** Defines the hardware features of the virtual device. For example, you can define whether the device has a camera, whether it uses a physical QWERTY keyboard or a dialing pad, how much memory it has, and so on.
**A mapping to a system image:** You can define what version of the Android platform will run on the virtual device. You can choose a version of the standard Android platform or the system image packaged with an SDK add-on.
**Other options:** You can specify the emulator skin you want to use with the AVD, which lets you control the screen dimensions, appearance, and so on. You can also specify the emulated SD card to use with the AVD.
**A dedicated storage area on your development machine:** the device's user data (installed applications, settings, and so on) and emulated SD card are stored in this area.

The easiest way to create an AVD is to use the graphical AVD Manager. You can also start the AVD Manager from the command line by calling the android tool with the avd options, from the <sdk>/tools/ directory.

You can also create AVDs on the command line by passing the android tool options.

You can create as many AVDs as you need, based on the types of device you want to model. To thoroughly test your application, you should create an AVD for each general device configuration (for example, different screen sizes and platform versions) with which your application is compatible and test your application on each one. Keep these points in mind when you are selecting a system image target for your AVD:

- The API Level of the target is important, because your application will not be able to run on a system image whose API Level is less than that required by your application, as specified in the minSdkVersion attribute of the application's manifest file.
- You should create at least one AVD that uses a target whose API Level is greater than that required by your application, because it allows you to test the forward-compatibility of your application. Forward-compatibility testing ensures that, when users who have downloaded your application receive a system update, your application will continue to function normally.
- If your application declares a uses-library element in its manifest file, the application can only run on a system image in which that external library is present. If you want to run your application on an emulator, create an AVD that includes the required library. Usually, you must create such an AVD using an Add-on component for the AVD's platform.

## Using Hardware Device to test Application

When building a mobile application, it's important that you always test your application on a real device before releasing it to users.

You can use any Android-powered device as an environment for running, debugging, and testing your applications. The tools included in the SDK make it easy to install and run your application on the device each time you compile. You can install your application on the device directly from Android Studio or from the command line with ADB.

## Android Studio IDE Components

Android Studio has just created an application project and opened the main window as shown below.

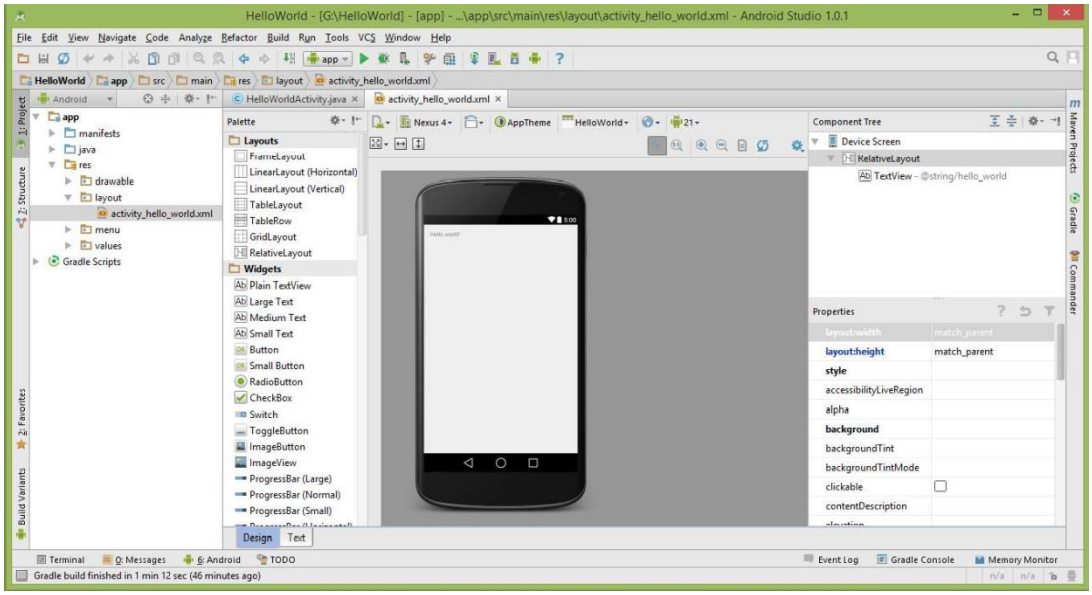



**Figure-24: Android studio IDE**

### Project Tool Window

The newly created project and references to associated files are listed in the Project tool window located on the left hand side of the main project window. The user interface design for our activity is stored in a file named activity_hello_world.xml which can be located using the Project tool window as shown below in Figure

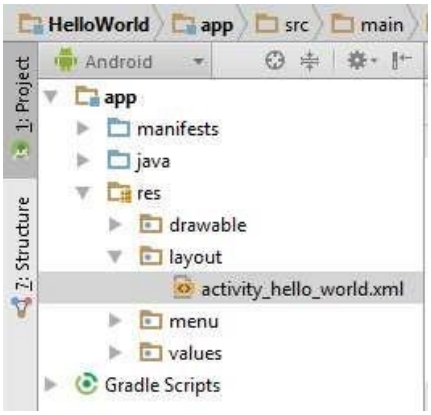



**Figure-25: Android studio IDE**

Double click on the file to load it into the User Interface Designer tool which will appear in the center panel of the Android Studio main window as shown below:

**Designer Window**

This is where you design your user interface. In the top of the Designer window is a menu set to Nexus 4 device which is shown in the Designer panel.

To change the orientation of the device representation between landscape and portrait simply use the drop down menu immediately to the right of the device selection menu showing the icon  .

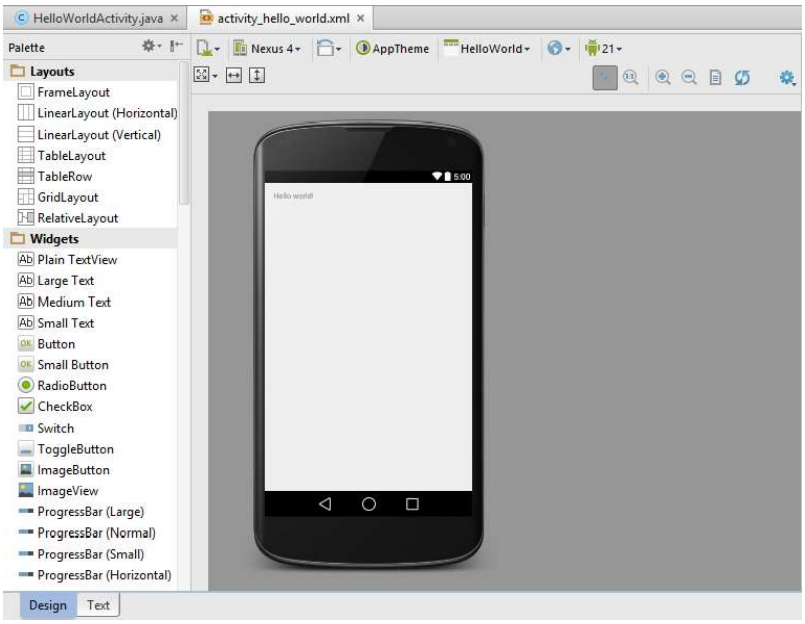



**Figure-26: Designer Window**

**Pellet**

On left hand side of the panel is a palette containing different categories of user interface components that may be used to construct a user interface, such as buttons, labels and text fields. Android supports a variety of different layouts that provide different levels of control over how visual user interface components are positioned and managed on the screen.

**Component Tree Panel**

Component Tree panel is by default located in the upper right hand corner of the Designer panel and is shown in Figure and shows layout used for user interface.

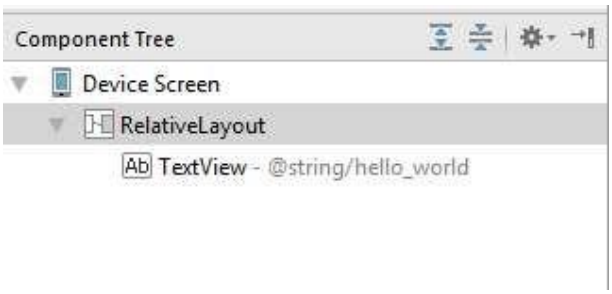



**Figure-27: Component Tree Panel**

**Property Window**

Property Window allows setting different properties of selected component.

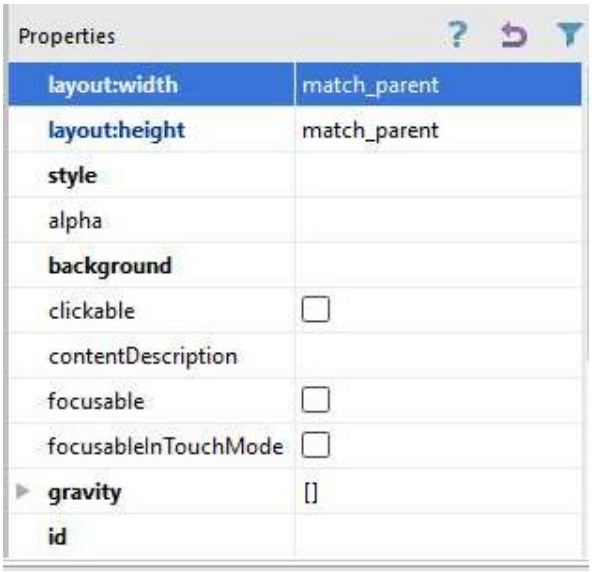



**Figure-28: Property Window**

### XML Editing Panel

We can modify user interface by modifying the activity_hello_world.xml using UI Designer tool but we can also modify design by editing XML file also. At the bottom of the Designer panel are two tabs labeled Design and Text respectively. To switch to the XML view simply select the Text tab as shown in Figure. At the right hand side of the XML editing panel is the Preview panel and shows the current visual state of the layout.

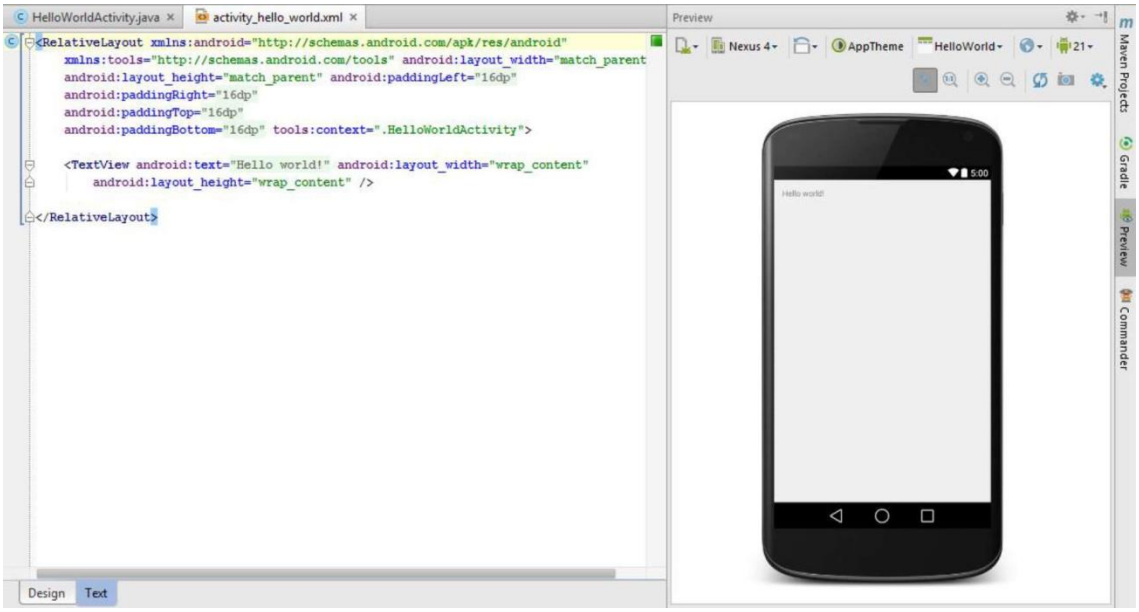



**Figure-29: XML Editing Panel**

### Previewing the Layout

In above figure layout has been previewed of the Nexus 4 device. The layout can be tested for other devices by making selections from the device menu in the toolbar across the top edge of the Designer panel. We can also preview screen size for all currently configured device as shown in figure.

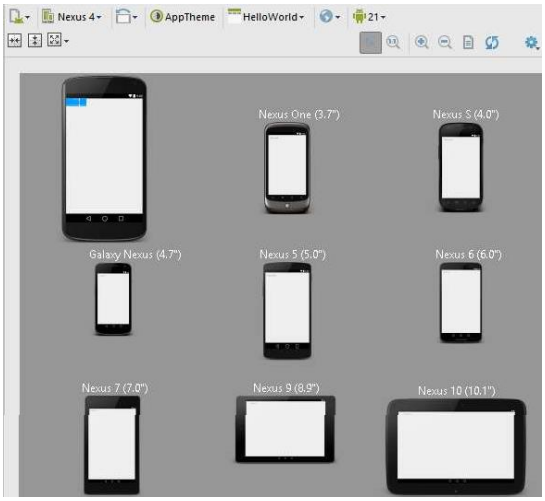



**Figure-30: Previewing the Layout**

## Android Studio Code Editor Customization

We customize code editor for font, displaying quick help when mouse moves over code.

**Font Customization:**

From File menu select settings option following dialog will open.

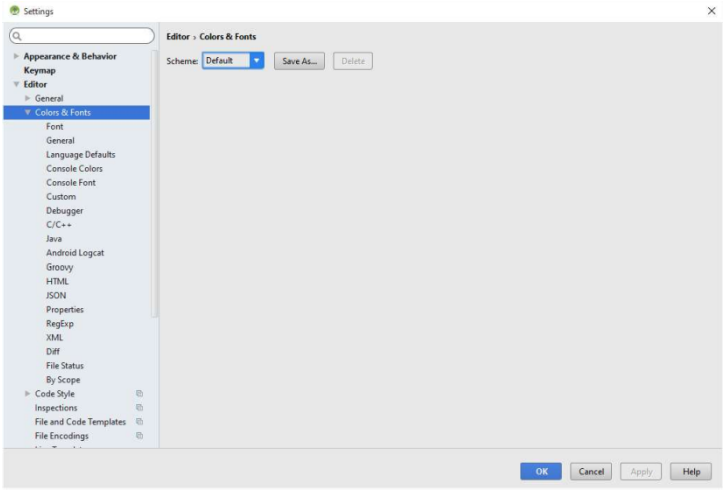



**Figure-31**

Select Colors & Fonts option. In scheme Default scheme is displayed. Click **"Save As__"** Button and give new name as **"My Settings"** to scheme. Now select font option as shown below and customize font as per your requirement.

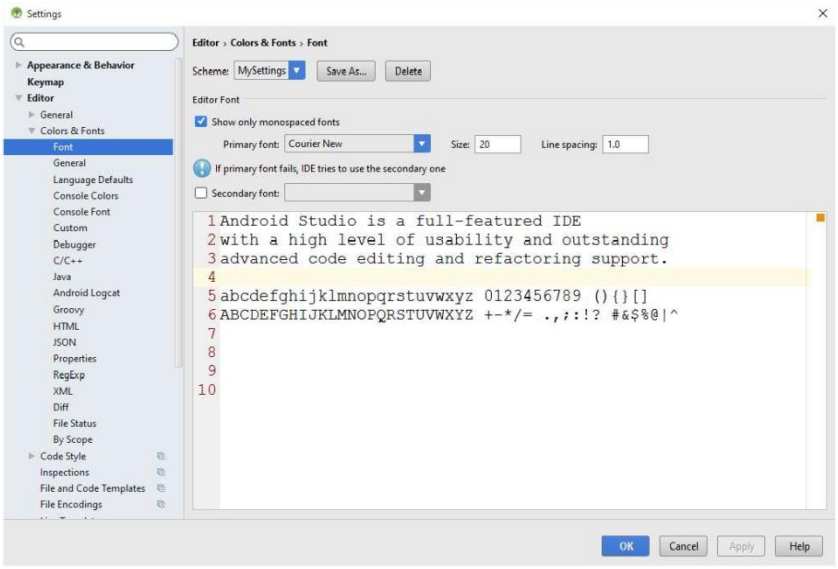



**Figure-32**

**Show quick doc on mouse move**

In android studio code editor, when you move your mouse over a method, class or interface, a documentation window would appear with a description of that programming element. This feature is by default disabled in android studio. This feature can be enabled in android studio as follows.

From file menu select settings option following dialog box appears. Select General option from list and scroll down to checkbox highlighted with red rectangle in below figure. Then press OK.

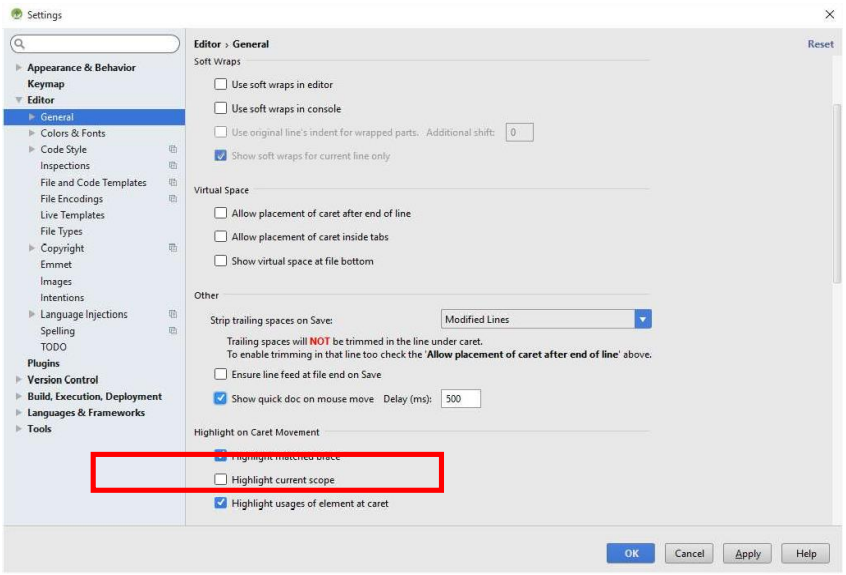



**Figure-33**

## Coding Best Practices

Following are some of the code editing practices you should follow when creating Android Studio apps.

| Code Practice | Description |
|---|---|
| **Alt + Enter key** | For quick fixes to coding errors such as missing imports, variable assignments, missing references, etc. You can use Alt + Enter key to fix errors for the most probable solution. |
| **Ctrl + D** | The Ctrl + D key are used for quickly duplicating code lines or fragments. Instead of copy and paste, you can select the desired line or fragment and enter this key. |
| **Navigate menu** | Use the Navigate menu to jump directly to the class of a method or field name without having to search through individual classes. |
| **Code folding** | This allows you to selectively hide and display sections of the code for readability. For example, resource expressions or code for a nested class can be folded or hidden in to one line to make the outer class structure easier to read. The inner class can be later expanded for updates. |
| **Image and color preview** | When referencing images and icons in your code, a preview of the image or icon appears (in actual size at different densities) in the code margin to help you verify the image or icon reference. Pressing F1 with the preview image or icon selected displays resource asset details, such as the dp settings. |
| **Quick documentation** | You can inspect theme attributes using View > Quick Documentation (Ctrl+Q), If you invoke View > Quick Documentation on the theme attribute ?android:textAppearanceLarge, you will see the theme inheritance hierarchy and resolved values for the various attributes that are pulled in. |

**Table-2**

The following tables list keyboard shortcuts for common operations.

| Action | Android Studio Key Command |
|---|---|
| **Command look-up (autocomplete command name)** | CTRL + SHIFT + A |
| **Project quick fix** | ALT + ENTER |
| **Reformat code** | CTRL + ALT + L (Win)<br>OPTION + CMD + L (Mac) |
| **Show docs for selected API** | CTRL + Q (Win)<br>F1 (Mac) |
| **Show parameters for selected method** | CTRL + P |
| **Generate method** | ALT + Insert (Win)<br>CMD + N (Mac) |
| **Jump to source** | F4 (Win)<br>CMD + down-arrow (Mac) |
| **Delete line** | CTRL + Y (Win)<br>CMD + Backspace (Mac) |
| **Search by symbol name** | CTRL + ALT + SHIFT + N (Win)<br>OPTION + CMD + O (Mac) |

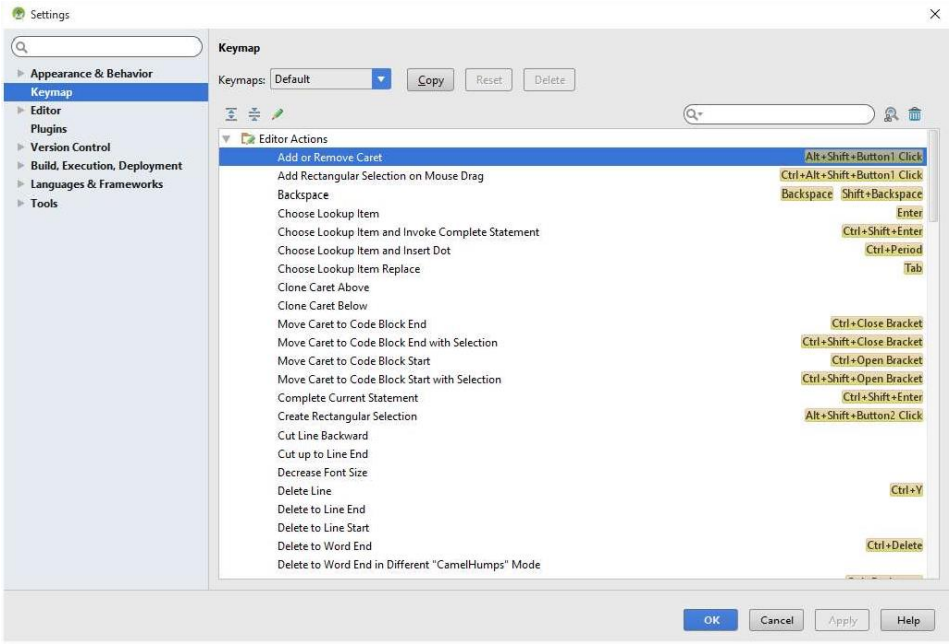
**Table-3**

Following table lists project and editor key commands:

| Action | Android Studio Key Command |
|---|---|
| **Build** | CTRL + F9 (Win), CMD + F9 (Mac) |
| **Build and run** | SHIFT + F10 (Win), CTRL + R (Mac) |
| **Toggle project visibility** | ALT + 1 (Win), CMD + 1 (Mac) |
| **Navigate open tabs** | ALT + left-arrow; ALT + right-arrow (Win)<br>CTRL + left-arrow; CTRL + right-arrow (Mac) |

**Table-4**

You can change these shortcuts from file menu settings option as shown below.


Settings
Appearance & Behavior
Keymap
Editor
Plugins
Version Control
Build, Execution, Deployment
Languages & Frameworks
Tools
Keymap
Keymaps: Default
Copy
Reset
Delete
Editor Actions
Add or Remove Caret
Add Rectangular Selection on Mouse Drag
Backspace
Choose Lookup Item
Choose Lookup Item and Invoke Complete Statement
Choose Lookup Item and Insert Dot
Choose Lookup Item Replace
Clone Caret Above
Clone Caret Below
Move Caret to Code Block End
Move Caret to Code Block End with Selection
Move Caret to Code Block Start
Move Caret to Code Block Start with Selection
Complete Current Statement
Create Rectangular Selection
Cut Line Backward
Cut up to Line End
Decrease Font Size
Delete Line
Delete to Line End
Delete to Line Start
Delete to Word End
Delete to Word End in Different "CamelHumps" Mode
Alt+Shift+Button1 Click
Ctrl+Alt+Shift+Button1 Click
Backspace Shift+Backspace
Enter
Ctrl+Shift+Enter
Ctrl+Period
Tab
Ctrl+Close Bracket
Ctrl+Shift+Close Bracket
Ctrl+Open Bracket
Ctrl+Shift+Open Bracket
Ctrl+Shift+Enter
Alt+Shift+Button2 Click
Ctrl+Y
Ctrl+Delete
OK
Cancel
Apply
Help

## Introduction

With all the tools and the SDK downloaded and installed, it is now time to start your engine! As in all programming books, the first example uses the ubiquitous Hello World application. This will enable you to have a detailed look at the various components that make up an Android project.

