



Indira Gandhi National Open University  
School of Computer and Information  
Sciences (SOCIS)

# BCS-093

## Introduction to Android



# Features of Android

# 1

**Block****1****FEATURES OF ANDROID**

---

<b>Unit 1</b>	<b>7</b>
---------------	----------

<b>Basics Of Android</b>	
--------------------------	--

---

<b>Unit 2</b>	<b>19</b>
---------------	-----------

<b>Android Architecture</b>	
-----------------------------	--

---

<b>Unit 3</b>	<b>31</b>
---------------	-----------

<b>Activity Lifecycle</b>	
---------------------------	--

---

<b>Unit 4</b>	<b>39</b>
---------------	-----------

<b>Android Development Environment</b>	
--	--

---

<b>Unit 5</b>	<b>49</b>
---------------	-----------

<b>Android Application Fundamentals</b>	
---	--

---

<b>Unit 6</b>	<b>69</b>
---------------	-----------

<b>Android Development</b>	
----------------------------	--

---

<b>Unit 7</b>	<b>79</b>
---------------	-----------

<b>Device Compatibility</b>	
-----------------------------	--

---

<b>Unit 8</b>	<b>87</b>
---------------	-----------

<b>User Interface Design</b>	
------------------------------	--

---

---

*Course Coordinator*

Prof.P.V.Suresh  
SOCIS, IGNOU, New Delhi-110068

Copyright : CC-BY-SA

*The content is adopted to IGNOU CRC format. Wherever needed, content for  
Check Your Progress is made (2021)*

Prof.P.V.Suresh  
SOCIS, IGNOU, New Delhi-110068

---



---

**CRC Preparation (2021)**

Mr.Vishal Singh, Skilled Worker, SOCIS, IGNOU, New Delhi - 110068

---

# Copyright

This course has been developed as part of the collaborative advanced ICT course development project of the Commonwealth of Learning (COL). COL is an intergovernmental organisation created by Commonwealth Heads of Government to promote the development and sharing of open learning and distance education knowledge, resources and technologies. Herewith, its acknowledged that the image portion in Cover Page is also adopted.

The Open University of Sri Lanka (OUSL) is the premier Open and Distance learning institution in the country where students can pursue their studies through Open and Distance Learning (ODL) methodologies. Degrees awarded by OUSL are treated as equivalent to the degrees awarded by other national universities in Sri Lanka by the University Grants Commission of Sri Lanka.



© 2017 by the Commonwealth of Learning and The Open University of Sri Lanka. Except where otherwise noted, *Introduction to Android* is made available under Creative Commons Attribution- ShareAlike 4.0 International (CC BY-SA 4.0) License: <https://creativecommons.org/licenses/by-sa/4.0/legalcode>.

For the avoidance of doubt, by applying this license the Commonwealth of Learning does not waive any privileges or immunities from claims that it may be entitled to assert, nor does the Commonwealth of Learning submit itself to the jurisdiction, courts, legal processes or laws of any jurisdiction. The ideas and opinions expressed in this publication are those of the author/s; they are not necessarily those of Commonwealth of Learning and do not commit the organisation.



The Open University of Sri Lanka  
P. O. Box 10250,  
Nawala,  
Nugegoda,  
Sri Lanka  
Phone: +94 112881481  
Fax: +94 112821285  
Email: [hdelect@ou.ac.lk](mailto:hdelect@ou.ac.lk)  
Website: [www.ou.ac.lk](http://www.ou.ac.lk)

Commonwealth of Learning  
4710 Kingsway, Suite 2500, Burnaby  
V5H 4M2,  
British Columbia,  
Canada  
Phone: +1 604 775 8200  
Fax: +1 604 775 8210  
Email: [info@col.org](mailto:info@col.org)  
Website: [www.col.org](http://www.col.org)

---

## Acknowledgements

---

Department of Electrical and Computer Engineering (ECE), The Open University of Sri Lanka (OUSL) wishes to thank those below for their contribution to this course material and accompanying Videos and Screencasts:

### **Chairperson of the Course team:**

H.U.W. Ratnayake (Senior Lecturer, Dept. of ECE, OUSL)

### **Authors:**

W.A.S.N. Perera (Lecturer, Dept. of ECE, OUSL)	Units 1 and 2
J. Nananyakkara (Lecturer, Dept. of ECE, OUSL)	Units 3 and 4
B.K. Werapitiya (Lecturer, Dept. of ECE, OUSL)	Units 5 and 6
S. Rajasingham (Lecturer, Dept. of ECE, OUSL)	Units 7 and 8
U.S. Premaratne (Lecturer, Dept. of ECE, OUSL)	Units 9 and 10
W.U. Erandaka (Lecturer, Dept. of Textile & Apparel Tech, OUSL)	Units 11 and 12
W.W.A.I.D. Wickramasinghe (Demonstrator, Dept. of ECE, OUSL)	Units 13 and 14
H.U.W. Ratnayake (Senior Lecturer, Dept. of ECE, OUSL)	Units 15 and 16

### **Content Editors:**

C.W.S. Goonatilleke (Senior Software Engineer, WSO2 Inc)

G.S.N. Meedin (Lecturer, Dept. of ECE, OUSL)

### **Language Editor:**

G.S.N. Meedin (Lecturer, Dept. of ECE, OUSL)

### **Reviewer:**

D.G. U. Kulasekara (Senior Lecturer/Centre for Educational Technology & Media, OUSL)

### **Video Presenters:**

U.S. Premartane (Lecturer, Dept. of ECE, OUSL)

S. Rajasingham (Lecturer, Dept. of ECE, OUSL)

### **Screen casters:**

C.W.S. Goonatilleke (Senior Software Engineer, WSO2 Inc)

---

## COURSE INTRODUCTION

---

This Course is developed by Open University of Sri Lanka and is adapted by Indira Gandhi National Open University. The copyright of the Course Material remains CC-BY-SA.