So, without any further ado, let's dive straight in! Generally a program is defined in terms of functionality and data, and an Android application is not an exception. It performs processing, show information on the screen, and takes data from a variety of sources.

To Develop Android applications for mobile devices with resource constraint requires a systematic understanding of the application lifecycle. This unit introduces you with the most important components of Android applications and provides you with a more detailed understanding of how to create and run an Android application.

## Building a sample Android application using Android Studio

Before developing sophisticated Android application, it is necessary to check whether all of the required development packages are installed and functioning correctly. The simple way to realize this aim is to create an Android application and compile and run it. This topic will explain how to create a simple Android application project using Android Studio. Once the project has been created, a later chapter will

explore the use of the Android emulator environment to perform a test run of the application.

## Creating a New Android Project

The first step in the application development process is to create a new project within the Android Studio environment. Launch Android Studio so that the "Welcome to Android Studio" screen appears as shown Figure:


**Figure-35**

To create the new project, simply click on the Start a new Android Studio project option to display the first screen of the New Project wizard as shown in Figure:

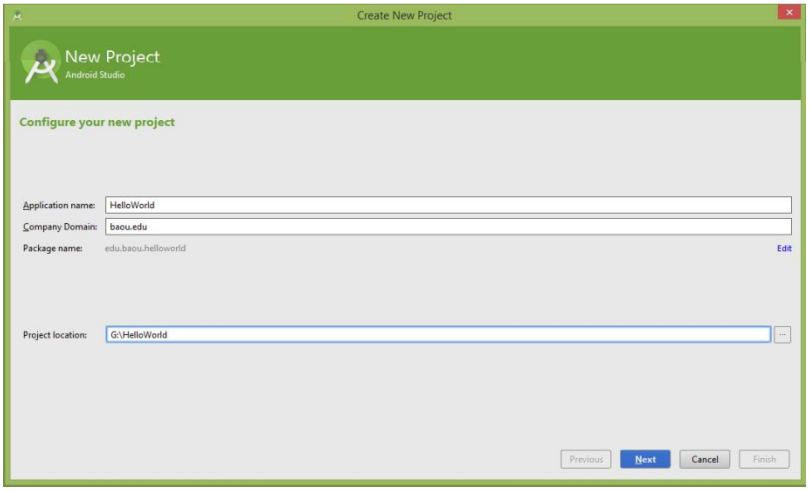



**Figure-36**

In the New Project window, set the Application name field to HelloWorld. The application name is the name by which the application will be referenced and

identified within Android Studio and is also the name that will be used when the completed application goes on sale in the Google Play store.

The Package Name is used to uniquely identify the application within the Android application ecosystem. It should be based on the reversed URL of your domain name followed by the name of the application. For example, if your domain is www.baou.edu, and the application has been named HelloWorld, then the package name might be specified as follows:
edu.baou.HelloWorld

The Project location setting will default to a location in the folder named AndroidStudioProjects located in your home directory and may be changed by clicking on the button to the right of the text field containing the current path setting.

Click Next to proceed.

## Defining the Project and SDK Settings

On the form factors screen, enable the Phone and Tablet option and set the minimum SDK setting to API 8: Android 2.2 (Froyo). The reason for selecting an older SDK release is that this ensures that the finished application will be able to run on the widest possible range of Android devices. The higher the minimum SDK selection, the more the application will be restricted to newer Android devices.

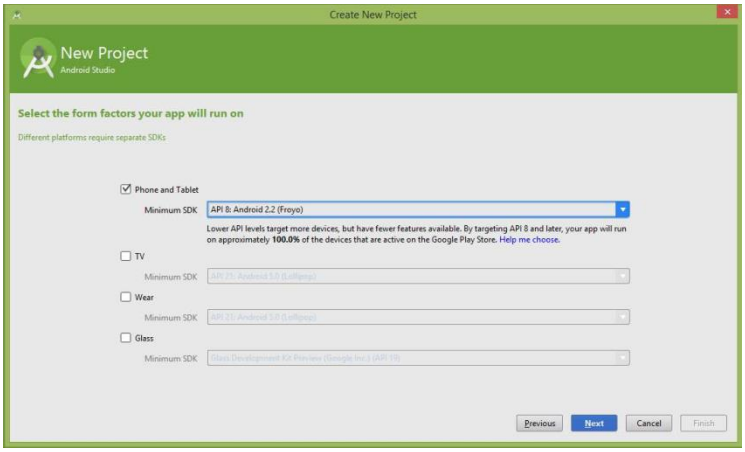



**Figure-37**

Click Next to proceed.

## Creating Activity

The next step is to define the type of initial activity that is to be created for the application. A range of different activity types is available when developing Android applications. For sack of simplicity we select the option to create a Blank Activity and Click Next to proceed.

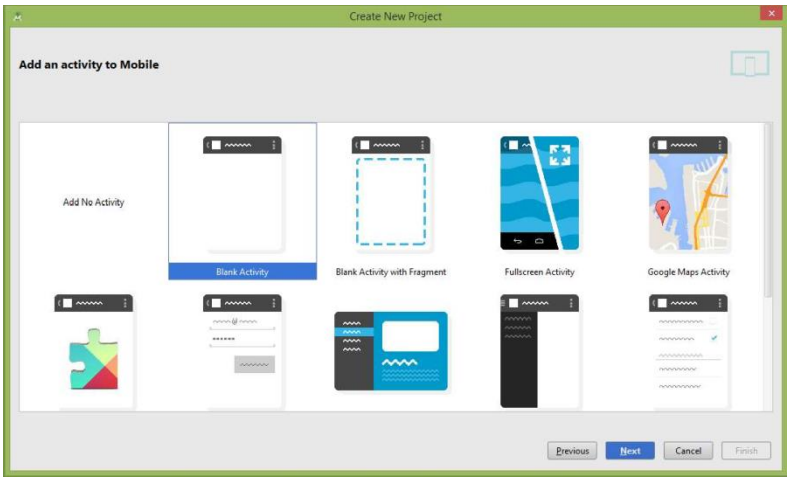



**Figure-38**

On the final screen name the activity and title HelloWorldActivity. The activity will consist of a single user interface screen layout which, for the purposes of this example, should be named activity_hello_world as shown in Figure and with a menu resource named menu_hello_world:

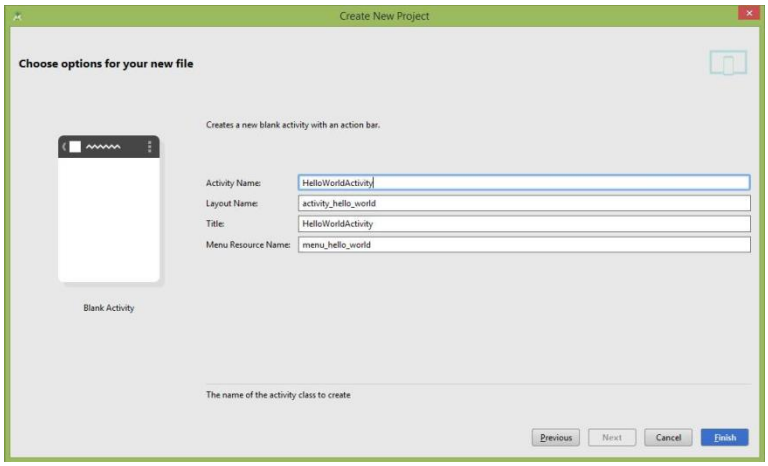



**Figure-39**

Finally, click on Finish to initiate the project creation process.

## Running a HelloWorld Application

1. Press Shift+F10 or 'Run App' button in taskbar. It will launch following dialog box.

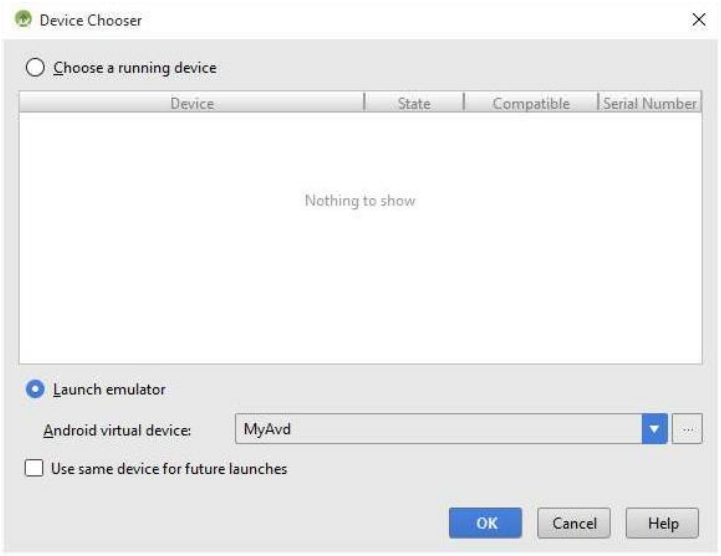



**Figure-40**

2. Select Launch emulator option and select your Android virtual device and Press OK.
3. The Android emulator starts up, which might take a moment
4. Press the Menu button to unlock the emulator.
5. The application starts, as shown in Figure below.

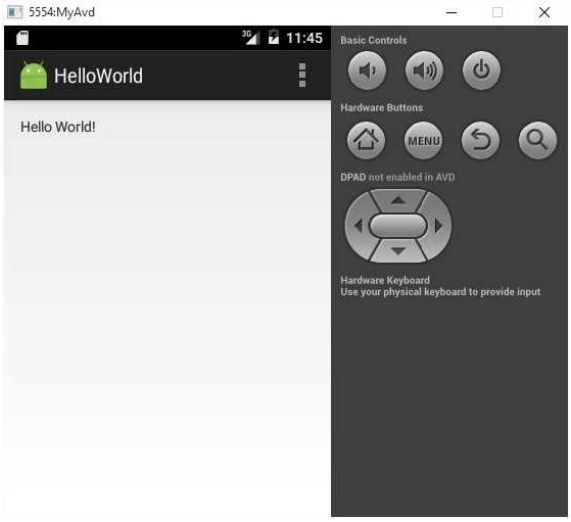



**Figure-41**

6. Click the Home button in the Emulator to end the application.

7. Pull up the Application Drawer to see installed applications. Your screen looks something
8. like Figure shown below

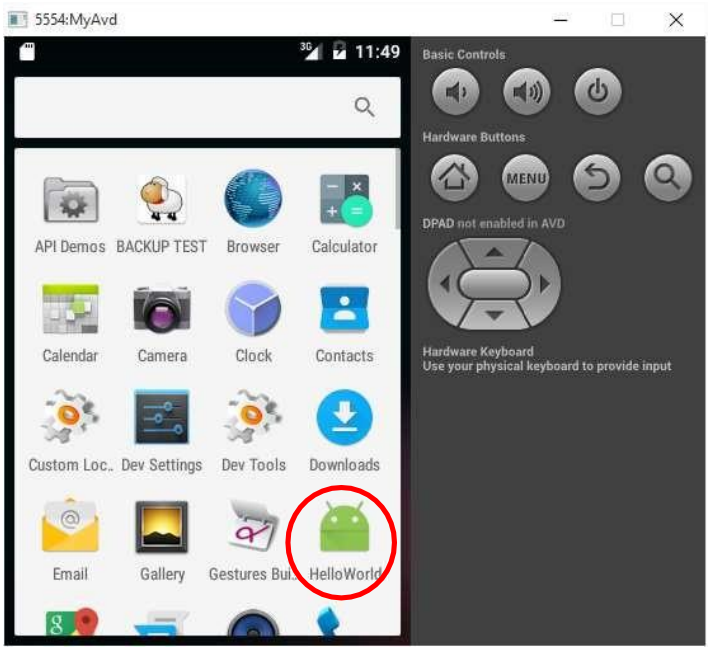



**Figure-42**

Recall that earlier you created a few AVDs using the AVD Manager. So which one will be launched by Android Studio when you run an Android application? Android studio will check the target that you specified (when you created a new project), comparing it against the list of AVDs that you have created. The first one that matches will be launched to run your application.

If you have more than one suitable AVD running prior to debugging the application, Android Studio will display the Android Device Chooser window, which enables you to select the desired emulator/device to debug the application.

## Introduction

The Android build system is organized around a specific directory tree structure for the Android project, similar to the any Java project. The project prepares the actual application that will run on the device or emulator. When you create a new Android project, you get several items in the project's root directory which is discussed in sub sequent sections.

When you create an Android project as discussed in previous unit, you provide the fully-qualified class name of the "main" activity for the application (e.g., edu.baou.HelloWorld).

You will then find that your project's src/ tree already has the namespace directory tree in place, plus a stub Activity subclass representing your main activity (e.g., src/edu/baou/HelloWorld.java). You can modify this file and add others to the src/ tree as per requirement implement your application.

When you compile the project for first time, in the "main" activity's namespace directory, the Android build chain will create R.java. This contains a number of constants tied to the various resources you placed out in the res/ directory tree. You should not modify R.java yourself, letting the Android tools handle it for you. You will see throughout many of the samples where we reference things in R.java (e.g., referring to a layout's identifier via R.layout.main).

## Android Project Structure

An Android project contains everything that defines your Android app. The SDK tools require that your projects follow a specific structure so it can compile and package your application correctly. Android Studio takes care of all this for you.

A module is the first level of control within a project that encapsulates specific types of source code files and resources. There are several types of modules with a project:

| Module | Description |
|---|---|
| **Android Application Modules** | It contain source code, resource files, and application level settings, such as the module-level build file, resource files, and Android Manifest file. |
| **Test Modules** | It contains code to test your application projects and is built into test applications that run on a device. |
| **Library Modules** | It contains shareable Android source code and resources that you can reference in Android projects. This is useful when you have common code that you want to reuse. |
| **App Engine Modules** | They are App Engine java Servlet Module for backend development, App Engine java Endpoints Module to convert server-side Java code annotations into RESTful backend APIs, and App Engine Backend with Google Cloud Messaging to send push notifications from your server to your Android devices. |

**Table-4**

When you use the Android development tools to create a new project and the module, the essential files and folders will be created for you. As your application grows in complexity, you might require new kinds of resources, directories, and files.

## Android Project Files

Android Studio project files and settings provide project-wide settings that apply across all modules in the project.

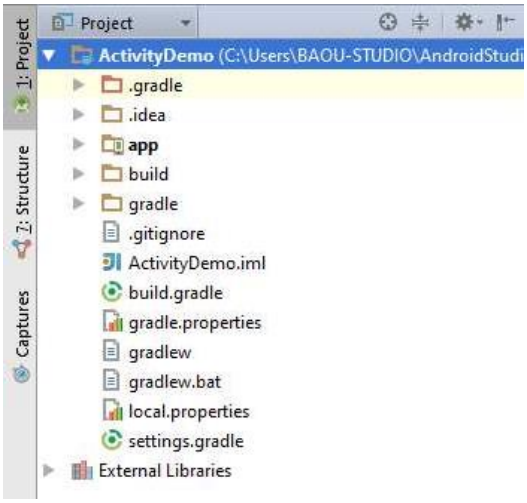



**Figure-43**

| File | Meaning |
|---|---|
| **.idea** | Directory for IntelliJ IDEA settings. |
| **App** | Application module directories and files. |
| **Build** | This directory stores the build output for all project modules. |
| **Gradle** | Contains the gradler-wrapper files. |
| **.gitignore** | Specifies the untracked files that Git should ignore. |
| **build.gradle** | Customizable properties for the build system. |
| **gradle.properties** | Project-wide Gradle settings. |
| **gradlew** | Gradle startup script for Unix. |
| **gradlew.bat** | Gradle startup script for Windows. |
| **local.properties** | Customizable computer-specific properties for the build system, such as the path to the SDK installation. |
| **.iml** | Module file created by the IntelliJ IDEA to store module information. |
| **settings.gradle** | Specifies the sub-projects to build. |

**Table-5**

## Android Application Modules

Android Application Modules contain things such as application source code and resource files. Most code and resource files are generated for you by default, while others should be created if required. The following directories and files comprise an Android application module:

When you le the for first time, in the “main” activity’s namespace

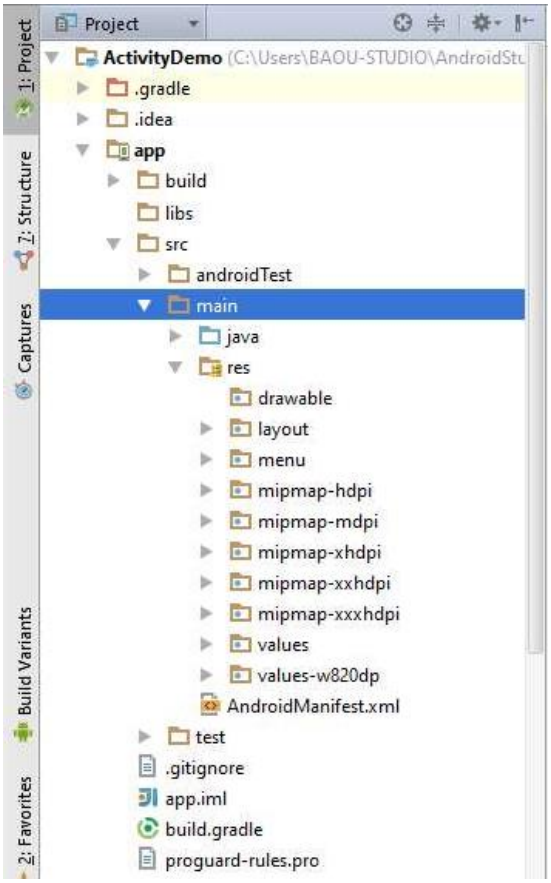



**Figure-44**

| File | Meaning |
|---|---|
| **build/** | Contains build folders for the specified build variants. Stored in the |

| File | Meaning |
|---|---|
| | main application module. |
| **libs/** | Contains private libraries. Stored in the main application module. |
| **src/** | Contains your stub Activity file, which is stored at src/main/java/<ActivityName>.java. All other source code files (such as .java or .aidl files) go here as well. |
| **androidTest/** | Contains the instrumentation tests. |
| **main/jni/** | Contains native code using the Java Native Interface (JNI). |
| **main/gen/** | Contains the Java files generated by Android Studio, such as your R.java file and interfaces created from AIDL files. |
| **main/assets/** | This is empty. You can use it to store raw asset files. For example, this is a good location for textures and game data. Files that you save here are compiled into an .apk file as-is, and the original filename is preserved. You can navigate this directory and read files as a stream of bytes using the AssetManager. |
| **main/res/** | Contains application resources, such as drawable files, layout files, and string values in the following directories. |
| **anim/** | For XML files that are compiled into animation objects. |
| **color/** | For XML files that describe colors. |
| **drawable/** | For bitmap files (PNG, JPEG, or GIF), 9-Patch image files, and XML files that describe Drawable shapes or Drawable objects that contain multiple states (normal, pressed, or focused). |
| **mipmap/** | For app launcher icons. The Android system retains the resources in this folder (and density-specific folders such as mipmap-xxxhdpi) regardless of the screen resolution of the device where your app is installed. This behavior allows launcher apps to pick the best resolution icon for your app to display on the home screen. |
| **layout/** | XML files that are compiled into screen layouts (or part of a screen). |
| **menu/** | For XML files that define application menus. |
| **raw/** | For arbitrary raw asset files. Saving asset files here is essentially |

| File | Meaning |
|---|---|
| | the same as saving them in the assets/ directory. The only difference is how you access them. These files are processed by aapt and must be referenced from the application using a resource identifier in the R class. For example, this is a good place for media, such as MP3 or Ogg files. |
| **values/** | For XML files that define resources by XML element type. Unlike other resources in the res/ directory, resources written to XML files in this folder are not referenced by the file name. Instead, the XML element type controls how the resources defined within the XML files are placed into the R class. |
| **xml/** | For miscellaneous XML files that configure application components. For example, an XML file that defines a PreferenceScreen, AppWidgetProviderInfo, or Searchability Metadata. |
| **AndroidManifest.xml** | The control file that describes the nature of the application and each of its components. For instance, it describes: certain qualities about the activities, services, intent receivers, and content providers; what permissions are requested; what external libraries are needed; what device features are required, what API Levels are supported or required; and others. |
| **.gitignore/** | Specifies the untracked files ignored by git. |
| **app.iml/** | IntelliJ IDEA module |
| **build.gradle** | Customizable properties for the build system. You can edit this file to override default build settings used by the manifest file and also set the location of your keystore and key alias so that the build tools can sign your application when building in release mode. This file is integral to the project, so maintain it in a source revision control system. |
| **proguard-rules.pro** | ProGuard settings file. |

**Table-6**

## Anatomy of an Android Application

Generally a program is defined in terms of functionality and data, and an Android application is not an exception. It performs processing, show information on the screen, and takes data from a variety of sources.

To Develop Android applications for mobile devices with resource constraint requires a systematic understanding of the application lifecycle. Important terminology for application building blocks terms are Context, Activity, and Intent. This section introduces you with the most important components of Android applications and

provides you with a more detailed understanding of how Android applications function and interact with one another.

## Important Android Terminology

The followings are the important terminology used in Android application development.

- **Context:** The context is the essential command for an Android application. It stores the current state of the application/object and all application related functionality can be accessed through the context. Typically you call it to get information regarding another part of your program such as an activity, package, and application.

- **Activity:** It is core to any Android application. An Android application is a collection of tasks, each of which is called an Activity. Each Activity within an application has an exclusive task or purpose. Typically, applications have one or more activities, and the main objective of an activity is to interact with the user.

- **Intent:** Intent is a messaging object which can be used to request an action from another app component. Each request is packaged as Intent. You can think of each such request as a message stating intent to do something. Intent mainly used for three tasks 1) to start an activity, 2) to start a service and 3) to deliver a broadcast.

- **Service:** Tasks that do not require user interaction can be encapsulated in a service. A service is most useful when the operations are lengthy (offloading time-consuming processing) or need to be done regularly (such as checking a server for new mail).

## Basic Android API Packages

Application program interface (API) is a set of routines, protocols, and tools for building software applications. An API specifies how software components should interact and APIs are used when programming graphical user interface (GUI) components. Android offers a number of APIs for developing your applications. The

following list of core APIs should provide an insight into what's available; all Android devices will offer support for at least these APIs:

| API Package | Use |
|---|---|
| **android.util** | Provides common utility methods such as date/time manipulation, base64 encoders and decoders, string and number conversion methods, and XML utilities. |
| **android.os** | Provides basic operating system services, message passing, and inter-process communication on the device. |
| **android.graphics** | Provides low level graphics tools such as canvases, color filters, points, and rectangles that let you handle drawing to the screen directly. |
| **android.text** | Provides classes used to render or track text and text spans on the screen. |
| **android.database** | Contains classes to explore data returned through a content provider. |
| **android.content** | Contains classes for accessing and publishing data on a device. |
| **android.view** | Provides classes that expose basic user interface classes that handle screen layout and interaction with the user. |
| **android.widget** | The widget package contains (mostly visual) UI elements to use on your Application screen. |
| **android.app** | Contains high-level classes encapsulating the overall Android application model. |
| **android.provider** | Provides convenience classes to access the content providers supplied by Android. |
| **android.webkit** | Provides tools for browsing the web. |

**Table-7**

## Advanced Android API Packages

The core libraries provide all the functionality you need to start creating applications for Android, but it won't be long before you're ready to delve into the advanced APIs that offer the really exciting functionality.

Android hopes to target a wide range of mobile hardware, so be aware that the suitability and implementation of the following APIs will vary depending on the device upon which they are implemented.

| API Package | Use |
|---|---|
| **android.location** | Contains the framework API classes that define Android location-based and related services. |
| **android.media** | Provides classes that manage various media interfaces in audio and video. |
| **android.opengl** | Provides an OpenGL ES static interface and utilities. |
| **android.hardware** | Provides support for hardware features, such as the camera and other sensors. |
| **android.bluetooth** | Provides classes that manage Bluetooth functionality, such as scanning for devices, connecting with devices, and managing data transfer between devices. The Bluetooth API supports both "Classic Bluetooth" and Bluetooth Low Energy. |
| **android.net.wifi** | Provides classes to manage Wi-Fi functionality on the device. |
| **android.telephony** | Provides APIs for monitoring the basic phone information, such as the network type and connection state, plus utilities for manipulating phone number strings. |

**Table-8**

## Introduction

Each and every android app project must have an AndroidManifest.xml file in the root of your project. The manifest file describes important information about your app. The manifest file declares the following:

- The app's package name
- The components of the app such as activities, services, broadcast receivers, and content providers.
- Which device configurations it can handle
- Intent filters that describe how the component can be started.
- Permissions required by the app
- The hardware and software features the app requires

Android Studio generally builds the manifest file for you when you create a project. For a simple application with a single activity and nothing else, the auto-generated manifest will work fine with little or no modifications.

## Component of Manifest file

Android manifest file is global application description file which defines your application's capabilities and permissions and how it runs. This topic describes some of the most important characteristics of your app which is stored in the manifest file.

## Package name and application ID

The manifest file's root element requires an attribute for your app's package name, For example, the following snippet shows the root <manifest> element with the package name "in.edu.baou.databasedemo":

```
<? xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="in.edu.baou.databasedemo"
   android:versionCode="1"
   android:versionName="1.0"  >
   ...
</manifest>
```

While building your app into the final APK, the Android build tools use the package attribute for two things:

It applies this name as the namespace for your app's generated R.java class. With the above manifest, the R class is created at in.edu.baou.databasedemo.R.

Android manifest file uses package this name to resolve any relative class names that are declared in the manifest file.

If, an activity declared as <activity android:name=".MainActivity"> is resolved to be in.edu.baou.databasedemo.MainActivity.

You should keep in mind that once the APK is compiled, the package attribute also represents your app's universally unique application ID. After the build tools perform the above tasks based on the package name, they replace the package value with the value given to the applicationId property in your project's build.gradle file.

## App Components

For each app component that you create in your app, you must declare a corresponding XML element in the manifest file so that the system can start it.

For each subclass of Activity, we have <activity>
For each subclass of Service, we have <service>
For each subclass of BroadcastReceiver we have <receiver>.
For each subclass of ContentProvider, we have <provider>

The name of your subclass must be specified with the name attribute, using the full package designation, e.g. an Activity subclass can be declared as follows

```
<manifest package=" in.edu.baou.databasedemo" ... >
   <application ... >
      <activity android:name=".SQLiteDBActivity " ... >
         ...
      </activity>
   </application>
</manifest>
```

In above example, the activity name is resolved to
"in.edu.baou.databasedemo.SQLiteDBActivity "

App activities, services, and broadcast receivers are activated by intents. It is an asynchronous messaging mechanism to match task requests with the appropriate Activity.

When an app issues intent to the system, the system locates an app component that can handle the intent based on intent filter declarations in each app's manifest file. The system launches an instance of the matching component and passes the Intent object to that component. If more than one app can handle the intent, then the user can select which app to use. An intent filters is defined with the <intent-filter> element as shown below.

```
<activity
        android:name=".SQLiteDBActivity" >
        <intent-filter>
           <action android:name="android.intent.action.MAIN" />
           <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
</activity>
```

A number of manifest elements have icon and label attributes for displaying a small icon and a text label, respectively, to users for the corresponding app component.

For example, the icon and label that are set in the <application> element are the default icon and label for each of the app's components.

The icon and label that are set in a component's <intent-filter> are shown to the user whenever that component is presented as an option to fulfill intent.

## Permissions

Android apps must request permission to access personnel user data such as contacts, SMS, camera, files, internet etc. Each permission is identified by a unique label. For example, an app that needs to send and receive SMS messages must have the following line in the manifest:

```
<manifest ... >
   <uses-permission android:name="android.permission.SEND_SMS" />
   <uses-permission android:name="android.permission.RECEIVE_SMS" />
   ...
</manifest>
```

From API level 23, the user can approve or reject some app permissions at runtime. You must declare all permission requests with a <uses-permission> element in the manifest. If the permission is granted, the app is able to use the protected features. If

not, its attempts to access those features fail. A new permission is declared with the <permission> element.

## Device Compatibility

In manifest file is you can declare what types of hardware or software features your app requires and types of devices with which your app is compatible. It can't be installed on devices that don't provide the features or system version that your app requires. The following table shows the most common tags for specifying device compatibility.

| Tag | Description |
|---|---|
| <uses-feature> | It allows you to declare hardware and software features your app needs<br>Example<br><manifest ... ><br><uses-feature<br>android:name="android.hardware.sensor.compass"<br>android:required="true" /><br>...<br></manifest> |
| <uses-sdk> | It indicates the minimum version with which your app is compatible element are overridden by corresponding properties in the build.gradle file.<br><manifest><br><uses-sdk android:minSdkVersion="5" /><br>...<br></manifest> |

**Table-9**

## File conventions

Following are the conventions and rules that generally apply to all elements and attributes in the manifest file.

- Only the <manifest> and <application> elements are required. They each must occur only once, other elements can occur zero or more times.
- Elements at the same level are generally not ordered hence elements can be placed in any order
- All attributes are optional but attributes must be specified so that an element can serve its purpose. If attributes are not provided then it indicates the default value
- Except for some attributes of the root <manifest> element, all attribute names begin with an android: prefix.

## Manifest elements reference

The following table provides links to reference documents for all valid elements in the AndroidManifest.xml file.

| Element | Description |
|---|---|
| <action> | It is used to add an action to an intent filter. |
| <activity> | It is used to declare an activity component. |
| <activity-alias> | It is used to declare an alias for an activity. |
| <application> | It is used to declare the application. |
| <category> | It is used to add category name to an intent filter. |
| <compatible-screens> | It is used to specifies each screen configuration with which the application is compatible. |
| <data> | Adds a data specification to an intent filter. |
| <grant-uri-permission> | Specifies the subsets of app data that the parent content provider has permission to access. |
| <instrumentation> | Declares an Instrumentation class that enables you to monitor an application's interaction with the system. |
| <intent-filter> | Specifies the types of intents that an activity, service, or broadcast receiver can respond to. |
| <manifest> | The root element of the AndroidManifest.xml file. |
| <meta-data> | A name-value pair for an item of additional, arbitrary data that can be supplied to the parent component. |
| <path-permission> | Defines the path and required permissions for a specific subset of data within a content provider. |
| <permission> | Declares a security permission that can be used to limit access to specific components or features of this or other applications. |

| | |
|---|---|
| <permission-group> | Declares a name for a logical grouping of related permissions. |
| <permission-tree> | Declares the base name for a tree of permissions. |
| <provider> | Declares a content provider component. |
| <receiver> | Declares a broadcast receiver component. |
| <service> | Declares a service component. |
| <supports-gl-texture> | Declares a single GL texture compression format that the app supports. |
| <supports-screens> | Declares the screen sizes your app supports and enables screen compatibility mode for screens larger than what your app supports. |
| <uses-configuration> | Indicates specific input features the application requires. |
| <uses-feature> | Declares a single hardware or software feature that is used by the application. |
| <uses-library> | Specifies a shared library that the application must be linked against. |
| <uses-permission> | Specifies a system permission that the user must grant in order for the app to operate correctly. |
| <uses-sdk> | Lets you express an application's compatibility with one or more versions of the Android platform, by means of an API level integer. |

**Table-10**

## Example of Manifest file

The XML below is a simple example AndroidManifest.xml that declares two activities for the app.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="in.edu.baou.listnameactivity"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="22" />
    <application
```

```
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme"  >
        <activity
            android:name="in.edu.baou.listnameactivity.NameDisplayActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name="in.edu.baou.listnameactivity.MultipleChoiceActivity"
            android:label="@string/title_activity_multiple_choice" >
        </activity>
    </application>
</manifest>
```

## Introduction

Generally a program is defined in terms of functionality and data, and an Android application is not an exception. It performs processing, show information on the screen, and takes data from a variety of sources.

To Develop Android applications for mobile devices with resource constraint requires a systematic understanding of the application lifecycle. Important terminology for application building blocks terms are Context, Activity, and Intent. This unit introduces you with the most important components of Android applications and provides you with a more detailed understanding of how Android applications function and interact with one another.

The Activity class is a crucial component of an Android app, and the way activities are launched and put together is a fundamental part of the platform's application model. Unlike programming paradigms in which apps are launched with a main() method, the Android system initiates code in an Activity instance by invoking specific callback methods that correspond to specific stages of its lifecycle.

This document introduces the concept of activities, and then provides some lightweight guidance about how to work with them. For additional information about best practices in architecting your app, see Guide to App Architecture.

## What is activity?

The mobile-app experience differs from its desktop counterpart in that a user's interaction with the app doesn't always begin in the same place. Instead, the user

journey often begins non-deterministically. For instance, if you open an email app from your home screen, you might see a list of emails. By contrast, if you are using a social media app that then launches your email app, you might go directly to the email app's screen for composing an email.

The Activity class is designed to facilitate this paradigm. When one app invokes another, the calling app invokes an activity in the other app, rather than the app as an atomic whole. In this way, the activity serves as the entry point for an app's interaction with the user. You implement an activity as a subclass of the Activity class.

An activity provides the window in which the app draws its UI. This window typically fills the screen, but may be smaller than the screen and float on top of other windows. Generally, one activity implements one screen in an app. For instance, one of an app's activities may implement a Preferences screen, while another activity implements a Select Photo screen.

Most apps contain multiple screens, which means they comprise multiple activities. Typically, one activity in an app is specified as the main activity, which is the first screen to appear when the user launches the app. Each activity can then start another activity in order to perform different actions. For example, the main activity in a simple e-mail app may provide the screen that shows an e-mail inbox. From there, the main activity might launch other activities that provide screens for tasks like writing e-mails and opening individual e-mails.

Although activities work together to form a cohesive user experience in an app, each activity is only loosely bound to the other activities; there are usually minimal dependencies among the activities in an app. In fact, activities often start up activities belonging to other apps. For example, a browser app might launch the Share activity of a social-media app.

To use activities in your app, you must register information about them in the app's manifest, and you must manage activity lifecycles appropriately. The rest of this document introduces these subjects.

## Configuring the AndroidManifest.xml

For your app to be able to use activities, you must declare the activities, and certain of their attributes, in the manifest.

**Declare activities:** To declare your activity, open your manifest file and add an <activity> element as a child of the <application> element. For example:

```
<manifest ... >
  <application ... >
      <activity android:name=".ExampleActivity" />
      ...
  </application ... >
  ...
</manifest >
```

The only required attribute for this element is android:name, which specifies the class name of the activity. You can also add attributes that define activity characteristics such as label, icon, or UI theme.

**Declare intent filters:** Intent filters are a very powerful feature of the Android platform. They provide the ability to launch an activity based not only on an explicit request, but also an implicit one. For example, an explicit request might tell the system to "Start the Send Email activity in the Gmail app". By contrast, an implicit request tells the system to "Start a Send Email screen in any activity that can do the job." When the system UI asks a user which app to use in performing a task, that's an intent filter at work.

You can take advantage of this feature by declaring an <intent-filter> attribute in the <activity> element. The definition of this element includes an <action> element and, optionally, a <category> element and/or a <data> element. These elements combine to specify the type of intent to which your activity can respond. For example, the following code snippet shows how to configure an activity that sends text data, and receives requests from other activities to do so:

```
<activity android:name=".ExampleActivity" android:icon="@drawable/app_icon">
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="text/plain" />
    </intent-filter>
</activity>
```

In this example, the <action> element specifies that this activity sends data. Declaring the <category> element as DEFAULT enables the activity to receive launch requests. The <data> element specifies the type of data that this activity can send. The following code snippet shows how to call the activity described above

```
// Create the text message with a string
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.setType("text/plain");
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
// Start the activity
startActivity(sendIntent);
```

If you intend for your app to be self-contained and not allow other apps to activate its activities, you don't need any other intent filters. Activities that you don't want to make available to other applications should have no intent filters, and you can start them yourself using explicit intents.

**Declare permissions:** You can use the manifest's <activity> tag to control which apps can start a particular activity. A parent activity cannot launch a child activity unless both activities have the same permissions in their manifest. If you declare a <uses-permission> element for a particular activity, the calling activity must have a matching<uses-permission> element.

For example, if your app wants to use a hypothetical app named SocialApp to share a post on social media, SocialApp itself must define the permission that an app calling it must have:

```
<manifest>
<activity android:name="..."
                android:permission="com.google.socialapp.permission.SHARE_POST"
/>
```

Then, to be allowed to call SocialApp, your app must match the permission set in SocialApp's manifest:

```
<manifest>
<uses-permission    android:name="com.google.socialapp.permission.SHARE_POST"
/>
</manifest>
```

**Check your progress-1**

a) When one app invokes another, the calling app invokes an activity in the other app, rather than the app as an atomic whole (True/False)

b) Most apps contain multiple screens, which means they comprise multiple _____
(A) Activities (B) Services (C) Contexts (D) Intents

c) You can use activities in your app, without registering the information about them in the app's manifest.

d) By default, the activity created for you contains the_____event

e) The only required attribute for <activity> element is _____

## Life Cycle of an Activity

The Activity class is an important for application's whole lifecycle. Android applications can be multi-process, and the multiple applications to run concurrently if memory and processing power is available. Applications can have background processes, and applications can be interrupted/paused when events such as message or phone calls occur. There can be only one active application visible to the user at a time or in other words only a single Activity is in the foreground at any given time.

Activities in the Android operating system are managed using an activity stack. When a new activity is started, it is placed on the top of the stack and becomes the running/foreground activity the previous activity always remains below it in the stack, and will not come to the foreground again until the new activity exits.

## Activity States

An activity has essentially four states:

| State | Description |
|---|---|
| Active or running | When an activity is in the foreground of the screen (at the top of the stack). |
| Paused | If an activity has lost focus but is still visible, it is paused. A paused activity maintains all state and member information and remains attached to the window manager, but can be killed by the system in extreme low memory situations. |
| Stopped | If an activity is completely hidden by another activity, it is stopped. It still retains all state and member information, it will often be killed by the system when memory is needed elsewhere. |
| Destroyed | If an activity is paused or stopped, the system can drop the activity from memory by either asking it to finish, or simply killing its process. When it is displayed again to the user, it must be completely restarted and restored to its previous state. |

**Table-11**

## Activity Events

The Activity base class defines a series of events that governs the life cycle of an activity. The Activity class defines the following events:

| Event | Description |
|---|---|
| onCreate() | Called when the activity is first created |
| onStart() | Called when the activity becomes visible to the user |
| onResume() | Called when the activity starts interacting with the user |
| onPause() | Called when the current activity is being paused and the previous activity is being resumed |

| | |
|---|---|
| onStop() | Called when the activity is no longer visible to the user |
| onDestroy() | Called before the activity is destroyed by the system |
| onRestart() | Called when the activity has been stopped and is restarting again |

**Table-12**

By default, the activity created for you contains the onCreate() event. Within this event handler is the code that helps to display the UI elements of your screen.

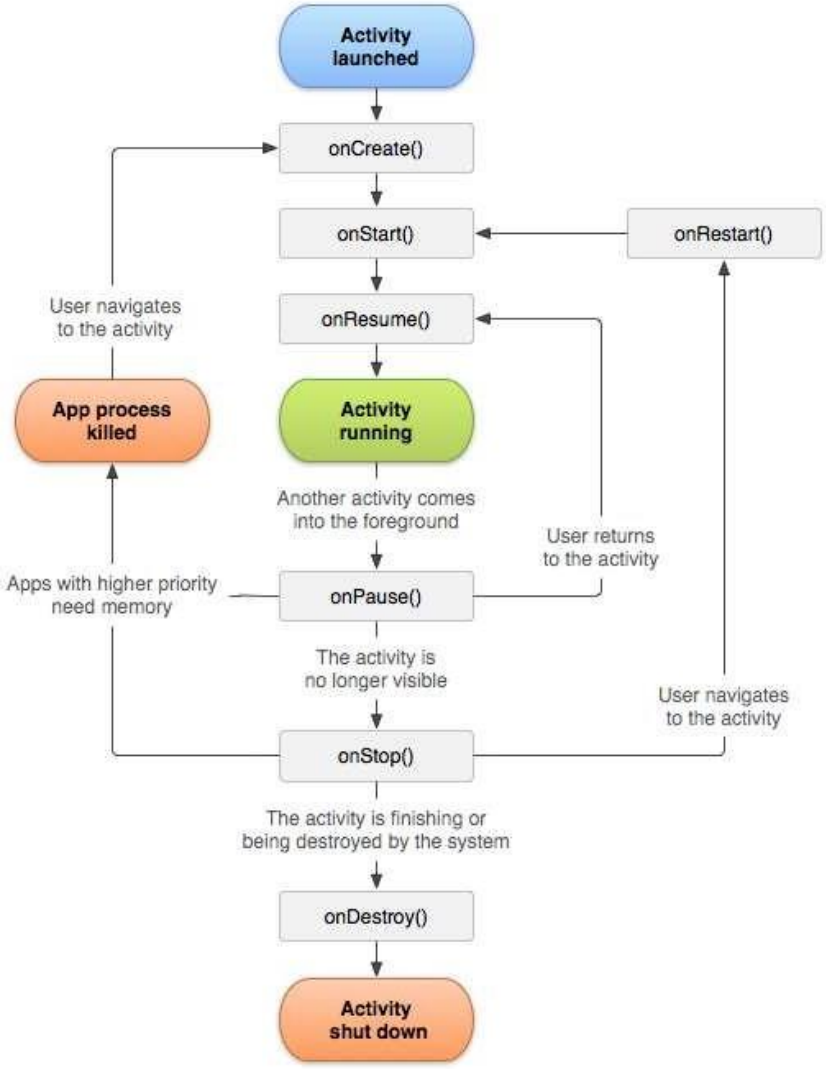



**Figure-45: important state paths of an Activity**

The above figure shows the important state paths of an Activity. The square rectangles represent callback methods you can implement to perform operations when the Activity moves between states. The colored ovals are major states the Activity can be in.

## Understanding Life Cycle of an Activity

The best way to understand the various stages experienced by an activity is to create a new project, implement the various events, and then subject the activity to various user interactions.

1. Create a New Android Studio Project as discussed in section 1.4 with project name ActivityDemo and Main Activity name as MainActivity
2. In the MainActivity.java file, add the following statements in bold:

```
package in.edu.baou.activitydemo;

import android.util.Log;

public class MainActivity extends ActionBarActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      Log.d("Event", "In the onCreate() event");
   }
   public void onStart()
   {
      super.onStart();
      Log.d("Event", "In the onStart() event");
   }
   public void onRestart()
   {
      super.onRestart();
      Log.d("Event", "In the onRestart() event");
   }

   public void onResume()
   {
      super.onResume();
      Log.d("Event", "In the onResume() event");
   }
   public void onPause()
   {
      super.onPause();
      Log.d("Event", "In the onPause() event");
   }
   public void onStop()
   {
      super.onStop();
      Log.d("Event", "In the onStop() event");
```

```
    }
    public void onDestroy()
    {
        super.onDestroy();
        Log.d("Event", "In the onDestroy() event");
    }
}
```

3. Press Shift+F10 or 'Run App' button in taskbar. It will launch following dialog box. Press OK.

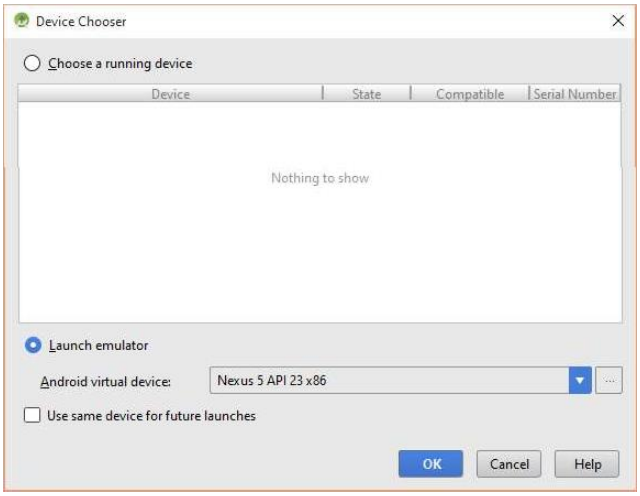



**Figure-46**

4. When the activity is first loaded, you should see the following in the LogCat window.

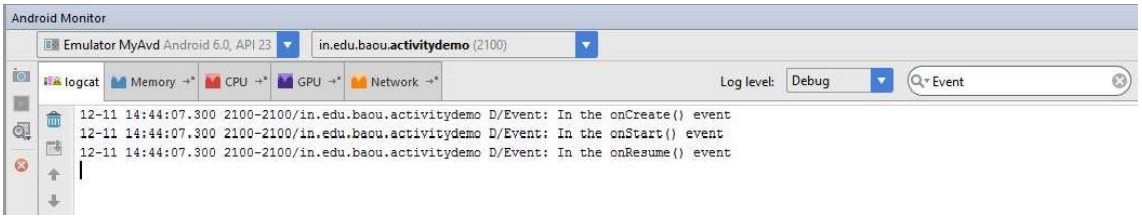



**Figure-47**

5. Now press the back button on the Android Emulator, observe that the following is printed:

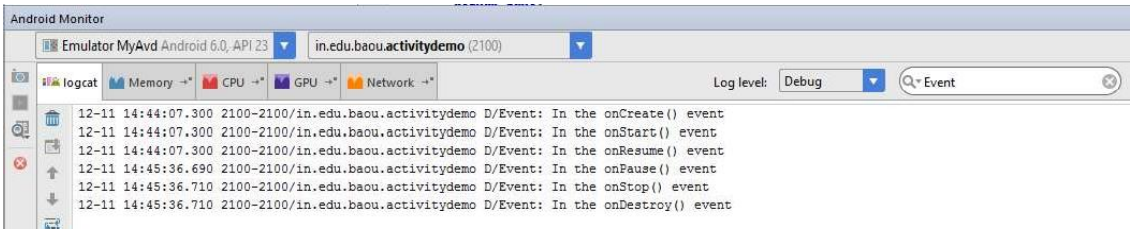



**Figure-48**

6. Click the Home button and hold it there. Click the ActivityDemo icon and observe the following:

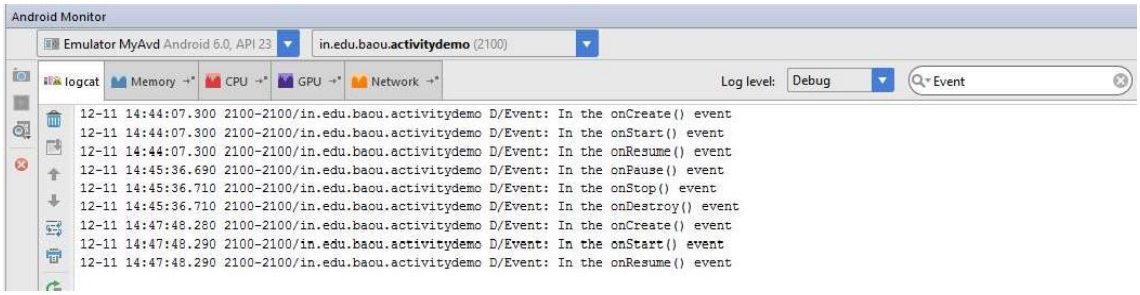



**Figure-49**

7. On Android Emulator from notification area open settings on so that the activity is pushed to the background. Observe the output in the LogCat window:

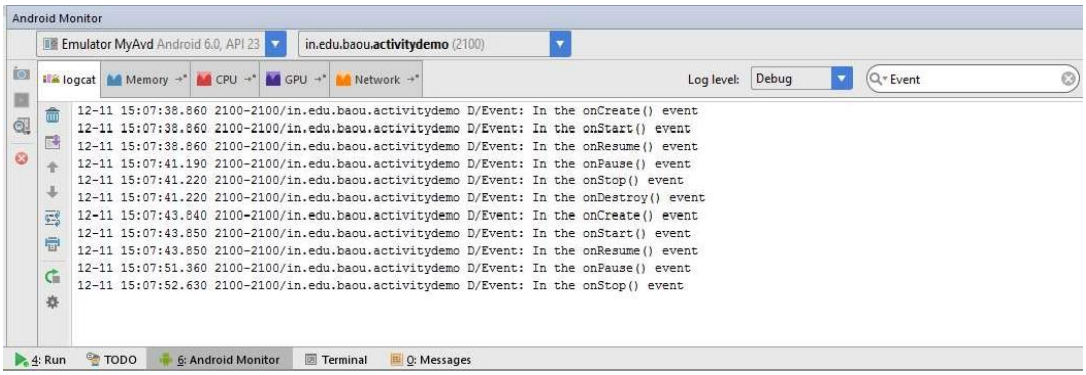



**Figure-50**

8. Notice that the onDestroy() event is not called, indicating that the activity is still in memory. Exit the settings by pressing the Back button. The activity is now visible again. Observe the output in the LogCat window:

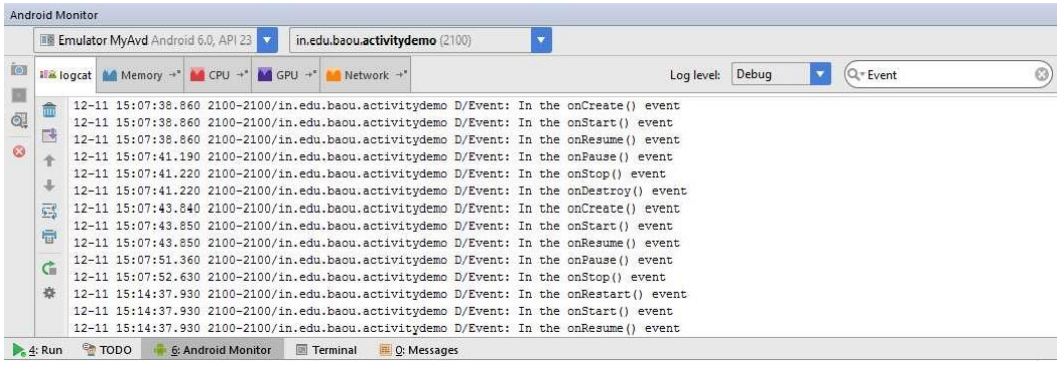



**Figure-51**

Please note that the onRestart() event is now fired, followed by the onStart() and onResume() events.

This application uses logging feature of Android. To add logging support to ActivityDemo app, edit the file MainActivity.java to add the following import statement for the Log class:

**import android.util.Log;**

Logging is a valuable resource for debugging and learning Android. Android logging features are in the Log class of the android.util package. Some helpful methods in the android.util.Log class are shown in Table. We have used Log.d() method to print message in LogCat Window when particular event of activity fired.

| Method | Purpose |
|---|---|
| Log.e() | Log errors |
| Log.w() | Log warnings |
| Log.i() | Log information messages |
| Log.d() | Log debug messages |
| Log.v() | Log verbose messages |

**Table-13**

## Context

As the name suggests, it is the context of current state of the application/object. It lets newly created objects understand what has been going on. Typically you call it to get information regarding other part of your program (activity, package/application). The application Context is the central location for all top-level application functionality. The Context class can be used to manage application-specific configuration details as well as application-wide operations and data. Use the application Context to access settings and resources shared across multiple Activity instances.

**Retrieving the Application Context**

You can get the context by invoking **getApplicationContext(), getContext(), getBaseContext()** or this (when in the activity class). You can retrieve the Context for the current process using the getApplicationContext() method, like this: **Context context = getApplicationContext();**

**Uses of the Application Context**

After you have retrieved a valid application Context, it can be used to access application-wide features and services. Typical uses of context are:

1) **Creating new views, adapters, listeners object**

```
TextView tv = new TextView(getContext());
ListAdapter adapter = new SimpleCursorAdapter(getApplicationContext(), ...);
```

2) **Retrieving Application Resources:** You can retrieve application resources using the getResources() method of the application Context. The most straightforward way to retrieve a resource is by using its resource identifier, a unique number automatically generated within the R.java class.The following example retrieves a String instance from the application resources by its resource ID:

```
String greeting = getResources().getString(R.string.settings);
```

3) **Retrieving Shared Application Preferences:** You can retrieve shared application preferences using the getSharedPreferences() method of the application Context. The SharedPreferences class can be used to save simple application data, such as configuration settings.

4) **Accessing Other Application Functionality Using Context:** The application Context provides access to a number of other top-level application features.
   Here are a few more things you can do with the application Context:
   - Launch Activity instances
   - Inspect and enforce application permissions
   - Retrieve assets packaged with the application
   - Request a system service (for example, location service)
   - Manage private application files, directories, and databases

## Activity Transition

In the course of the lifetime of an Android application, the user might transition between a numbers of different Activity instances. At times, there might be multiple Activity instances on the activity stack. Developers need to pay attention to the lifecycle of each Activity during these transitions.

Some Activity instances such as the application splash/startup screen are shown and then permanently discarded when the Main menu screen Activity takes over. The user cannot return to the splash screen Activity without re-launching the application.

Other Activity transitions are temporary, such as a child Activity displaying a dialog box, and then returning to the original Activity (which was paused on the activity stack and now resumes). In this case, the parent Activity launches the child Activity and expects a result.

**Transitioning between Activities with Intents:** As previously mentioned, Android applications can have multiple entry points. There is no main() function, such as you find in iPhone development. Instead, a specific Activity can be designated as the main Activity to launch by default within the AndroidManifest.xml file; Other Activities might be designated to launch under specific circumstances. For example, a music application might designate a generic Activity to launch by default from the Application menu, but also define specific alternative entry point Activities for accessing specific music playlists by playlist ID or artists by name.

**Launching a New Activity by Class Name:** You can start activities in several ways. The simplest method is to use the Application Context object to call the startActivity() method, which takes a single parameter, an Intent.

Intent (android.content.Intent) is an asynchronous message mechanism used by the Android operating system to match task requests with the appropriate Activity or Service (launching it, if necessary) and to dispatch broadcast Intents events to the system at large. For now, though, we focus on Intents and how they are used with

Activities. The following line of code calls the startActivity() method with an explicit Intent. This Intent requests the launch of the target Activity named MyDrawActivity by its class. This class is implemented elsewhere within the package.

```
startActivity(new  Intent(getApplicationContext(),
MyDrawActivity.class));
```

This line of code might be sufficient for some applications, which simply transition from one Activity to the next. However, you can use the Intent mechanism in a much more robust manner. For example, you can use the Intent structure to pass data between Activities.

**Creating Intents with Action and Data:** You've seen the simplest case to use Intent to launch a class by name. Intents need not specify the component or class they want to launch explicitly. Instead, you can create an Intent Filter and register it within the Android Manifest file. The Android operating system attempts to resolve the Intent requirements and launch the appropriate Activity based on the filter criteria.

The guts of the Intent object are composed of two main parts: the action to be performed and the data to be acted upon. You can also specify action/data pairs using Intent Action types and Uri objects. An Uri object represents a string that gives the location and name of an object. Therefore, an Intent is basically saying "do this" (the action) to "that" (the Uri describing what resource to do the action to). The most common action types are defined in the Intent class, including ACTION_MAIN (describes the main entry point of an Activity) and ACTION_EDIT (used in conjunction with a Uri to the data edited).You also find Action types that generate integration points with Activities in other applications, such as the Browser or Phone Dialer.

**Launching an Activity Belonging to another Application:** Initially, your application might be starting only Activities defined within its own package

However, with the appropriate permissions, applications might also launch external Activities within other applications. For example, a Customer Relationship Management (CRM) application might launch the Contacts application to browse the

Contact database, choose a specific contact, and return that Contact's unique identifier to the CRM application for use.

Here is an example of how to create a simple Intent with a predefined Action (ACTION_DIAL) to launch the Phone Dialer with a specific phone number to dial in the form of a simple Uri object:

```
Uri number = Uri.parse(tel:5555551212);
Intent dial = new Intent(Intent.ACTION_DIAL, number);
startActivity(dial);
```

## Introduction

In Android a service is an application that runs in the background without any interaction with the user. For example, while using an application, you may want to download some file at the same time. In this case, the code that is downloading file has no need to interact with the user, and hence it can be run as a service. Services are also perfect for circumstances in which there is no need to present a user interface (UI) to the user. For example, consider an application that continually logs the geographical coordinates of the device. In this case, you can write a service to do that in the background. You can create your own services and use them to perform background tasks asynchronously.

To improve application responsiveness and performance, consider implementing a service to handle the task outside the main application lifecycle. Any Services exposed by an Android application must be registered in the Android Manifest file.

## Uses Services

You can use services for different purposes. Generally, you use a service when no input is required from the user. Here are some circumstances in which you might want to implement or use an Android service:

A weather, email, or social network app might implement a service to routinely check for updates.

•

- A photo or media app that keeps its data in sync online might implement a service to package and upload new content in the background when the device is idle.
- A video-editing app might offload heavy processing to a queue on its service in order to avoid affecting overall system performance for non-essential tasks.
- A news application might implement a service to "pre-load" content by downloading news stories in advance of when the user launches the application, to improve performance.

## Creating a Service

To create service you must defined a class that extends the Service base class. Inside your service class, you have to implement four methods discussed below:

| Method | Description |
|---|---|
| **onStartCommand()** | • The system calls this method when another component, such as an activity, requests that the service be started, by calling startService().<br>• Once this method executes, the service is started and can run in the background indefinitely.<br>• It is your responsibility to stop the service when its work is done, by calling stopSelf() or stopService().<br>• If you only want to provide binding, you don't need to implement this method. |
| **onBind()** | • The system calls this method when another component wants to bind with the service by calling bindService().<br>• In your implementation of this method, you must provide an interface that clients use to communicate with the service, by returning an IBinder.<br>• If you don't want to allow binding, then you should return null. |
| **onCreate()** | • The system calls this method when the service is first created, to perform one-time setup procedures before it |

| Method | Description |
|---|---|
| | calls either onStartCommand() or onBind().<br>• If the service is already running, this method is not called. |
| **onDestroy()** | • The system calls this method when the service is no longer used and is being destroyed.<br>• This method should be implemented to clean up any resources such as threads, registered listeners, receivers, etc.<br>• This is the last call the service receives. |

**Table-14**

## Start and Stop a Service

You can use Intents and Activities to launch services using the startService() and bindService() methods. A service can essentially take two forms. The difference between two is as follows:

| startService() | bindService() |
|---|---|
| A service is "started" when an application component starts it by calling startService() | A service is "bound" when an application component binds to it by calling bindService() |
| Once started, a service can run in the background indefinitely, even if the component that started it is destroyed | A bound service runs only as long as another application component is bound to it. Multiple components can bind to the service. |
| Usually, a started service performs a single operation and does not return a result to the caller. For example, it might download or upload a file over the network. When the operation is done, the service should stop itself | A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, and even do so across processes with inter process communication (IPC) |

**Table-15**

## Service Life Cycle

Like an activity, a service has lifecycle callback methods that you can implement to monitor changes in the service's state and perform work at the appropriate times as discussed above. Below Figure illustrates the typical callback methods for a service for that are created by startService() and from those created by bindService.

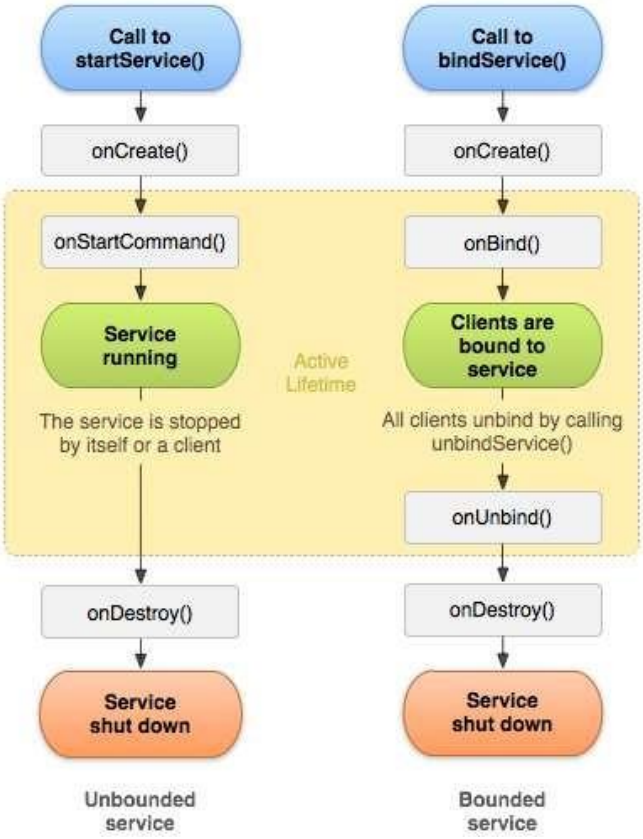



**Figure-52**

## Creating your own service

We will create service to logs counter which starts from 1 and incremented by one at interval of one second. To do so perform following steps:

1. Create a New Android Studio Project with project name ServiceDemo and Main Activity name as ServiceActivity
2. Add new service by right click on package and select New ➔ Service ➔ Service and click.

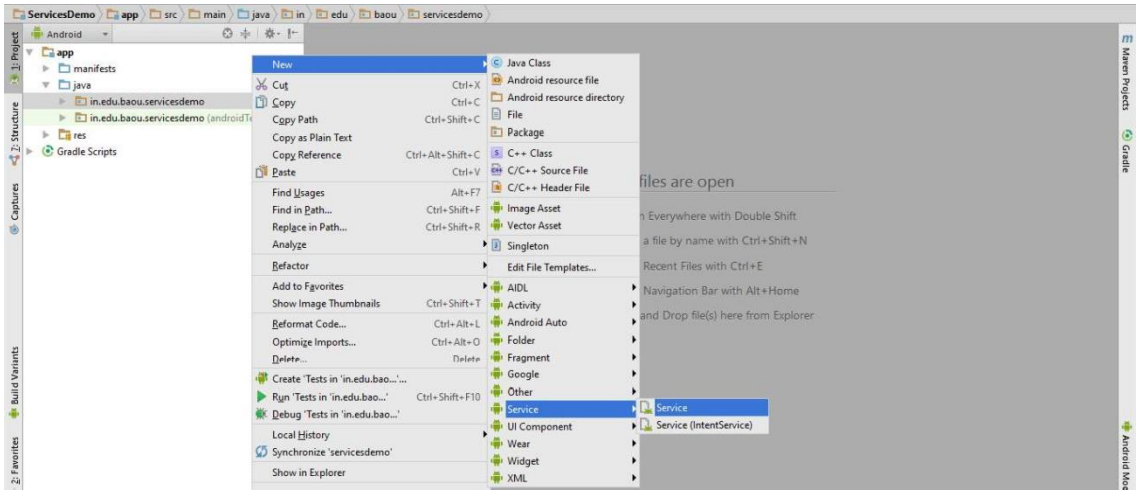



**Figure-53**

3. In dialog box, Enter class name as TimerService as shown in figure and press finish Button.

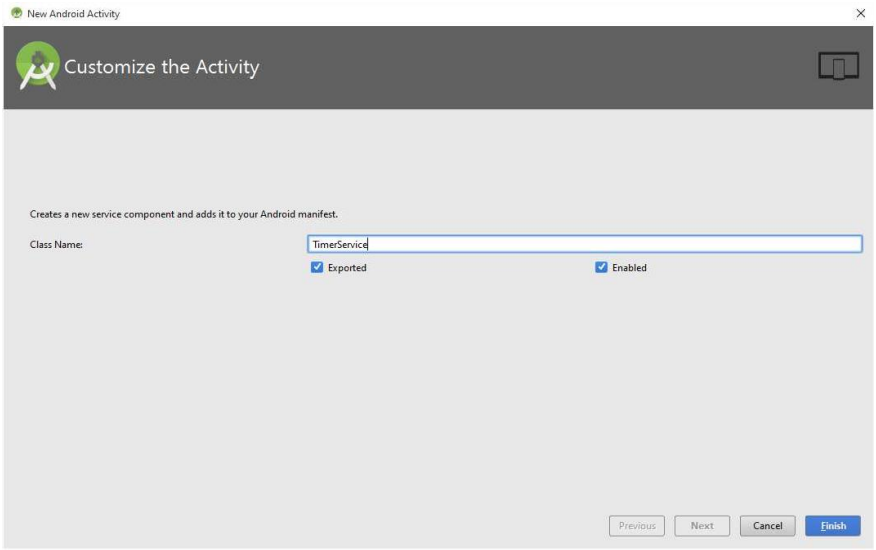



**Figure-54**

4. Write following code inside TimerService class

```
package in.edu.baou.servicesdemo;

import android.util.Log;
import  android.widget.Toast;

import java.util.Timer;
import  java.util.TimerTask;
```

```
public class TimerService extends Service {
   int counter = 0;
   Timer timer = new Timer();

   public TimerService() {
   }

   @Override
   public IBinder onBind(Intent intent) {
      return null;
   }
   @Override
   public int onStartCommand(Intent intent, int flags, int startId) {
      Toast.makeText(this, "Service Started!", Toast.LENGTH_LONG).show();
      timer.scheduleAtFixedRate(new TimerTask() {
         public void run() {
            Log.d("MyService", String.valueOf(++counter));
         }
      }, 0, 1000);
      return START_STICKY;
   }
   @Override
   public void onDestroy() {
      super.onDestroy();
      if (timer != null){
         timer.cancel();
      }
      Toast.makeText(this, "Service Destroyed!", Toast.LENGTH_LONG).show();
   }
}
```

5. In android_service.xml file add the following statements in bold

```
<RelativeLayout 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:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
android:paddingBottom="@dimen/activity_vertical_margin"
tools:context=".ServiceActivity">

   <TextView android:text="Service Demonstration"
android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:id="@+id/textView" />

   <Button
      android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:text="Start Timer Service"
        android:id="@+id/btnStartTimer"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_below="@+id/textView" />

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Stop Timer Service"
        android:id="@+id/btnStopTimer"
        android:layout_below="@+id/btnStartTimer"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />

</RelativeLayout>
```

6. Add the following statements in bold to the ServiceActivity.java file:

```
package in.edu.baou.servicesdemo;

import android.content.Intent;
import android.view.View;
import  android.widget.Button;
public class ServiceActivity extends ActionBarActivity {

    Button startTimer,stopTimer;

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

        startTimer = (Button)findViewById(R.id.btnStartTimer);
        stopTimer = (Button)findViewById(R.id.btnStopTimer);

        startTimer.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                startService(new Intent(getBaseContext(), TimerService.class));
            }
        });

        stopTimer.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                stopService(new Intent(getBaseContext(), TimerService.class));
            }
```

```
        });
}
```

7. Press Shift+F10 or 'Run App' button in taskbar. It will launch following dialog box. Press OK.

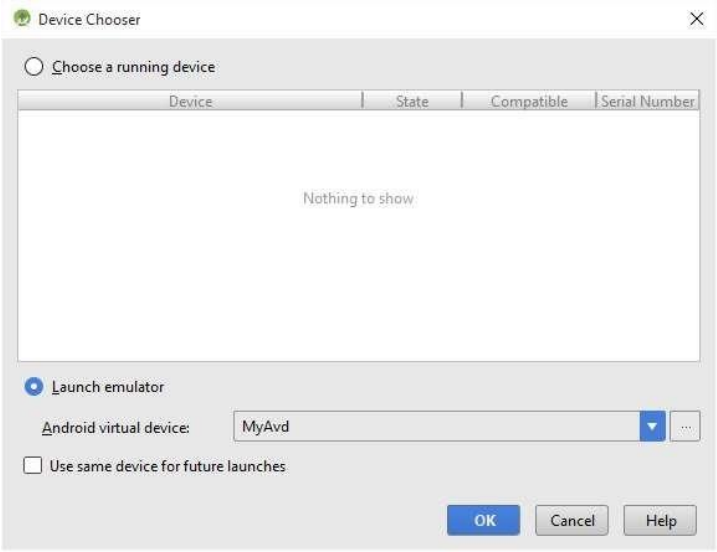



**Figure-55**

8. It will open activity in emulator as shown below. Clicking the **START TIMER SERVICE** button will start the service as shown below. To stop the service, click the **STOP TIMER SERVICE**.

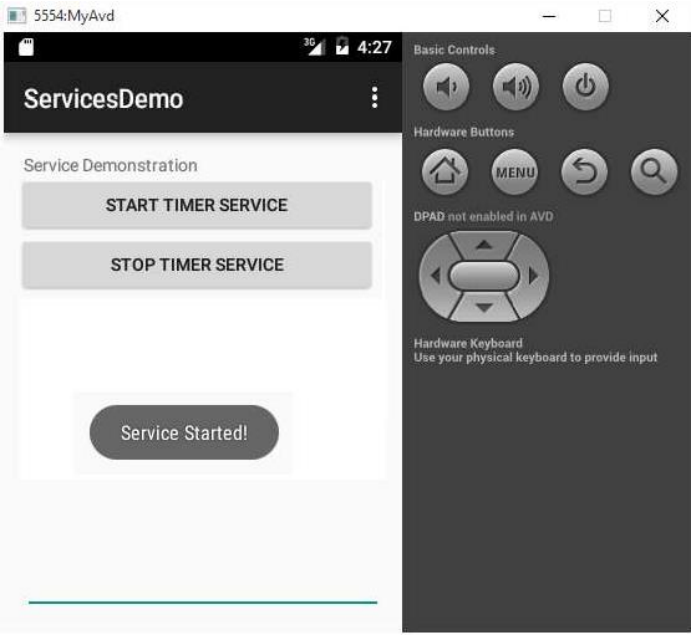



**Figure-56**

9. Once service is started, you can see counter value incremented by in LogCat window

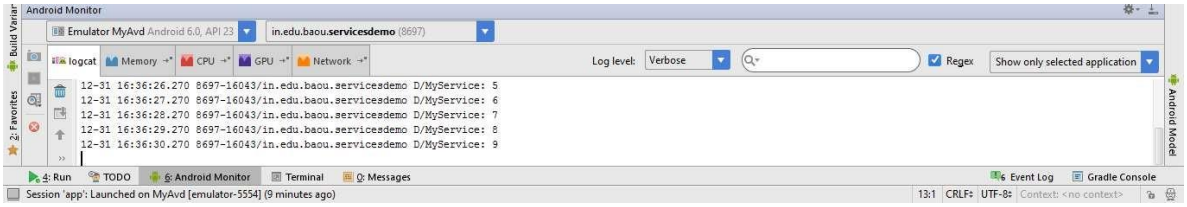



**Figure-57**

## Explanation

- Inside project layout file we created two buttons to start and stop service with ID btnStartTimer and btnStopTimer..
- Inside ServiceActivity we define two button objects that represents button in layout file.
- The findViewByID() method is used to take reference of button.
- Button clicked event is handled by onClick() method of OnClickListener associated to button using setOnClickListener().
- Inside TimerService class we define counter variable which initialized to zero at the start of service and increment by one every one second using Timer class scheduledAtFixedRate() Method.
- The value of counter is logs inside LogCat window using Log.d() Method with tag “MyService”.
- Toast is temporary message displayed on screen such as “Service Started” in step-8 and is displayed using makeText method of Toast class.

## Introduction

Intent is an abstract description of an operation to be performed. It can be used to launch an Activity, broadcastIntent to send it to any interested BroadcastReceiver components, and to communicate with a background Service.

Intent provides a facility for performing late runtime binding between the codes in different applications. Its most significant use is in the launching of activities, where it can be thought of as the glue between activities. It is basically a passive data structure holding an abstract description of an action to be performed.

## Intent Structure

The intent has primary attributes which are mandatory and secondary attributes which are optional.

### Primary Attributes

**Primary Attributes:** The primary pieces of information in intent are:

1. **Action:** The general action to be performed, such as ACTION_VIEW, ACTION_EDIT, ACTION_MAIN, etc.
2. **Data:** The data to operate on, such as a person record in the contacts database, expressed as an Uri.

Some examples of action/data pairs are:

- ACTION_VIEW ***content://contacts/people/9*** : Display information about the person whose identifier is "9".
- ACTION_DIAL ***content://contacts/people/9*** : Display the phone dialer with the person filled in.
- ACTION_VIEW ***tel:123*** : Display the phone dialer with the given number filled in. Note how the VIEW action does what is considered the most reasonable thing for a particular URI.
- ACTION_DIAL ***tel:123*** : Display the phone dialer with the given number filled in.
- ACTION_EDIT ***content://contacts/people/9*** : Edit information about the person whose identifier is "9".
- ACTION_VIEW ***content://contacts/people/*** : Display a list of people, which the user can browse through. This example is a typical top-level entry into the Contacts application, showing you the list of people. Selecting a particular person to view would result in a new intent {ACTION_VIEWcontent://contacts/people/N } being used to start an activity to display that person.

## Secondary Attributes

In addition to these primary attributes, there are a number of secondary attributes that you can also include with intent:

- **Category:** Gives additional information about the action to execute. For example, CATEGORY_LAUNCHERmeans it should appear in the Launcher as a top-level application, while CATEGORY_ALTERNATIVE means it should be included in a list of alternative actions the user can perform on a piece of data.
- **Type:** Specifies an explicit type (a MIME type) of the intent data. Normally the type is inferred from the data itself. By setting this attribute, you disable that evaluation and force an explicit type.
- **Component:** Specifies an explicit name of a component class to use for the intent. Normally this is determined by looking at the other information in the intent (the action, data/type, and categories) and matching that with a component that can handle it. If this attribute is set then none of the evaluation is performed, and this

component is used exactly as is. By specifying this attribute, all of the other Intent attributes become optional.

- **Extras:** This is a Bundle of any additional information. This can be used to provide extended information to the component. For example, if we have a action to send an e-mail message, we could also include extra pieces of data here to supply a subject, body, etc.

## Other Operations on Intent

Here are some examples of other operations you can specify as intents using these additional parameters:

- **ACTION_MAIN with category CATEGORY_HOME:** Launch the home screen.
- **ACTION_GET_CONTENT with MIME type vnd.android.cursor.item/phone:** Display the list of people's phone numbers, allowing the user to browse through them and pick one and return it to the parent activity.
- **ACTION_GET_CONTENT with MIME type */* and category CATEGORY_OPENABLE**: Display all pickers for data that can be opened and allowing the user to pick one of them and then some data inside of it and returning the resulting URI to the caller. This can be used, for example, in an e-mail application to allow the user to pick some data to include as an attachment.

There are a variety of standard Intent action and category constants defined in the Intent class, but applications can also define their own, for example, the standard ACTION_VIEW is called "android.intent.action.VIEW".

## Types of Intent

There are two primary forms of intents you will use.

- **Explicit Intents** have specified a component which provides the exact class to be run. Often these will not include any other information, simply being a way for an application to launch various internal activities it has as the user interacts with the application.

- **Implicit Intents** have not specified a component; instead, they must include enough information for the system to determine which of the available components is best to run for that intent.

## Intent Resolution

When using implicit intents, given such an arbitrary intent we need to know what to do with it. This is handled by the process of Intent resolution, which maps an Intent to an Activity, BroadcastReceiver, or Service that can handle it.

The intent resolution mechanism basically revolves around matching Intent against all of the <intent-filter> descriptions in the installed application packages.

There are three pieces of information in the Intent that are used for resolution: the action, type, and category. Using this information, a query is done on the PackageManager for a component that can handle the intent. The appropriate component is determined based on the intent information supplied in the AndroidManifest.xml file as follows:

- The action, if given, must be listed by the component as one it handles.
- The type is retrieved from the Intent's data, if not already supplied in the Intent. Like the action, if a type is included in the intent (either explicitly or implicitly in its data), then this must be listed by the component as one it handles.
- For data that is not a content: URI and where no explicit type is included in the Intent, instead the scheme of the intent data (such as http: or mailto:) is considered. Again like the action, if we are matching a scheme it must be listed by the component as one it can handle.
- The categories, if supplied, must all be listed by the activity as categories it handles. That is, if you include the categories CATEGORY_LAUNCHER and CATEGORY_ALTERNATIVE, then you will only resolve to components with an intent that lists both of those categories. Activities will very often need to support the CATEGORY_DEFAULT so that they can be found by startActivity.

## Example of Intent

For example, consider the Note Pad sample application that allows a user to browse through a list of notes data and view details about individual items.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="in.edu.baou.notepad">
  <application android:icon="@drawable/app_notes"  android:label="@string/app_name">
        <provider class=".NotePadProvider" android:authorities="in.edu.baou.provider.NotePad" />

        <activity class=".NotesList" android:label="@string/title_notes_list">
                <intent-filter>
                        <action android:name="android.intent.action.MAIN" />
                        <category android:name="android.intent.category.LAUNCHER" />
                </intent-filter>
                <intent-filter>
                        <action android:name="android.intent.action.VIEW" />
                        <action android:name="android.intent.action.EDIT" />
                        <action android:name="android.intent.action.PICK" />
                        <category android:name="android.intent.category.DEFAULT" />
                        <data android:mimeType="vnd.android.cursor.dir/vnd.google.note" />
                </intent-filter>
                <intent-filter>
                        <action android:name="android.intent.action.GET_CONTENT" />
                        <category android:name="android.intent.category.DEFAULT" />
                        <data android:mimeType="vnd.android.cursor.item/vnd.google.note" />
                </intent-filter>
      </activity>

      <activity class=".NoteEditor" android:label="@string/title_note">
                <intent-filter android:label="@string/resolve_edit">
                        <action android:name="android.intent.action.VIEW" />
                        <action android:name="android.intent.action.EDIT" />
                        <category android:name="android.intent.category.DEFAULT" />
                        <data android:mimeType="vnd.android.cursor.item/vnd.google.note" />
                </intent-filter>
                <intent-filter>
                        <action android:name="android.intent.action.INSERT" />
                        <category android:name="android.intent.category.DEFAULT" />
                        <data android:mimeType="vnd.android.cursor.dir/vnd.google.note" />
```

```
            </intent-filter>
        </activity>
        <activity class=".TitleEditor" android:label="@string/title_edit_title"
                android:theme="@android:style/Theme.Dialog">
            <intent-filter android:label="@string/resolve_title">
                <action android:name="com.android.notepad.action.EDIT_TITLE" />
                <category android:name="android.intent.category.DEFAULT" />
                <category android:name="android.intent.category.ALTERNATIVE" />
                <category android:name="android.intent.category.SELECTED_ALTERNATIVE" />
                <data android:mimeType="vnd.android.cursor.item/vnd.google.note" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

## Explanation of Example

In above example, the first activity, in.edu.baou.provider.notepad,NotesList, serves as our main entry into the app. It can do three things as described by its three intent templates:

```
<intent-filter>
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

This provides a top-level entry into the NotePad application: the standard MAIN action is a main entry point (not requiring any other information in the Intent), and the LAUNCHER category says that this entry point should be listed in the application launcher.

```
<intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <action android:name="android.intent.action.EDIT" />
    <action android:name="android.intent.action.PICK" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:mimeType="vnd.android.cursor.dir/vnd.google.note" />
</intent-filter>
```

This declares the things that the activity can do on a directory of notes. The type being supported is given with the <type> tag, where vnd.android.cursor.dir/vnd.google.note is a URI from which a Cursor of zero or more items (vnd.android.cursor.dir) can be retrieved which holds our note pad data (vnd.google.note). The activity allows the user to view or edit the directory of data (via the VIEW and EDIT actions), or to pick a particular note and return it to the caller (via the PICK action). Note also the DEFAULT category supplied here: this is *required* for the startActivity method to resolve your activity when its component name is not explicitly specified.

```
<intent-filter>
    <action android:name="android.intent.action.GET_CONTENT" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:mimeType="vnd.android.cursor.item/vnd.google.note" />
</intent-filter>
```

This filter describes the ability to return to the caller a note selected by the user without needing to know where it came from. The data type vnd.android.cursor.item/vnd.google.note is a URI from which a Cursor of exactly one (vnd.android.cursor.item) item can be retrieved which contains our note pad data (vnd.google.note). The GET_CONTENT action is similar to the PICK action, where the activity will return to its caller a piece of data selected by the user. Here, however, the caller specifies the type of data they desire instead of the type of data the user will be picking from.

Given these capabilities, the following intents will resolve to the NotesList activity:

- **{ action=android.app.action.MAIN }** matches all of the activities that can be used as top-level entry points into an application.
- **{ action=android.app.action.MAIN, category=android.app.category.LAUNCHER }** is the actual intent used by the Launcher to populate its top-level list.
- **{ action=android.intent.action.VIEW data=content://com.google.provider.NotePad/notes }** displays a list of all the notes under "content://com.google.provider.NotePad/notes", which the user can browse through and see the details on.

- **{ action=android.app.action.PICK data=content://com.google.provider.NotePad/notes }** provides a list of the notes under "content://com.google.provider.NotePad/notes", from which the user can pick a note whose data URL is returned back to the caller.
- **{ action=android.app.action.GET_CONTENT type=vnd.android.cursor.item/vnd.google.note }** is similar to the pick action, but allows the caller to specify the kind of data they want back so that the system can find the appropriate activity to pick something of that data type.
- The second activity, in.edu.baou.notepad.NoteEditor, shows the user a single note entry and allows them to edit it. It can do two things as described by its two intent templates:

```
<intent-filter android:label="@string/resolve_edit">
    <action android:name="android.intent.action.VIEW" />
    <action android:name="android.intent.action.EDIT" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:mimeType="vnd.android.cursor.item/vnd.google.note" />
</intent-filter>
```

The first, primary, purpose of this activity is to let the user interact with a single note, as described by the MIME type vnd.android.cursor.item/vnd.google.note. The activity can either VIEW a note or allow the user to EDIT it. Again we support the DEFAULT category to allow the activity to be launched without explicitly specifying its component.

```
<intent-filter>
    <action android:name="android.intent.action.INSERT" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:mimeType="vnd.android.cursor.dir/vnd.google.note" />
</intent-filter>
```

The secondary use of this activity is to insert a new note entry into an existing directory of notes. This is used when the user creates a new note: the INSERT action is executed on the directory of notes, causing this activity to run and have the user create the new note data which it then adds to the content provider.

Given these capabilities, the following intents will resolve to the NoteEditor activity:

- **{action=android.intent.action.VIEW data=content://in.edu.baou.provider.NotePad/notes/*{ID}*}** shows the user the content of note ***{ID}***.
- **{ action=android.app.action.EDIT data=content:// in.edu.baou.provider.NotePad/notes/*{ID}* }** allows the user to edit the content of note ***{ID}***.
- **{ action=android.app.action.INSERT data=content:// in.edu.baou.provider.NotePad/notes }** creates a new, empty note in the notes list at "content://com.google.provider.NotePad/notes" and allows the user to edit it. If they keep their changes, the URI of the newly created note is returned to the caller.
- The last activity, com.android.notepad.TitleEditor, allows the user to edit the title of a note. This could be implemented as a class that the application directly invokes (by explicitly setting its component in the Intent), but here we show a way you can publish alternative operations on existing data:

```
<intent-filter android:label="@string/resolve_title">
    <action android:name="com.android.notepad.action.EDIT_TITLE" />
    <category android:name="android.intent.category.DEFAULT" />
    <category android:name="android.intent.category.ALTERNATIVE" />
    <category android:name="android.intent.category.SELECTED_ALTERNATIVE"
/>
    <data android:mimeType="vnd.android.cursor.item/vnd.google.note" />
</intent-filter>
```

- In the single intent template here, we have created our own private action calledcom.android.notepad.action.EDIT_TITLE which means to edit the title of a note. It must be invoked on a specific note like the previous view and edit actions, but here displays and edits the title contained in the note data.
- In addition to supporting the default category as usual, our title editor also supports two other standard categories: ALTERNATIVE and SELECTED_ALTERNATIVE. Implementing these categories allows others to find the special action it provides without directly knowing about it, through the PackageManager.queryIntentActivityOptions(ComponentName, Intent[], Intent, int) method, or more often to build dynamic menu items with Menu.addIntentOptions(int, int, int, ComponentName, Intent[], Intent, int,

MenuItem[]). Note that in the intent template here was also supply an explicit name for the template (via android:label="@string/resolve_title") to better control what the user sees when presented with this activity as an alternative action to the data they are viewing.

- Given these capabilities, the following intent will resolve to the TitleEditor activity:
- {action=com.android.notepad.action.EDIT_TITLE data=content://com.google.provider.NotePad/notes/*{ID}*} displays and allows the user to edit the title associated with note *{ID}*.

## Standard Activity Actions

These are the current standard actions that Intent defines for launching activities (usually through Context#startActivity. The most important, and by far most frequently used, are ACTION_MAIN and ACTION_EDIT.

| | | |
|---|---|---|
| ACTION_MAIN | ACTION_DIAL | ACTION_RUN |
| ACTION_VIEW | ACTION_CALL | ACTION_SYNC |
| ACTION_ATTACH_DATA | ACTION_SEND | ACTION_PICK_ACTIVITY |
| ACTION_EDIT | ACTION_SENDTO | ACTION_SEARCH |
| ACTION_PICK | ACTION_ANSWER | ACTION_WEB_SEARCH |
| ACTION_CHOOSER | ACTION_INSERT | ACTION_FACTORY_TEST |
| ACTION_GET_CONTENT | ACTION_DELETE | |

## Standard Broadcast Actions

These are the current standard actions that Intent defines for receiving broadcasts (usually through registerReceiver or a <receiver> tag in a manifest).

| | |
|---|---|
| ACTION_TIME_TICK | ACTION_PACKAGE_DATA_CLEARED |
| ACTION_TIME_CHANGED | ACTION_PACKAGES_SUSPENDED |
| ACTION_TIMEZONE_CHANGED | ACTION_PACKAGES_UNSUSPENDED |
| ACTION_BOOT_COMPLETED | ACTION_UID_REMOVED |
| ACTION_PACKAGE_ADDED | ACTION_BATTERY_CHANGED |
| ACTION_PACKAGE_CHANGED | ACTION_POWER_CONNECTED |
| ACTION_PACKAGE_REMOVED | ACTION_POWER_DISCONNECTED |
| ACTION_PACKAGE_RESTARTED | ACTION_SHUTDOWN |

## Introduction

The purpose of a *permission* is to protect the privacy of an Android user. Android apps must request permission to access sensitive user data (such as contacts and SMS), as well as certain system features (such as camera and internet). Depending on the feature, the system might grant the permission automatically or might prompt the user to approve the request.

A central design point of the Android security architecture is that no app, by default, has permission to perform any operations that would adversely impact other apps, the operating system, or the user. This includes reading or writing the user's private data (such as contacts or emails), reading or writing another app's files, performing network access, keeping the device awake, and so on.

This unit provides an overview of how Android permissions work, including: how permissions are presented to the user, the difference between install-time and runtime permission requests, how permissions are enforced, and the types of permissions and their groups.

## Permission Approval

An app must publicize the permissions it requires by including <uses-permission> tags in the app manifest. For example, an app that needs to send SMS messages would have this line in the manifest:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="com.example.snazzyapp">
    <uses-permission  android:name="android.permission.SEND_SMS"/>
```

```
    <application ...>
        ...
    </application>
</manifest>
```

If your app lists normal permissions in its manifest (that is, permissions that don't pose much risk to the user's privacy or the device's operation), the system automatically grants those permissions to your app.

If your app lists dangerous permissions in its manifest (that is, permissions that could potentially affect the user's privacy or the device's normal operation), such as the SEND_SMS permission above, the user must explicitly agree to grant those permissions.

## Request prompts for dangerous permissions

Only dangerous permissions require user agreement. The way Android asks the user to grant dangerous permissions depends on the version of Android running on the user's device, and the system version targeted by your app.

**Runtime requests (Android 6.0 and higher)**

If the device is running Android 6.0 (API level 23) or higher, and the app's targetSdkVersion is 23 or higher, the user isn't notified of any app permissions at install time. Your app must ask the user to grant the dangerous permissions at runtime. When your app requests permission, the user sees a system dialog as shown in figure 1 telling the user which permission group your app is trying to access. The dialog includes a Deny and Allow button.

If the user denies the permission request, the next time your app requests the permission, the dialog contains a checkbox that, when checked, indicates the user doesn't want to be prompted for the permission again as shown in figure 2.

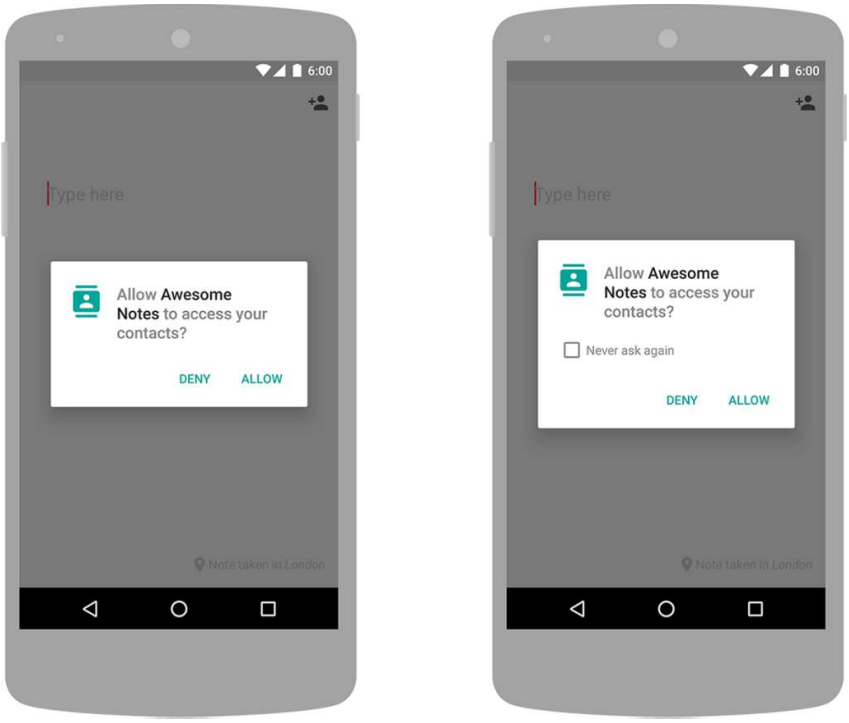



**Figure-58**

If the user checks the Never ask again box and taps Deny, the system no longer prompts the user if you later attempt to requests the same permission.

Even if the user grants your app the permission it requested you cannot always rely on having it. Users also have the option to enable and disable permissions one-by-one in system settings. You should always check for and request permissions at runtime to guard against runtime errors (SecurityException).

**Install-time requests (Android 5.1.1 and below)**

If the device is running Android 5.1.1 (API level 22) or lower, or the app's targetSdkVersion is 22 or lower while running on any version of Android, the system automatically asks the user to grant all dangerous permissions for your app at install-time as shown in figure 2.

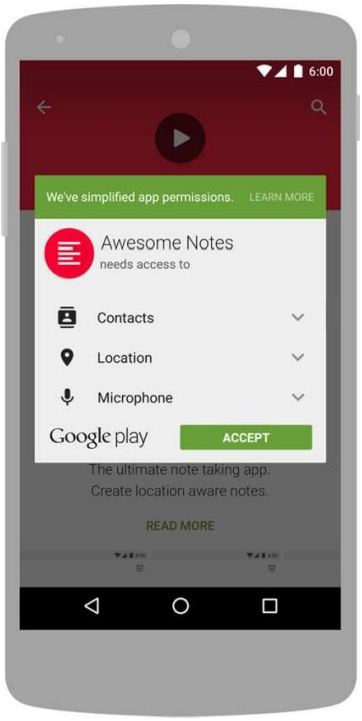



**Figure-59**

If the user clicks Accept, all permissions the app requests are granted. If the user denies the permissions request, the system cancels the installation of the app.

If an app update includes the need for additional permissions the user is prompted to accept those new permissions before updating the app.

## Permissions for optional hardware features

Access to some hardware features such as Bluetooth or the camera requires app permission. However, not all Android devices actually have these hardware features. So if your app requests the CAMERA permission, it's important that you also include the <uses-feature> tag in your manifest to declare whether or not this feature is actually required. For example:

<uses-feature android:name="android.hardware.camera" android:required="false" />

If you declare android:required="false" for the feature, then Google Play allows your app to be installed on devices that don't have the feature. You then must check if the current device has the feature at runtime by calling

PackageManager.hasSystemFeature(), and gracefully disable that feature if it's not available.

If you don't provide the <uses-feature> tag, then when Google Play sees that your app requests the corresponding permission, it assumes your app requires this feature. So it filters your app from devices without the feature, as if you declared android:required="true" in the <uses-feature> tag.

## Custom App Permission

Permissions aren't only for requesting system functionality. Services provided by apps can enforce custom permissions to restrict who can use them.

**Activity permission enforcement**

Permissions applied using the android:permission attribute to the <activity> tag in the manifest restrict who can start that Activity. The permission is checked during Context.startActivity() and Activity.startActivityForResult(). If the caller doesn't have the required permission then SecurityException is thrown from the call.

**Service permission enforcement**

Permissions applied using the android:permission attribute to the <service> tag in the manifest restrict who can start or bind to the associated Service. The permission is checked during Context.startService(), Context.stopService() and Context.bindService(). If the caller doesn't have the required permission then SecurityException is thrown from the call.

**Broadcast permission enforcement**

Permissions applied using the android:permission attribute to the <receiver> tag restrict who can send broadcasts to the associated BroadcastReceiver. The permission is checked after Context.sendBroadcast() returns, as the system tries to deliver the submitted broadcast to the given receiver. As a result, a permission failure doesn't result in an exception being thrown back to the caller; it just doesn't deliver the Intent.

In the same way, a permission can be supplied to Context.registerReceiver() to control who can broadcast to a programmatically registered receiver. Going the other way, a permission can be supplied when calling Context.sendBroadcast() to restrict which broadcast receivers are allowed to receive the broadcast.

Note that both a receiver and a broadcaster can require permission. When this happens, both permission checks must pass for the intent to be delivered to the associated target.

**Content Provider permission enforcement**

Permissions applied using the android:permission attribute to the <provider> tag restrict who can access the data in a ContentProvider. Unlike the other components, there are two separate permission attributes you can set: android:readPermission restricts who can read from the provider, and android:writePermission restricts who can write to it. Note that if a provider is protected with both a read and write permission, holding only the write permission doesn't mean you can read from a provider.

The permissions are checked when you first retrieve a provider and as you perform operations on the provider.

Using ContentResolver.query() requires holding the read permission;

using ContentResolver.insert(), ContentResolver.update(), ContentResolver.delete() requires the write permission. In all of these cases, not holding the required permission results in a SecurityException being thrown from the call.

**URI permissions**

The standard permission system described so far is often not sufficient when used with content providers. A content provider may want to protect itself with read and write permissions, while its direct clients also need to hand specific URIs to other apps for them to operate on.

A typical example is attachments in a email app. Access to the emails should be protected by permissions, since this is sensitive user data. However, if a URI to an image attachment is given to an image viewer, that image viewer no longer has permission to open the attachment since it has no reason to hold a permission to access all email.

The solution to this problem is per-URI permissions: when starting an activity or returning a result to an activity, the caller can set Intent.FLAG_GRANT_READ_URI_PERMISSION and/or Intent.FLAG_GRANT_WRITE_URI_PERMISSION. This grants the receiving activity permission access the specific data URI in the intent, regardless of whether it has any permission to access data in the content provider corresponding to the intent.

This mechanism allows a common capability-style model where user interaction (such as opening an attachment or selecting a contact from a list) drives ad-hoc granting of fine-grained permission. This can be a key facility for reducing the permissions needed by apps to only those directly related to their behavior.

To build the most secure implementation that makes other apps accountable for their actions within yor app, you should use fine-grained permissions in this manner and declare your app's support for it with the android:grantUriPermissions attribute or <grant-uri-permissions> tag.

**Other permission enforcement**

Arbitrarily fine-grained permissions can be enforced at any call into a service. This is accomplished with the Context.checkCallingPermission() method. Call with a desired permission string and it returns an integer indicating whether that permission has been granted to the current calling process. Note that this can only be used when you are executing a call coming in from another process, usually through an IDL interface published from a service or in some other way given to another process.

There are a number of other useful ways to check permissions. If you have the process ID (PID) of another process, you can use the Context.checkPermission() method to check a permission against that PID. If you have the package name of

another app, you can use the PackageManager.checkPermission() method to find out whether that particular package has been granted a specific permission.

## Permission Protection levels

Permissions are divided into several protection levels. The protection level affects whether runtime permission requests are required.

There are three protection levels that affect third-party apps: normal, signature, and dangerous permissions.

**Normal permissions**

Normal permissions cover areas where your app needs to access data or resources outside the app's sandbox, but where there's very little risk to the user's privacy or the operation of other apps. For example, permission to set the time zone is a normal permission.

If an app declares in its manifest that it needs a normal permission, the system automatically grants the app that permission at install time. The system doesn't prompt the user to grant normal permissions, and users cannot revoke these permissions.

As of Android 9 (API level 28), the following permissions are classified as PROTECTION_NORMAL:

ACCESS_LOCATION_EXTRA_COMMANDS
ACCESS_NETWORK_STATE
ACCESS_NOTIFICATION_POLICY
ACCESS_WIFI_STATE
BLUETOOTH
BLUETOOTH_ADMIN
BROADCAST_STICKY
CHANGE_NETWORK_STATE
CHANGE_WIFI_MULTICAST_STATE
CHANGE_WIFI_STATE
DISABLE_KEYGUARD
EXPAND_STATUS_BAR
FOREGROUND_SERVICE
GET_PACKAGE_SIZE
INSTALL_SHORTCUT
INTERNET
KILL_BACKGROUND_PROCESSES
MANAGE_OWN_CALLS
MODIFY_AUDIO_SETTINGS
NFC
READ_SYNC_SETTINGS
READ_SYNC_STATS
RECEIVE_BOOT_COMPLETED
REORDER_TASKS
REQUEST_DELETE_PACKAGES
SET_ALARM
SET_WALLPAPER
SET_WALLPAPER_HINTS
TRANSMIT_IR
USE_FINGERPRINT
VIBRATE
WAKE_LOCK
WRITE_SYNC_SETTINGS

**Signature permissions**

The system grants these app permissions at install time, but only when the app that attempts to use permission is signed by the same certificate as the app that defines the permission.

As of Android 8.1 (API level 27), the following permissions that third-party apps can use are classified as PROTECTION_SIGNATURE:

BIND_ACCESSIBILITY_SERVICE
BIND_AUTOFILL_SERVICE
BIND_CARRIER_SERVICES
BIND_CHOOSER_TARGET_SERVICE
BIND_CONDITION_PROVIDER_SERVICE
BIND_DEVICE_ADMIN
BIND_DREAM_SERVICE
BIND_INCALL_SERVICE
BIND_INPUT_METHOD
BIND_MIDI_DEVICE_SERVICE
BIND_NFC_SERVICE
BIND_NOTIFICATION_LISTENER_SERVICE
BIND_PRINT_SERVICE
BIND_SCREENING_SERVICE
BIND_TELECOM_CONNECTION_SERVICE
BIND_TEXT_SERVICE
BIND_TV_INPUT
BIND_VISUAL_VOICEMAIL_SERVICE
BIND_VOICE_INTERACTION
BIND_VPN_SERVICE
BIND_VR_LISTENER_SERVICE
BIND_WALLPAPER
CLEAR_APP_CACHE
MANAGE_DOCUMENTS
READ_VOICEMAIL
REQUEST_INSTALL_PACKAGES
SYSTEM_ALERT_WINDOW
WRITE_SETTINGS
WRITE_VOICEM

### Dangerous permissions

Dangerous permissions cover areas where the app wants data or resources that involve the user's private information, or could potentially affect the user's stored data or the operation of other apps. For example, the ability to read the user's contacts is a dangerous permission. If an app declares that it needs a dangerous permission, the user has to explicitly grant the permission to the app. Until the user

approves the permission, your app cannot provide functionality that depends on that permission.

To use a dangerous permission, your app must prompt the user to grant permission at runtime. For a list of dangerous permissions, see table 16 below.

| Permission Group | Permissions |
|---|---|
| **CALENDAR** | READ_CALENDAR<br>WRITE_CALENDAR |
| **CALL_LOG** | READ_CALL_LOG<br>WRITE_CALL_LOG<br>PROCESS_OUTGOING_CALLS |
| **CAMERA** | CAMERA |
| **CONTACTS** | READ_CONTACTS<br>WRITE_CONTACTS<br>GET_ACCOUNTS |
| **LOCATION** | ACCESS_FINE_LOCATION<br>ACCESS_COARSE_LOCATION |
| **MICROPHONE** | RECORD_AUDIO |
| **PHONE** | READ_PHONE_STATE<br>READ_PHONE_NUMBERS<br>CALL_PHONE<br>ANSWER_PHONE_CALLS<br>ADD_VOICEMAIL<br>USE_SIP |
| **SENSORS** | BODY_SENSORS |
| **SMS** | SEND_SMS<br>RECEIVE_SMS<br>READ_SMS<br>RECEIVE_WAP_PUSH<br>RECEIVE_MMS |
| **STORAGE** | READ_EXTERNAL_STORAGE<br>WRITE_EXTERNAL_STORAGE |

Table-16: Dangerous permissions and permission groups.

### Special permissions

There are a couple of permissions that don't behave like normal and dangerous permissions. SYSTEM_ALERT_WINDOW and WRITE_SETTINGS are particularly sensitive, so most apps should not use them. If an app needs one of these permissions, it must declare the permission in the manifest, and send an intent requesting the user's authorization. The system responds to the intent by showing a detailed management screen to the user.

## How to View app's permissions

You can view all the permissions currently defined in the system using the Settings app and the shell command adb shell pm list permissions. To use the Settings app, go to Settings > Apps. Pick an app and scroll down to see the permissions that the app uses. For developers, the adb '-s' option displays the permissions in a form similar to how the user sees them:

```
$ adb shell pm list permissions -s

All Permissions:

Network communication: view Wi-Fi state, create Bluetooth connections, full internet access, view network state

Your location: access extra location provider commands, fine (GPS) location, mock location sources for testing, coarse (network-based) location

Services that cost you money: send SMS messages, directly call phone numbers

...
```

You can also use the adb -g option to grant all permissions automatically when installing an app on an emulator or test device:

```
$ adb shell install -g MyApp.apk
```

- 
- 
- 

# Introduction

Most Android applications inevitably need some form of user interface. In this unit, we will discuss the user interface elements available within the Android Software Development Kit (SDK). Some of these elements display information to the user, whereas others gather information from the user.

You learn how to use a variety of different components and controls to build a screen and how your application can listen for various actions performed by the user. Finally, you learn how to style controls and apply themes to entire screens.

# Introduction to Views, Controls and Layout

Before we go any further, we need to define a few terms. This gives you a better understanding of certain capabilities provided by the Android SDK before they are fully introduced. First, let's talk about the View class.

## Introduction to Android Views

This class represents the basic building block for user interface components. A View occupies a rectangular area on the screen and is responsible for drawing and event handling. View is the base class for widgets, which are used to create interactive UI components (buttons, text fields, etc.). The ViewGroup subclass is the base class for

layouts, which are invisible containers that hold other Views (or other ViewGroups) and define their layout properties.

All of the views in a window are arranged in a single tree. You can add views either from code or by specifying a tree of views in one or more XML layout files. There are many specialized subclasses of views that act as controls or are capable of displaying text, images, or other content.

Once you have created a tree of views, there are typically a few types of common operations you may wish to perform:

**Set properties:** for example setting the text of a TextView. The available properties and the methods that set them will vary among the different subclasses of views. Note that properties that are known at build time can be set in the XML layout files.
Set focus: The framework will handle moving focus in response to user input. To force focus to a specific view, call requestFocus().

**Set up listeners:** Views allow clients to set listeners that will be notified when something interesting happens to the view. For example, all views will let you set a listener to be notified when the view gains or loses focus. You can register such a listener using setOnFocusChangeListener(android.view.View.OnFocusChangeListener).
Other view subclasses offer more specialized listeners. For example, a Button exposes a listener to notify clients when the button is clicked.

**Set visibility:** You can hide or show views using setVisibility(int).

## Introduction to Android Controls

The Android SDK contains a Java package named android.widget. When we refer to controls, we are typically referring to a class within this package.The Android SDK includes classes to draw most common objects, including ImageView, FrameLayout, EditText, and Button classes. All controls are typically derived from the View class. We cover many of these basic controls in detail.

## Introduction to Android Layout

One special type of control found within the android.widget package is called a layout. A layout control is still a View object, but it doesn't actually draw anything specific on the screen. Instead, it is a parent container for organizing other controls (children). Layout controls determine how and where on the screen child controls are drawn. Each type of layout control draws its children using particular rules. For instance, the LinearLayout control draws its child controls in a single horizontal row or a single vertical column. Similarly, a TableLayout control displays each child control in tabular format (in cells within specific rows and columns).

By necessity, we use some of the layout View objects within this unit to illustrate how to use the controls previously mentioned. However, we don't go into the details of the various layout types available as part of the Android SDK until the next unit. We will lean in more details about layout in next unit.

## TextView

TextView is a user interface element that displays text to the user. Following table shows important XML Attributes of TextView control.

| Attribute | Description |
|---|---|
| id | id is an attribute used to uniquely identify a text view |
| gravity | The gravity attribute is an optional attribute which is used to control the alignment of the text like left, right, center, top, bottom, center_vertical, center_horizontal etc. |
| text | text attribute is used to set the text in a text view. |
| textColor | textColor attribute is used to set the text color of a text view. Color value is in the form of "#argb", "#rgb", "#rrggbb", or "#aarrggbb" |
| textSize | textSize attribute is used to set the size of text of a text view. We can set the text size in sp(scale independent pixel) or dp(density pixel). |
| textStyle | textStyle attribute is used to set the text style of a text view. The possible text styles are bold, italic and normal. |

| | |
|---|---|
| background | background attribute is used to set the background of a text view. We can set a color or a drawable in the background of a text view |
| padding | padding attribute is used to set the padding from left, right, top or bottom. |

**Table-18**

The following code sample shows a typical use, with an XML layout and code to modify the contents of the text view:

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
  <TextView
      android:id="@+id/text_view_id"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="This is TextView"
      android:layout_centerInParent="true"
      android:textSize="35sp"
      android:padding="15dp"
      android:textColor="#aaa"
      android:background="#fff"/>
</LinearLayout>
```

This code sample demonstrates how to modify the contents of the text view defined in the previous XML layout:

```
public class MainActivity extends Activity {

  protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      final TextView helloTextView = (TextView) findViewById(R.id.text_view_id);
```

```
        helloTextView.setText(R.string.user_greeting);
    }
}
```

To display this TextView on the screen, all your Activity needs to do is call the setContentView() method with the layout resource identifier in which you defined in the preceding XML shown.

You can change the text displayed programmatically by calling the setText() method on the TextView object. Retrieving the text is done with the getText() method. To customize the appearance of TextView we can use Styles and Themes.

## EditText

EditText is a user interface element for entering and modifying text. Following table shows important XML Attributes of EditText control.

| Attribute | Description |
|---|---|
| id | This is an attribute used to uniquely identify an edit text |
| gravity | The gravity attribute is an optional attribute which is used to control the alignment of the text like left, right, center, top, bottom, center_vertical, center_horizontal etc. |
| text | This attribute is used to set the text in a text view. |
| hint | It is an attribute used to set the hint i.e. what you want user to enter in this edit text. Whenever user start to type in edit text the hint will automatically disappear. |
| lines | Defines how many lines tall the input box is. If this is not set, the entry field grows as the user enters text. |
| textColorHint | It is an attribute used to set the color of displayed hint. |
| textColor | This attribute is used to set the text color of a edit text. Color value is in the form of “#argb”, “#rgb”, “#rrggbb”, or “#aarrggbb” |
| textSize | This attribute is used to set the size of text of a edit text. We can set the text size in sp(scale independent pixel) or dp(density pixel). |

| | |
|---|---|
| textStyle | This attribute is used to set the text style of a edit text. The possible text styles are bold, italic and normal. |
| background | This attribute is used to set the background of a edit text. We can set a color or a drawable in the background of a edit text |
| padding | Padding attribute is used to set the padding from left, right, top or bottom. |

**Table-19**

Following layout code shows a basic EditText element.

```
<EditText
android:id="@+id/txtName"
android:layout_height="wrap_content"
android:hint="Full Name"
android:lines="4"
android:layout_width="fill_parent" />
```

The EditText object is essentially an editable TextView. You can read text from it in by using the getText() method. You can also set initial text to draw in the text entry area using the setText() method. You can also highlight a portion of the text from code by call to setSelection() method and a call to selectAll() method highlights the entire text entry field.

By default, the user can perform a long press to bring up a context menu. This provides to the user some basic copy, cut, and paste operations as well as the ability to change the input method and add a word to the user's dictionary of frequently used words. You can set the editable attribute to false, so the user cannot edit the text in the field but can still copy text out of it using a long press.

## AutoCompleteTextView

In Android, AutoCompleteTextView is a view i.e. similar to EditText, except that it displays a list of completion suggestions automatically while the user is typing. A list

of suggestions is displayed in drop down menu from which user can choose an item which actually replace the content of EditBox with that.
It is a subclass of EditText class so we can inherit all the properties of EditText in a AutoCompleteTextView.

Following layout code shows a basic AutoCompleteTextView element.

```
<AutoCompleteTextView
android:id="@+id/ac"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text=" Auto Suggestions EditText"/>
```

To display the Array content in an AutoCompleteTextView we need to implement Adapter. In AutoCompleteTextView we mainly display text values so we use Array Adapter for that. ArrayAdapter is used when we need list of single type of items which is backed by an Array. For example, list of phone contacts, countries or names.

```
ArrayAdapter(Context context, int resource, int textViewResourceId, T[] objects)
AutoCompleteTextView ac = (AutoCompleteTextView) findViewById(R.id.ac);
```

Following code retrieve the value from a AutoCompleteTextView in Java class.

```
String v = ac.getText().toString();
```

## Spinner

In Android, Spinner provides a quick way to select one value from a set of values. It is similar to dropdown list in other programming language. In a default state, a spinner shows its currently selected value. It provides an easy way to select a value from a known set. Following table shows important XML Attributes of spinner control.

<table>
<tr><th>Attribute</th><th>Description</th></tr>
<tr><td>dropDownHorizontalOffset</td><td>Amount of pixels by which the drop down should be offset horizontally.</td></tr>
<tr><td>dropDownSelector</td><td>List selector to use for spinnerMode="dropdown" display.<br><br>May be a reference to another resource, in the form "@[+][package:]type/name" or a theme attribute in the form "?[package:]type/name".<br><br>May be a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".</td></tr>
<tr><td>dropDownVerticalOffset</td><td>Amount of pixels by which the drop down should be offset vertically.</td></tr>
<tr><td>dropDownWidth</td><td>Width of the dropdown in spinnerMode="dropdown".</td></tr>
<tr><td>gravity</td><td>Gravity setting for positioning the currently selected item.</td></tr>
<tr><td>popupBackground</td><td>Background drawable to use for the dropdown in spinnerMode="dropdown".</td></tr>
<tr><td>prompt</td><td>The prompt to display when the spinner's dialog is shown.</td></tr>
<tr><td>spinnerMode</td><td>Display mode for spinner options. Must be one of the following constant values.<br><table><tr><th>Constant</th><th>Value</th><th>Description</th></tr></table></td></tr>
</table>

| | | | |
|---|---|---|---|
| | dialog | 0 | Spinner options will be presented to the user as a dialog window. |
| | dropdown | 1 | Spinner options will be presented to the user as an inline dropdown anchored to the spinner widget itself. |

**Table-20**

As with the auto-complete method, the possible choices for a spinner can come from an Adapter. You can also set the available choices in the layout definition by using the entries attribute with an array resource. Following is an XML layout for showing spinner

```
<Spinner
android:id="@+id/Spinner01"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:entries="@array/colors"
android:prompt="@string/spin_prompt" />
```

This places a Spinner control on the screen. When the user selects it, a pop-up shows the prompt text followed by a list of the possible choices. This list allows only a single item to be selected at a time, and when one is selected, the pop-up goes away.

First, the entries attribute is set to the values that shows by assigning it to an array resource, referred to here as @array/colors.

## Populate the Spinner with User Choices

The choices you provide for the spinner can come from any source, but must be provided through a SpinnerAdapter, such as an ArrayAdapter if the choices are available in an array or a CursorAdapter if the choices are available from a database query.

For instance, if the available choices for your spinner are pre-determined, you can provide them with a string array defined in a string resource file:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string-array name="planets_array">
      <item>Mercury</item>
      <item>Venus</item>
      <item>Earth</item>
      <item>Mars</item>
      <item>Jupiter</item>
      <item>Saturn</item>
      <item>Uranus</item>
      <item>Neptune</item>
   </string-array>
</resources>
```

With an array such as this one, you can use the following code in your Activity or Fragment to supply the spinner with the array using an instance of ArrayAdapter:

```
Spinner spinner = (Spinner) findViewById(R.id.spinner);

// Create an ArrayAdapter using the string array and a default spinner layout
ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this,
R.array.planets_array, android.R.layout.simple_spinner_item);

// Specify the layout to use when the list of choices appears
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_ite
m);

// Apply the adapter to the spinner
spinner.setAdapter(adapter);
```

The createFromResource() method allows you to create an ArrayAdapter from the string array. The third argument for this method is a layout resource that defines how the selected choice appears in the spinner control. The simple_spinner_item layout is provided by the platform and is the default layout you should use unless you'd like to define your own layout for the spinner's appearance.

You should then call setDropDownViewResource(int) to specify the layout the adapter should use to display the list of spinner choices.

Call setAdapter() to apply the adapter to your Spinner.

## Responding to User Selections

When the user selects an item from the drop-down, the Spinner object receives an on-item-selected event.

To define the selection event handler for a spinner, implement the AdapterView.OnItemSelectedListener interface and the corresponding onItemSelected() callback method. For example, here's an implementation of the interface in an Activity:

```
public class SpinnerActivity extends Activity implements OnItemSelectedListener {
    ...

    public void onItemSelected(AdapterView<?> parent, View view,
            int pos, long id) {
        // An item was selected. You can retrieve the selected item using
        // parent.getItemAtPosition(pos)
    }

    public void onNothingSelected(AdapterView<?> parent) {
        // Another interface callback
    }
}
```

The AdapterView.OnItemSelectedListener requires the onItemSelected() and onNothingSelected() callback methods.

Then you need to specify the interface implementation by calling setOnItemSelectedListener():

```
Spinner spinner = (Spinner) findViewById(R.id.spinner);
spinner.setOnItemSelectedListener(this);
```

If you implement the AdapterView.OnItemSelectedListener interface with your Activity or Fragment (such as in the example above), you can pass this as the interface instance.

## Button

A user interface element the user can tap or click to perform an action. To display a button in an activity, add a button to the activity's layout XML file:

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

To specify an action when the button is pressed, set a click listener on the button object in the corresponding activity code:


**Figure-61**

```
public class MyActivity extends Activity {
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.content_layout_id);

        final Button button = findViewById(R.id.button_id);
        button.setOnClickListener(new View.OnClickListener() {
            public void onClick(View v) {
                // Code here executes on main thread after user presses button
            }
        });
    }
}
```

The above snippet creates an instance of View.OnClickListener and wires the listener to the button using setOnClickListener(View.OnClickListener). As a result, the system executes the code you write in onClick(View) after the user presses the button.

Every button is styled using the system's default button background, which is often different from one version of the platform to another. If you are not satisfied with the default button style, you can customize it.

## Checkbox

A checkbox is a specific type of two-states button that can be either checked or unchecked.

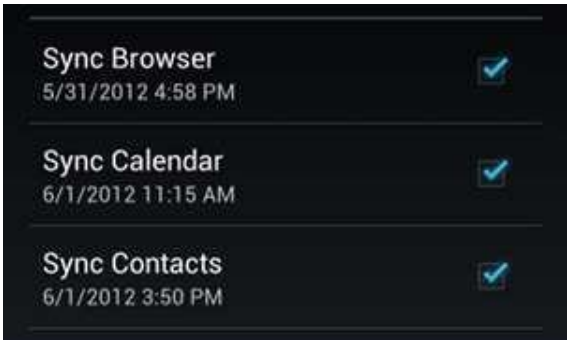



**Figure-62**

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
```

```
    }
}
```

## Radio Button

Radio buttons allow the user to select one option from a set. You should use radio buttons for optional sets that are mutually exclusive if you think that the user needs to see all available options side-by-side. If it's not necessary to show all options side-by-side, use a spinner instead.

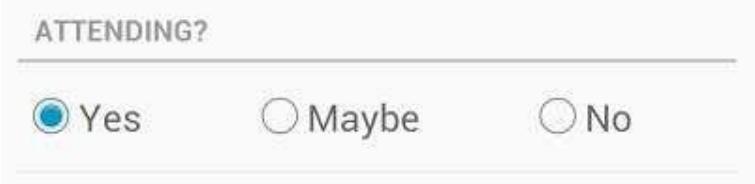



**Figure-63**

To create each radio button option, create a RadioButton in your layout. However, because radio buttons are mutually exclusive, you must group them together inside a RadioGroup. By grouping them together, the system ensures that only one radio button can be selected at a time.

### Responding to Click Events

When the user selects one of the radio buttons, the corresponding RadioButton object receives an on-click event.

To define the click event handler for a button, add the android:onClick attribute to the <RadioButton> 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 RadioButton objects:

```
<?xml version="1.0" encoding="utf-8"?>
<RadioGroup  xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <RadioButton android:id="@+id/radio_pirates"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/pirates"
        android:onClick="onRadioButtonClicked"/>
    <RadioButton android:id="@+id/radio_ninjas"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/ninjas"
        android:onClick="onRadioButtonClicked"/>
</RadioGroup>
```

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

```
public void onRadioButtonClicked(View view) {
    // Is the button now checked?
    boolean checked = ((RadioButton) view).isChecked();

    // Check which radio button was clicked
    switch(view.getId()) {
        case R.id.radio_pirates:
            if (checked)
                // Pirates are the best
            break;
        case R.id.radio_ninjas:
            if (checked)
                // Ninjas rule
            break;
    }
}
```

# Pickers

Android provides controls for the user to pick a time or pick a date as ready-to-use dialogs. Each picker provides controls for selecting each part of the time (hour, minute, AM/PM) or date (month, day, year). Using these pickers helps ensure that your users can pick a time or date that is valid, formatted correctly, and adjusted to the user's locale.

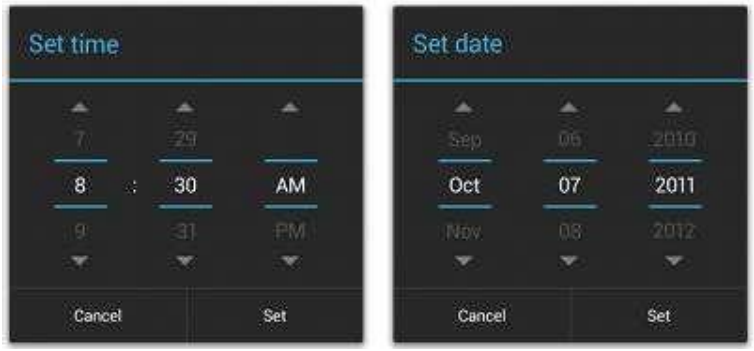



**Figure-64**

It is recommended that you use DialogFragment to host each time or date picker. The DialogFragment manages the dialog lifecycle for you and allows you to display the pickers in different layout configurations, such as in a basic dialog on handsets or as an embedded part of the layout on large screens.

## Creating a Time Picker

To display a TimePickerDialog using DialogFragment, you need to define a fragment class that extends DialogFragment and return a TimePickerDialog from the fragment's onCreateDialog() method.

## Extending DialogFragment for a time picker

To define a DialogFragment for a TimePickerDialog, you must:

- Define the onCreateDialog() method to return an instance of TimePickerDialog
- Implement the TimePickerDialog.OnTimeSetListener interface to receive a callback when the user sets the time.

Here's an example:

```
public static class TimePickerFragment extends DialogFragment
                        implements TimePickerDialog.OnTimeSetListener {

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Use the current time as the default values for the picker
        final Calendar c = Calendar.getInstance();
        int hour = c.get(Calendar.HOUR_OF_DAY);
        int minute = c.get(Calendar.MINUTE);

        // Create a new instance of TimePickerDialog and return it
        return new TimePickerDialog(getActivity(), this, hour, minute,
                DateFormat.is24HourFormat(getActivity()));
    }

    public void onTimeSet(TimePicker view, int hourOfDay, int minute) {
        // Do something with the time chosen by the user
    }
}
```

## Showing the time picker

Once you've defined a DialogFragment like the one shown above, you can display the time picker by creating an instance of the DialogFragment and calling show().

For example, here's a button that, when clicked, calls a method to show the dialog:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

```
    android:text="@string/pick_time"
    android:onClick="showTimePickerDialog" />
```

When the user clicks this button, the system calls the following method:

```
public void showTimePickerDialog(View v) {
    DialogFragment newFragment = new TimePickerFragment();
    newFragment.show(getSupportFragmentManager(), "timePicker");
}
```

This method calls show() on a new instance of the DialogFragment defined above. The show() method requires an instance of FragmentManager and a unique tag name for the fragment.

## Creating a Date Picker

Creating a DatePickerDialog is just like creating a TimePickerDialog. The only difference is the dialog you create for the fragment.

To display a DatePickerDialog using DialogFragment, you need to define a fragment class that extends DialogFragment and return a DatePickerDialog from the fragment's onCreateDialog() method.

## Extending DialogFragment for a date picker

To define a DialogFragment for a DatePickerDialog, you must:

- Define the onCreateDialog() method to return an instance of DatePickerDialog
- Implement the DatePickerDialog.OnDateSetListener interface to receive a callback when the user sets the date.

Here's an example:

```
public static class DatePickerFragment extends DialogFragment
                    implements DatePickerDialog.OnDateSetListener {

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        // Use the current date as the default date in the picker
        final Calendar c = Calendar.getInstance();
        int year = c.get(Calendar.YEAR);
        int month = c.get(Calendar.MONTH);
        int day = c.get(Calendar.DAY_OF_MONTH);

        // Create a new instance of DatePickerDialog and return it
        return new DatePickerDialog(getActivity(), this, year, month, day);
    }

    public void onDateSet(DatePicker view, int year, int month, int day) {
        // Do something with the date chosen by the user
    }
}
```

### Showing the date picker

Once you've defined a DialogFragment like the one shown above, you can display the date picker by creating an instance of the DialogFragment and calling show().

For example, here's a button that, when clicked, calls a method to show the dialog:

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/pick_date"
    android:onClick="showDatePickerDialog" />
```

When the user clicks this button, the system calls the following method:

```
public void showDatePickerDialog(View v) {
    DialogFragment newFragment = new DatePickerFragment();
    newFragment.show(getSupportFragmentManager(), "datePicker");
}
```

This method calls show() on a new instance of the DialogFragment defined above. The show() method requires an instance of Fragment Manager and a unique tag name for the fragment.

---

## Introduction

One special type of control found within the android.widget package is called a layout. A layout control is still a View object, but it doesn't actually draw anything specific on the screen. Instead, it is a parent container for organizing other controls (children). Layout controls determine how and where on the screen child controls are drawn. Each type of layout control draws its children using particular rules. For instance, the LinearLayout control draws its child controls in a single horizontal row or a single vertical column. Similarly, a TableLayout control displays each child control in tabular format (in cells within specific rows and columns).

Application user interfaces can be simple or complex, involving many different screens or only a few. Layouts and user interface controls can be defined as application resources or created programmatically at runtime.

## Creating Layouts Using XML Resources

Android provides a simple way to create layout files in XML as resources provided in the /res/layout project directory. This is the most common and convenient way to build Android user interfaces and is especially useful for defining static screen elements and control properties that you know in advance, and to set default attributes that you can modify programmatically at runtime.

You can configure almost any ViewGroup or View (or View subclass) attribute using the XML layout resource files.This method greatly simplifies the user interface design process, moving much of the static creation and layout of user interface controls, and basic definition of control attributes, to the XML, instead of littering the code. Developers reserve the ability to alter these layouts programmatically as necessary, but they can set all the defaults in the XML template. You will recognize the following as a simple layout file with a LinearLayout and a single TextView control.

```
<?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>
```

This block of XML shows a basic layout with a single TextView. The first line, which you might recognize from most XML files, is required.

Creating only an XML file, though, won't actually draw anything on the screen.A particular layout is usually associated with a particular Activity. In your default Android project, there is only one activity, which sets the main.xml layout by default. To associate the main.xml layout with the activity, use the method call setContentView() with the identifier of the main.xml layout.The ID of the layout matches the XML filename without the extension. In this case, the preceding example came from main.xml, so the identifier of this layout is simply main:

```
setContentView(R.layout.main);
```

## Creating Layouts Programmatically

You can create user interface components such as layouts at runtime programmatically, but for organization and maintainability, it's best that you leave this for the odd case rather than the norm. The main reason is because the creation of layouts programmatically is onerous and difficult to maintain, whereas the XML resource method is visual, more organized, and could be done by a separate designer with no Java skills.

The following example shows how to programmatically have an Activity instantiate a LinearLayout view and place two TextView objects within it. No resources whatsoever are used; actions are done at runtime instead.

```
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TextView text1 = new TextView(this);
        text1.setText("Hi there!");
        TextView text2 = new TextView(this);
        text2.setText("I'm second. I need to wrap.");
        text2.setTextSize((float) 60);
        LinearLayout ll = new LinearLayout(this);
        ll.setOrientation(LinearLayout.VERTICAL);
        ll.addView(text1);
        ll.addView(text2);
        setContentView(ll);
}
```

The onCreate() method is called when the Activity is created. The first thing this method does is some normal Activity housekeeping by calling the constructor for the base class. Next, two TextView controls are instantiated.The Text property of each TextView is set using the setText() method.All TextView attributes, such as TextSize, are set by making method calls on the TextView object. These actions perform the same function that you have in the past by setting the properties Text and TextSize

using the layout resource designer, except these properties are set at runtime instead of defined in the layout files compiled into your application package.

## Built in layouts

We talked a lot about the LinearLayout layout, but there are several other types of layouts. Each layout has a different purpose and order in which it displays its child View controls on the screen. Layouts are derived from android.view.ViewGroup.

The types of layouts built-in to the Android SDK framework include:

- FrameLayout
- LinearLayout
- RelativeLayout
- TableLayout

All layouts, regardless of their type, have basic layout attributes. Layout attributes apply to any child View within that layout. You can set layout attributes at runtime programmatically, but ideally you set them in the XML layout files using the following syntax: android:layout_attribute_name="value"

There are several layout attributes that all ViewGroup objects share.These include size attributes and margin attributes. You can find basic layout attributes in the ViewGroup.LayoutParams class.The margin attributes enable each child View within a layout to have padding on each side. Find these attributes in the ViewGroup.MarginLayoutParams classes. There are also a number of ViewGroup attributes for handling child View drawing bounds and animation settings.

## Frame Layout

FrameLayout 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.

Child views are drawn in a stack, with the most recently added child on top. The size of the FrameLayout is the size of its largest child (plus padding), visible or not (if the FrameLayout's parent permits). Views that are View.GONE are used for sizing only if setConsiderGoneChildrenWhenMeasuring() is set to true.

Following Table describes some of the important attributes specific to FrameLayout views.

| Attribute Name | Applies To | Description | Value |
|---|---|---|---|
| **android: foreground** | Parent view | Drawable to draw over the content | Drawable resource. |
| **android: foreground-Gravity** | Parent view | Gravity of foreground drawable. | One or more constants separated by "\|". The constants available are top, bottom, left, right, center_vertical, fill_vertical, center_horizontal, fill_horizontal, center, and fill. |
| **android: measureAll-Children** | Parent view | Restrict size of layout to all child views or just the child views set to VISIBLE (and not those set to INVISIBLE). | True or false. |
| **android: layout_ gravity** | Child view | A gravity constant that describes how to place the child View within the parent. | One or more constants separated by "\|". The constants available are top, bottom, left, right, center_vertical, fill_ vertical, center_horizontal, fill_horizontal, center, and fill. |

**Table-21**

An Example of FlowLayout is shown below

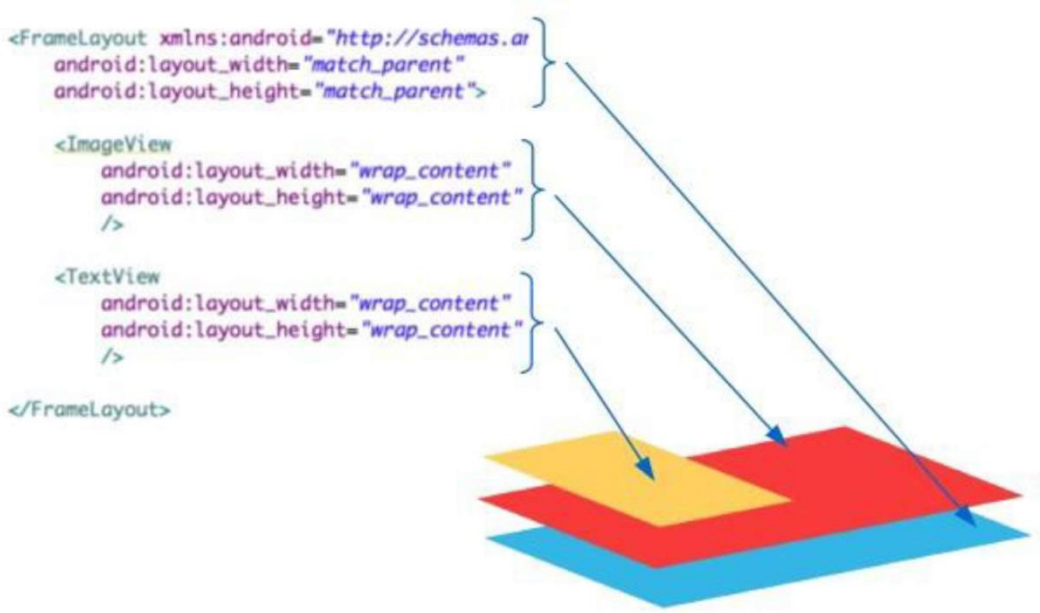



**Figure-65**

## LinearLayout

A LinearLayout view organizes its child View objects in a single row, shown in Figure below, or column, depending on whether its orientation attribute is set to horizontal or vertical. This is a very handy layout method for creating forms.

You can find the layout attributes available for LinearLayout child View objects in android.control.LinearLayout.LayoutParams. Following table describes some of the important attributes specific to LinearLayout views.

| Attribute Name | Applies To | Description | Value |
|---|---|---|---|
| **android: orientation** | Parent view | Layout is a single row (horizontal) or single column (vertical). | Horizontal or Vertical |
| **android: gravity** | Parent view | Gravity of child views within layout. | One or more constants separated by "\|". The constants available are top, bottom, left, right, center_vertical, fill_vertical, center_horizontal, fill_horizontal, center, and fill. |
| **android: layout_ gravity** | Child View | The gravity for a specific child view. Used for positioning of views. | One or more constants separated by "\|". The constants available are top, bottom, left, right, center_vertical, fill_vertical, center_horizontal, fill_horizontal, center, and fill. |
| **android: layout_ weight** | Child view | The weight for a specific child view. Used to provide ratio of screen space used within the parent control. | The sum of values across all child views in a parent view must equal 1. For example, one child control might have a value of .3 and another have a value of .7. |

**Table-22**

An Example of Linear Layout is shown below

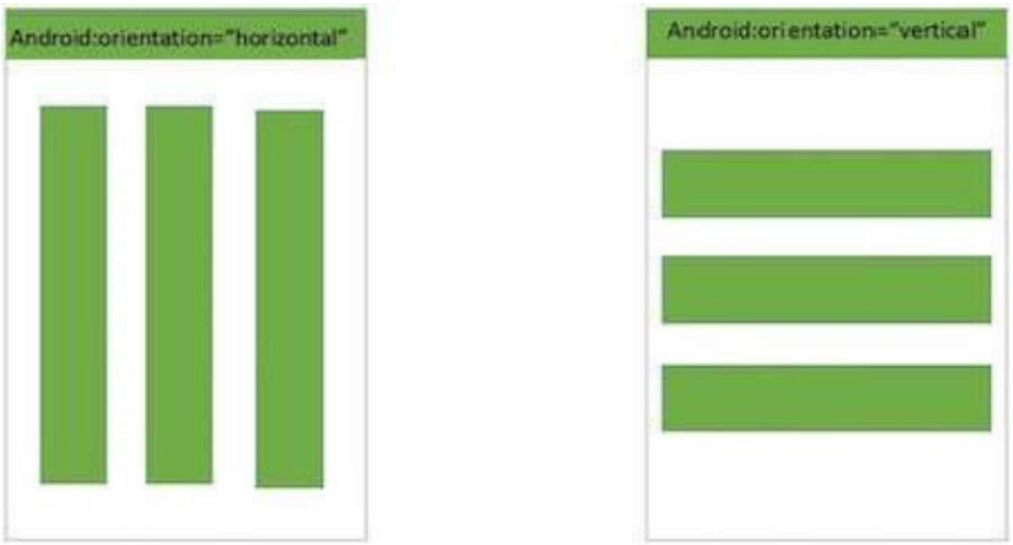



**Figure-66**

## RelativeLayout

The RelativeLayout view enables you to specify where the child view controls are in relation to each other. For instance, you can set a child View to be positioned "above" or "below" or "to the left of " or "to the right of " another View, referred to by its unique identifier. You can also align child View objects relative to one another or the parent layout edges. Combining RelativeLayout attributes can simplify creating interesting user interfaces without resorting to multiple layout groups to achieve a desired effect. You can find the layout attributes available for RelativeLayout child View objects in android.control.RelativeLayout.LayoutParams. Following Table describes some of the important attributes specific to RelativeLayout views.

| Attribute Name | Applies To | Description | Value |
|---|---|---|---|
| **android:gravity** | Parent view | Gravity of child views within layout. | One or more constants separated by "\|". The constants available are top, bottom, left, right, center_vertical, fill_vertical, center_horizontal, fill_horizontal, center, and fill. |
| **android: layout_centerInParent** | Child view | Centers child view horizontally and vertically within parent view. | True or False |
| **android: layout_centerHorizontal** | Child view | Centers child view horizontally within parent view | True or False |
| **android: layout_centerVertical** | Child view | Centers child view vertically within parent view. | True or False |
| **android: layout_alignParentTop** | Child view | Aligns child view with top edge of parent view | True or False |
| **android: layout_alignParentBottom** | Child view | Aligns child view with bottom edge of parent view. | True or False |

| Attribute Name | Applies To | Description | Value |
|---|---|---|---|
| **android: layout_alignParentLeft** | Child View | Aligns child view with left edge of parent view. | True or False |
| **android: layout_alignParentRight** | Child View | Aligns child view with right edge of parent view. | True or False |
| **android: layout_alignRight** | Child View | Aligns child view with right edge of another child view, specified by ID. | A view ID; for example, @id/Button1 |
| **android: layout_alignLeft** | Child View | Aligns child view with left edge of another child view, specified by ID. | A view ID; for example, @id/Button1 |
| **android: layout_alignTop** | Child View | | A view ID; for example, @id/Button1 |
| **android: layout_alignBottom** | Child View | | A view ID; for example, @id/Button1 |
| **android: layout_above** | Child View | Positions bottom edge of child view above another child view, specified by ID. | A view ID; for example, @id/Button1 |
| **android: layout_below** | Child View | Positions top edge of child view below another child view, specified by ID. | A view ID; for example, @id/Button1 |
| **android: layout_toLeftOf** | Child View | Positions right edge of child view to the left of another child view, specified by ID. | A view ID; for example, @id/Button1 |
| **android: layout_toRightOf** | Child View | Positions left edge of child view to the right of another child view, specified by ID. | A view ID; for example, @id/Button1 |

**Table-23**

Following figure shows how each of the button controls is relative to each other.


**Figure-67**

Here's an example of an XML layout resource with a RelativeLayout and two child View objects, a Button object aligned relative to its parent, and an ImageView aligned and positioned relative to the Button (and the parent):

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android= "http://schemas.android.com/apk/res/android"
        android:id="@+id/RelativeLayout01"
        android:layout_height="fill_parent"
        android:layout_width="fill_parent">
        <Button android:id="@+id/ButtonCenter"
                android:text="Center"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_centerInParent="true" />
        <ImageView android:id="@+id/ImageView01"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_above="@id/ButtonCenter"
                android:layout_centerHorizontal="true"
                android:src="@drawable/arrow" />
</RelativeLayout>
```

## TableLayout

A TableLayout view organizes children into rows, as shown in following Figure-68. You add individual View objects within each row of the table using a TableRow layout View (which is basically a horizontally oriented LinearLayout) for each row of the table. Each column of the TableRow can contain one View (or layout with child View objects).You place View items added to a TableRow in columns in the order they are added. You can specify the column number (zero-based) to skip columns as necessary (the bottom row shown in above figure demonstrates this); otherwise, the View object is put in the next column to the right. Columns scale to the size of the largest View of that column. You can also include normal View objects instead of TableRow elements, if you want the View to take up an entire row.

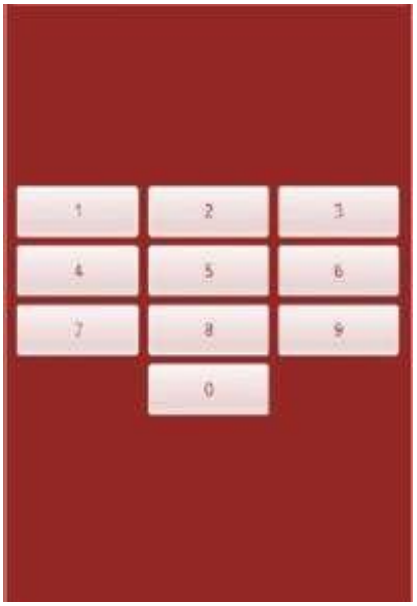



**Figure-68**

You can find the layout attributes available for TableLayout child View objects in android.control.TableLayout.LayoutParams.You can find the layout attributes available for TableRow child View objects in android.control.TableRow.LayoutParams. Following Table describes some of the important attributes specific to TableLayout View objects.

| Attribute Name | Applies To | Description | Value |
|---|---|---|---|
| **android: collapseColumns** | TableLayout | A comma-delimited list of column indices to collapse (0-based) | String or string resource. For example, "0,1,3,5" |
| **android: shrinkColumns** | TableLayout | A comma-delimited list of column indices to shrink (0-based) | String or string resource. Use "*" for all columns. For example, "0,1,3,5" |
| **andriod: stretchColumns** | TableLayout | A comma-delimited list of column indices to stretch (0-based) | String or string resource. Use "*" for all columns. For example, "0,1,3,5" |
| **android: layout_column** | TableRow child view | Index of column this child view should be displayed in (0-based) | Integer or integer resource. For example, 1 |
| **android: layout_span** | TableRow child view | Number of columns this child view should span across | Integer or integer resource greater than or equal to 1. For example, 3 |

**Table-24**

Here's an example of an XML layout resource with a TableLayout with two rows (two TableRow child objects).The TableLayout is set to stretch the columns to the size of the screen width. The first TableRow has three columns; each cell has a Button object. The second TableRow puts only one Button view into the second column explicitly:

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/TableLayout01"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:stretchColumns="*">
        <TableRow android:id="@+id/TableRow01">
```

```
            <Button android:id="@+id/ButtonLeft"
                    android:text="Left Door" />
            <Button android:id="@+id/ButtonMiddle"
                    android:text="Middle Door" />
            <Button android:id="@+id/ButtonRight"
                     android:text="Right Door" />
        </TableRow>
        <TableRow android:id="@+id/TableRow02">
            <Button android:id="@+id/ButtonBack"
                    android:text="Go Back"
                    android:layout_column="1" />
        </TableRow>
</TableLayout>
```

## Using Data-Driven Containers

Some of the View container controls are designed for displaying repetitive View objects in a particular way. Examples of this type of View container control include ListView, GridView, and GalleryView:

- ListView: Contains a vertically scrolling, horizontally filled list of View objects, each of which typically contains a row of data; the user can choose an item to perform some action upon.
- GridView: Contains a grid of View objects, with a specific number of columns; this container is often used with image icons; the user can choose an item to perform some action upon.
- GalleryView: Contains a horizontally scrolling list of View objects, also often used with image icons; the user can select an item to perform some action upon.

These containers are all types of AdapterView controls. An AdapterView control contains a set of child View controls to display data from some data source. An Adapter generates these child View controls from a data source. As this is an important part of all these container controls, we talk about the Adapter objects first.

In this section, you learn how to bind data to View objects using Adapter objects. In the Android SDK, an Adapter reads data from some data source and provides a View object based on some rules, depending on the type of Adapter used. This View is used to populate the child View objects of a particular AdapterView.

The most common Adapter classes are the CursorAdapter and the ArrayAdapter. The CursorAdapter gathers data from a Cursor, whereas the ArrayAdapter gathers data from an array. A CursorAdapter is a good choice to use when using data from a database. The ArrayAdapter is a good choice to use when there is only a single column of data or when the data comes from a resource array.

There are some common elements to know about Adapter objects. When creating an Adapter, you provide a layout identifier.This layout is the template for filling in each row of data. The template you create contains identifiers for particular controls that the Adapter assigns data to.A simple layout can contain as little as a single TextView control.

When making an Adapter, refer to both the layout resource and the identifier of the TextView control. The Android SDK provides some common layout resources for use in your application.

**How to Use the Adapter**

An ArrayAdapter binds each element of the array to a single View object within the layout resource. Here is an example of creating an ArrayAdapter:

```
private String[] items = { “Item 1”, “Item 2”, “Item 3” };
ArrayAdapter adapt = new ArrayAdapter<String> (this, R.layout.textview, items);
```

In this example, we have a String array called items. This is the array used by the ArrayAdapter as the source data. We also use a layout resource, which is the View that is repeated for each item in the array. This is defined as follows:

```
<TextView  xmlns:android= “http://schemas.android.com/apk/res/android”
        android:layout_width=”fill_parent”
        android:layout_height=”wrap_content”
        android:textSize=”20px” />
```

This layout resource contains only a single TextView. However, you can use a more complex layout with the constructors that also take the resource identifier of a TextView within the layout. Each child View within the AdapterView that uses this Adapter gets one TextView instance with one of the strings from the String array.

If you have an array resource defined, you can also directly set the entries attribute for an AdapterView to the resource identifier of the array to automatically provide the ArrayAdapter.

**How to use Cursor Adapter**

A CursorAdapter binds one or more columns of data to one or more View objects within the layout resource provided.This is best shown with an example. The following example demonstrates creating a CursorAdapter by querying the Contacts content provider. The CursorAdapter requires the use of a Cursor.

```
Cursor names = managedQuery(Contacts.Phones.CONTENT_URI, null, null, null, null);

startManagingCursor(names);
ListAdapter adapter = new SimpleCursorAdapter(
        this, R.layout.two_text,
        names, new String[] {
                Contacts.Phones.NAME,
                Contacts.Phones.NUMBER
        }, new int[] {
                R.id.scratch_text1,
                R.id.scratch_text2
        });
```

In this example, we present a couple of new concepts. First, you need to know that the Cursor must contain a field named _id. In this case, we know that the Contacts content provider does have this field. This field is used later when you handle the user selecting a particular item.

We make a call to managedQuery() to get the Cursor.Then, we instantiate a SimpleCursorAdapter as a ListAdapter. Our layout, R.layout.two_text, has two TextView objects in it, which are used in the last parameter. SimpleCursorAdapter enables us to match up columns in the database with particular controls in our layout. For each row returned from the query, we get one instance of the layout within our AdapterView.

**Binding Data to the AdapterView**

Now that you have an Adapter object, you can apply this to one of the AdapterView controls. Any of them works.Although the Gallery technically takes a SpinnerAdapter, the instantiation of SimpleCursorAdapter also returns a SpinnerAdapter. Here is an example of this with a ListView, continuing on from the previous sample code:

```
((ListView)findViewById(R.id.list)).setAdapter(adapter);
```

The call to the setAdapter() method of the AdapterView, a ListView in this case, should come after your call to setContentView().This is all that is required to bind data to your AdapterView. Figure given below shows the same data in a ListView, Gallery, and GridView.

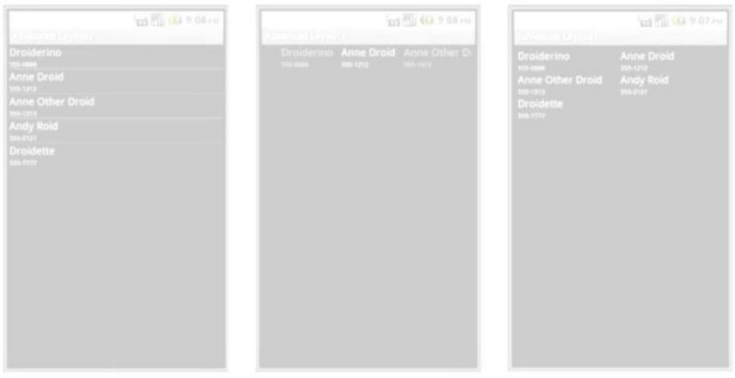



**Figure-69**

**Handling Selection Events**

You often use AdapterView controls to present data from which the user should select. All three of the discussed controls—ListView, GridView, and Gallery—enable your application to monitor for click events in the same way.You need to call setOnItemClickListener() on your AdapterView and pass in an implementation of the AdapterView.OnItemClickListener class.

Following is an example implementation of this class:

```
av.setOnItemClickListener(
        new AdapterView.OnItemClickListener() {
        public void onItemClick(
                AdapterView<?> parent, View view,
                int position, long id) {
                        Toast.makeText(Scratch.this, "Clicked _id="+id,
                                Toast.LENGTH_SHORT).show();
        }
});
```

In the preceding example, av is our AdapterView.The implementation of the onItemClick() method is where all the interesting work happens.The parent parameter is the AdapterView where the item was clicked.This is useful if your screen has more than one AdapterView on it. The View parameter is the specific View within the item that was clicked. The position is the zero-based position within the list of items that the user selects.

Finally, the id parameter is the value of the _id column for the particular item that the user selects. This is useful for querying for further information about that particular row of data that the item represents.

Your application can also listen for long-click events on particular items.Additionally, your application can listen for selected items.Although the parameters are the same, your application receives a call as the highlighted item changes.This can be in

response to the user scrolling with the arrow keys and not selecting an item for action.

## 3.1. Introduction

With Android, we can display images such as PNG and JPG graphics, as well as text and primitive shapes to the screen. We can paint these items with various colors, styles, or gradients and modify them using standard image transforms. We can even animate objects to give the illusion of motion.

## Canvas and Paint

The Canvas class holds the "draw" calls. To draw something, you need 4 basic components:

1. A Bitmap to hold the pixels,
2. A Canvas to host the draw calls (writing into the bitmap),
3. A drawing primitive (e.g. Rect, Path, text, Bitmap), and
4. A paint (to describe the colors and styles for the drawing).

The android.graphics framework divides drawing into two areas:

- What to draw, handled by Canvas

  How to draw, handled by Paint.

For instance, Canvas provides a method to draw a line, while Paint provides methods to define that line's color. Canvas has a method to draw a rectangle, while Paint defines whether to fill that rectangle with a color or leave it empty. Simply put, Canvas defines shapes that you can draw on the screen, while Paint defines the color, style, font, and so forth of each shape you draw.

So, before you draw anything, you need to create one or more Paint objects. The PieChart example does this in a method called init, which is called from the constructor from Java.

```
private void init() {
  textPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
  textPaint.setColor(textColor);
  if (textHeight == 0) {
     textHeight = textPaint.getTextSize();
  } else {
     textPaint.setTextSize(textHeight);
  }

  piePaint = new Paint(Paint.ANTI_ALIAS_FLAG);
  piePaint.setStyle(Paint.Style.FILL);
  piePaint.setTextSize(textHeight);

  shadowPaint = new Paint(0);
  shadowPaint.setColor(0xff101010);
  shadowPaint.setMaskFilter(new BlurMaskFilter(8, BlurMaskFilter.Blur.NORMAL));

    . . .
```

Creating objects ahead of time is an important optimization. Views are redrawn very frequently, and many drawing objects require expensive initialization. Creating drawing objects within your onDraw() method significantly reduces performance and can make your UI appear sluggish.

Once you have your object creation and measuring code defined, you can implement onDraw(). Every view implements onDraw() differently, but there are some common operations that most views share:

- Draw text using drawText(). Specify the typeface by calling setTypeface(), and the text color by calling setColor().
- Draw primitive shapes using drawRect(), drawOval(), and drawArc(). Change whether the shapes are filled, outlined, or both by calling setStyle().
- Draw more complex shapes using the Path class. Define a shape by adding lines and curves to a Path object, then draw the shape using drawPath(). Just as with primitive shapes, paths can be outlined, filled, or both, depending on the setStyle().

- Define gradient fills by creating LinearGradient objects. Call setShader() to use your LinearGradient on filled shapes.
- Draw bitmaps using drawBitmap().

For example, here's the code that draws PieChart. It uses a mix of text, lines, and shapes.

```
protected void onDraw(Canvas canvas) {
  super.onDraw(canvas);

  // Draw the shadow
  canvas.drawOval(shadowBounds, shadowPaint);

  // Draw the label text
  canvas.drawText(data.get(currentItem).mLabel, textX, textY, textPaint);

  // Draw the pie slices
  for (int i = 0; i < data.size(); ++i) {
     Item it = data.get(i);
     piePaint.setShader(it.shader);
     canvas.drawArc(bounds,360 - it.endAngle, it.endAngle - it.startAngle,
           true, piePaint);
  }
  // Draw the pointer
  canvas.drawLine(textX, pointerY, pointerX, pointerY, textPaint);
  canvas.drawCircle(pointerX, pointerY, pointerSize, mTextPaint);
}
```

## Bitmaps

You can find lots of goodies for working with graphics such as bitmaps in the android.graphics package. The core class for bitmaps is android.graphics.Bitmap.

### Drawing Bitmap Graphics on a Canvas

You can draw bitmaps onto a valid Canvas, such as within the onDraw() method of a View, using one of the drawBitmap() methods. For example, the following code loads a Bitmap resource and draws it on a canvas:

```
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
...
Bitmap pic = BitmapFactory.decodeResource(getResources(),
R.drawable.bluejay);
canvas.drawBitmap(pic, 0, 0, null);
```

### Scaling Bitmap Graphics

Perhaps you want to scale your graphic to a smaller size. In this case, you can use the createScaledBitmap() method, like this:

```
Bitmap sm = Bitmap.createScaledBitmap(pic, 50, 75, false);
```

You can preserve the aspect ratio of the Bitmap by checking the getWidth() and getHeight() methods and scaling appropriately.

## Shapes

You can define and draw primitive shapes such as rectangles and ovals using the ShapeDrawable class in conjunction with a variety of specialized Shape classes. You can define Paintable drawables as XML resource files, but more often, especially with more complex shapes, this is done programmatically.

### Defining Shape Drawables as XML Resources

In Unit-5, "Application Resources" of block-3, we show you how to define primitive shapes such as rectangles using specially formatted XML files within the /res/drawable/resource directory.

The following resource file called /res/drawable/green_rect.xml describes a simple, green rectangle shape drawable:

```
<?xml version=”1.0” encoding=”utf-8”?>
<shape xmlns:android=“http://schemas.android.com/apk/res/android”
        android:shape=”rectangle”>
        <solid android:color=”#0f0”/>
</shape>
```

You can then load the shape resource and set it as the Drawable as follows:

```
ImageView iView = (ImageView)findViewById(R.id.ImageView1);
iView.setImageResource(R.drawable.green_rect);
```

You should note that many Paint properties can be set via XML as part of the Shape definition. For example, the following Oval shape is defined with a linear gradient (red to white) and stroke style information:

```
<?xml version=”1.0” encoding=”utf-8”?>
<shape xmlns:android=”http://schemas.android.com/apk/res/android”
android:shape=”oval”>
<solid android:color=”#f00”/>
<gradient android:startColor=”#f00” android:endColor=”#fff” android:angle=”180”/>
<stroke android:width=”3dp”       android:color=”#00f”
      android:dashWidth=”5dp”    android:dashGap=”3dp”
/>
</shape>
```

**Defining Shape Drawables Programmatically**

You can also define this ShapeDrawable instances programmatically. The different shapes are available as classes within the android.graphics.drawable.shapes package. For example, you can programmatically define the aforementioned green rectangle as follows:

```
import android.graphics.drawable.ShapeDrawable;
import android.graphics.drawable.shapes.RectShape;
...

ShapeDrawable rect = new ShapeDrawable(new RectShape());
rect.getPaint().setColor(Color.GREEN);
```

You can then set the Drawable for the ImageView directly:

```
ImageView iView = (ImageView)findViewById(R.id.ImageView1);
iView.setImageDrawable(rect);
```

**Drawing Different Shapes**

Some of the different shapes available within the android.graphics.drawable.shapes package include

- Rectangles (and squares)
- Rectangles with rounded corners
- Ovals (and circles)
- Arcs and lines
- Other shapes defined as paths

You can create and use these shapes as Drawable resources directly within ImageView views, or you can find corresponding methods for creating these primitive shapes within a Canvas.

**Drawing Rectangles and Squares**

Drawing rectangles and squares (rectangles with equal height/width values) is simply a matter of creating a ShapeDrawable from a RectShape object. The RectShape object has no dimensions but is bound by the container object—in this case, the ShapeDrawable.

You can set some basic properties of the ShapeDrawable, such as the Paint color and the default size.

For example, here we create a magenta-colored rectangle that is 100-pixels long and 2-pixels wide, which looks like a straight, horizontal line. We then set the shape as the drawable for an ImageView so the shape can be displayed:

```
import android.graphics.drawable.ShapeDrawable;
import android.graphics.drawable.shapes.RectShape;
…

ShapeDrawable rect = new ShapeDrawable(new RectShape());
rect.setIntrinsicHeight(2);
rect.setIntrinsicWidth(100);
rect.getPaint().setColor(Color.MAGENTA);
ImageView iView = (ImageView)findViewById(R.id.ImageView1);
iView.setImageDrawable(rect);
```

Similarly we can draw other shapes.

## Frame by Frame animation

You can think of frame-by-frame animation as a digital flipbook in which a series of similar images display on the screen in a sequence, each subtly different from the last. When you display these images quickly, they give the illusion of movement. This technique is called frame-by-frame animation and is often used on the Web in the form of animated GIF images.

Frame-by-frame animation is best used for complicated graphics transformations that are not easily implemented programmatically.

An object used to create frame-by-frame animations, defined by a series of Drawable objects, which can be used as a View object's background.

The simplest way to create a frame-by-frame animation is to define the animation in an XML file, placed in the res/drawable/ folder, and set it as the background to a View object. Then, call start() to run the animation.

An AnimationDrawable defined in XML consists of a single <animation-list> element and a series of nested <item> tags. Each item defines a frame of the animation. See the example below.

spin_animation.xml file in res/drawable/ folder:

```
<animation-list android:id="@+id/selected" android:oneshot="false">
    <item android:drawable="@drawable/wheel0" android:duration="50" />
    <item android:drawable="@drawable/wheel1" android:duration="50" />
    <item android:drawable="@drawable/wheel2" android:duration="50" />
    <item android:drawable="@drawable/wheel3" android:duration="50" />
    <item android:drawable="@drawable/wheel4" android:duration="50" />
    <item android:drawable="@drawable/wheel5" android:duration="50" />
 </animation-list>
```

Here is the code to load and play this animation.

```
// Load the ImageView that will host the animation and
// set its background to our AnimationDrawable XML resource.
   ImageView img = (ImageView)findViewById(R.id.spinning_wheel_image);
   img.setBackgroundResource(R.drawable.spin_animation);

 // Get the background, which has been compiled to an AnimationDrawable object.
   AnimationDrawable frameAnimation = (AnimationDrawable) img.getBackground();

 // Start the animation (looped playback by default).
    frameAnimation.start();
```

## Tweened Animation

With tweened animation, you can provide a single Drawable resource - it is a Bitmap graphic, a ShapeDrawable, a TextView, or any other type of View object—and the intermediate frames of the animation are rendered by the system. Android provides tweening support for several common image transformations, including alpha, rotate, scale, and translate animations. You can apply tweened animation transformations to any View, whether it is an ImageView with a Bitmap or shape Drawable, or a layout such as a TableLayout.

**Defining Tweening Transformations**

You can define tweening transformations as XML resource files or programmatically. All tweened animations share some common properties, including when to start, how long to animate, and whether to return to the starting state upon completion.

**Defining Tweened Animations as XML Resources**

In Unit-5 of Block-3, we showed you how to store animation sequences as specially formatted XML files within the /res/anim/ resource directory. For example, the following resource file called /res/anim/spin.xml describes a simple five-second rotation:

```
<?xml version="1.0" encoding="utf-8" ?>
<set xmlns:android= "http://schemas.android.com/apk/res/android"
      android:shareInterpolator="false">
        <rotate android:fromDegrees="0"
                  android:toDegrees="360"
                  android:pivotX="50%"
                  android:pivotY="50%"
                  android:duration="5000" />
</set>
```

**Defining Tweened Animations Programmatically**

You can programmatically define these animations.The different types of transformations are available as classes within the android.view.animation package. For example, you can define the aforementioned rotation animation as follows:

```
import android.view.animation.RotateAnimation;
...
RotateAnimation rotate = new RotateAnimation(0, 360,
                                RotateAnimation.RELATIVE_TO_SELF, 0.5f,
                                RotateAnimation.RELATIVE_TO_SELF, 0.5f);
rotate.setDuration(5000);
```

**Defining Simultaneous and Sequential Tweened Animations**

Animation transformations can happen simultaneously or sequentially when you set the startOffset and duration properties, which control when and for how long an animation takes to complete. You can combine animations into the <set> tag (programmatically, using AnimationSet) to share properties.

For example, the following animation resource file /res/anim/grow.xml includes a set of two scale animations: First, we take 2.5 seconds to double in size, and then at 2.5 seconds, we start a second animation to shrink back to our starting size:

```
<?xml version="1.0" encoding="utf-8" ?>
<set xmlns:android=http://schemas.android.com/apk/res/android
                android:shareInterpolator="false">
        <scale android:pivotX="50%"
                android:pivotY="50%"
                android:fromXScale="1.0"
                android:fromYScale="1.0"
                android:toXScale="2.0"
                android:toYScale="2.0"
                android:duration="2500" />
        <scale
```

```
            android:startOffset="2500"
            android:duration="2500"
            android:pivotX="50%"
            android:pivotY="50%"
            android:fromXScale="1.0"
            android:fromYScale="1.0"
            android:toXScale="0.5"
            android:toYScale="0.5" />
</set>
```

**Loading Animations**

Loading animations is made simple by using the AnimationUtils helper class.The following code loads an animation XML resource file called /res/anim/grow.xml and applies it to an ImageView whose source resource is a green rectangle shape drawable:

```
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
...
ImageView iView = (ImageView)findViewById(R.id.ImageView1);
iView.setImageResource(R.drawable.green_rect);
Animation an = AnimationUtils.loadAnimation(this, R.anim.grow);
iView.startAnimation(an);
```

We can listen for Animation events, including the animation start, end, and repeat events, by implementing an AnimationListener class, such as the MyListener class shown here:

```
class MyListener implements Animation.AnimationListener {
        public void onAnimationEnd(Animation animation) {
                // Do at end of animation
        }
        public void onAnimationRepeat(Animation animation) {
                // Do each time the animation loops
```

```
        }
        public void onAnimationStart(Animation animation) {
                // Do at start of animation
        }
}
```

You can then register your AnimationListener as follows:

```
an.setAnimationListener(new MyListener());
```

Now let's look at each of the four types of tweening transformations individually. These types are:

- Transparency changes (Alpha)
- Rotations (Rotate)
- Scaling (Scale)
- Movement (Translate)

**Working with Alpha Transparency Transformations**

Transparency is controlled using Alpha transformations. Alpha transformations can be used to fade objects in and out of view or to layer them on the screen.

Alpha values range from 0.0 (fully transparent or invisible) to 1.0 (fully opaque or visible). Alpha animations involve a starting transparency (fromAlpha) and an ending transparency (toAlpha).

The following XML resource file excerpt defines a transparency-change animation, taking five seconds to fade in from fully transparent to fully opaque:

```
<alpha android:fromAlpha="0.0"
        android:toAlpha="1.0"
        android:duration="5000">
</alpha>
```

Programmatically, you can create this same animation using the AlphaAnimation class within the android.view.animation package.

**Working with Rotating Transformations**

You can use rotation operations to spin objects clockwise or counterclockwise around a pivot point within the object's boundaries.

Rotations are defined in terms of degrees. For example, you might want an object to make one complete clockwise rotation. To do this, you set the fromDegrees property to 0 and the toDegrees property to 360. To rotate the object counterclockwise instead, you set the toDegrees property to -360.

By default, the object pivots around the (0,0) coordinate, or the top-left corner of the object. This is great for rotations such as those of a clock's hands, but much of the time, you want to pivot from the center of the object; you can do this easily by setting the pivot point, which can be a fixed coordinate or a percentage.

The following XML resource file excerpt defines a rotation animation, taking five seconds to make one full clockwise rotation, pivoting from the center of the object:

```
<rotate android:fromDegrees="0"
        android:toDegrees="360"
        android:pivotX="50%"
        android:pivotY="50%"
        android:duration="5000" />
```

Programmatically, you can create this same animation using the RotateAnimation class within the android.view.animation package.

**Working with Scaling Transformations**

You can use scaling operations to stretch objects vertically and horizontally. Scaling operations are defined as relative scales.Think of the scale value of 1.0 as 100 percent, or fullsize. To scale to half-size, or 50 percent, set the target scale value of 0.5. You can scale horizontally and vertically on different scales or on the same scale (to preserve aspect ratio).You need to set four values for proper scaling:

starting scale (fromXScale, fromYScale) and target scale (toXScale, toYScale). Again, you can use a pivot point to stretch your object from a specific (x,y) coordinate such as the center or another coordinate.

The following XML resource file excerpt defines a scaling animation, taking five seconds to double an object's size, pivoting from the center of the object:

```
<scale android:pivotX="50%"
       android:pivotY="50%"
       android:fromXScale="1.0"
       android:fromYScale="1.0"
       android:toXScale="2.0"
       android:toYScale="2.0"
       android:duration="5000" />
```

Programmatically, you can create this same animation using the ScaleAnimation class within the android.view.animation package.

**Working with Moving Transformations**

You can move objects around using translate operations.Translate operations move an object from one position on the (x,y) coordinate to another coordinate.

To perform a translate operation, you must specify the change, or delta, in the object's coordinates. You can set four values for translations: starting position (fromXDelta, fromYDelta) and relative target location (toXDelta, toYDelta).

The following XML resource file excerpt defines a translate animation, taking 5 seconds to move an object up (negative) by 100 on the y-axis.We also set the fillAfter property to be true, so the object doesn't "jump" back to its starting position when the animation finishes:

```
<translate    android:toYDelta="-100"
              android:fillAfter="true"
              android:duration="2500" />
```

Programmatically, you can create this same animation using the TranslateAnimation class within the android.view.animation package.