This Course comprises of 2 blocks. The blocks may be increased in future as per need.

This Course introduces Android and covers Android Architecture, Android Development Environment, Android APP fundamentals, Development in Android environment, Developing applications so that they run across Devices, Designing user interfaces, Testing and Debugging the applications, Including Multimedia, Saving Data on Android Devices, Making APPS location aware and using sensors, Publishing APP to Android Market, , Measuring the performance of APPs, and ensuring security of APS.



---

## BLOCK INTRODUCTION

---

This Block primarily introduces features of Android. There are 8 units in this Block:

Unit 1 introduces Android

Unit 2 introduces Architecture of Android

Unit 3 introduces Activity Lifecycle.

Unit 4 introduces the Android Development Environment

Unit 5 introduces fundamentals of Android Applications

Unit 6 enables you to develop First Program in Android Environment

Unit 7 focuses on Device compatibility

Unit 8 focuses on User Interface Design



---

# UNIT 1 BASICS OF ANDROID

---

- 1.0 Introduction
- 1.1 Objectives
- 1.2 Android as a popular mobile platform
  - 1.2.1 Rapid innovation
  - 1.2.2 Powerful development framework
  - 1.2.3 Video V-1: Course Overview
  - 1.2.4 Check Your Progress
- 1.3 History of Android
  - 1.3.1 Version History
  - 1.3.2 Video V-2: Evolution of Android
  - 1.3.3 Check Your Progress
- 1.4 Features of Android
- 1.5 Comparison of mobile Operating systems
- 1.6 Devices that run Android as the Operating System
  - 1.6.1 Check Your Progress
- 1.7 Categories of Android applications
- 1.8 Summary
- 1.9 Answers to Check Your Progress
- 1.10 Further Readings

---

## 1.0 INTRODUCTION

---

The purpose of this unit is to give you the first glimpse of the Android operating system for mobile devices and tablets. It is designed to empower mobile software developers to write innovative mobile applications. First part of this unit looks at the version history of the Android mobile operating system. Hence, you will be able to compare different mobile operating systems with unique features of Android.

The next part of the unit will help you to identify the devices that run Android as the Operating System with its open and customizable nature. Furthermore, at the end of this unit, you would be able to have an idea about the available categories of applications in Google Play.

One video material will be provided with this unit and you are expected to watch this and complete the relevant activities.

---

## 1.1 OBJECTIVES

---

After studying this unit, you should be able to:



### Outcomes

- describe how versioning and naming of Android Operating Systems has evolved.
- explain the unique features of Android OS comparing to existing mobile Operating Systems.
- identify the devices that run Android as the Operating System.
- identify the types of applications run on top of Android devices.





### Terminology

IDE:	Integrated Development Environment to write, compile and and execute programs
app:	mobile application software
API:	Application Program Interface is a set of protocols and tools for building software
Kernel	essential core of an operating system

---

## 1.2 ANDROID AS A POPULAR MOBILE PLATFORM

---

Android is an open-source operating system for mobile devices such as smart phones, smart watches, tablets, and other Android enabled platforms including Android TV and Android Auto. Android Auto is a smartphone projection standard developed by Google to allow mobile devices running the Android operating system (version 5.0 "Lollipop" and later) to be operated in automobiles through the dashboard's head unit. Android is a Linux based operating system.

*“Android is the first truly open and comprehensive platform for mobile devices. It includes an operating system, user-interface and applications - all of the software to run a mobile phone, but without the proprietary obstacles that have hindered mobile innovation.”*

-By Andy Rubin (Founder of Android Inc.)

In other words, Android can be defined as a system that includes an open source operating system, an open source development platform and devices that run the operating system and applications created for it.

In the next section, we will be discussing about the popularity of Android OS among people.

### 1.2.1 Rapid innovation

Android is continuously pushing the boundaries of hardware and software forward, to bring new capabilities to users and developers. For developers like you, the rapid evolution of Android technology lets you stay in front with powerful, differentiated applications.

Android gives you access to the latest technologies and innovations across a multitude of device form factors, chipset architectures, and price points. If you are not familiar with the terms, a form factor is the size, configuration, or physical arrangement of the device and a price point is a point on a scale of possible prices at which something might be marketed. The range of features further includes multicore processing and high-performance graphics, state of the art sensors, vibrant touch-screens, and emerging mobile technologies.

### 1.2.2 Powerful development framework

Android facilitates you with everything you need to build best-in-class app experiences. It gives you a single application model that lets you deploy your apps broadly to hundreds of millions of users across a wide range of devices from phones to tablets and beyond.

Android also gives you tools for creating apps while taking advantage of the hardware capabilities available on each device. It automatically adapts your User Interface (UI) to look its best on each device, and gives you control over UI on different device types. This is further discussed later in this material.

For example, you can create a single app binary that's optimized for both phone and tablet form factors. What is a single app binary and how a single app binary can be deployed is discussed in a later unit of this material. Android allows you to declare your UI in lightweight sets of Extensible Markup Language (XML) resources, one set for parts of the UI that are common to all form factors and other sets for optimizations specific to phones or tablets. How UI designing is done is further explained in a later unit of this material. At runtime, Android applies the correct resource sets based on its screen size, density, locale, and so on.

To help you develop efficiently, the Android Developer Tools offer a full Java Integrated development environment (IDE) with advanced features for developing, debugging, and packaging Android apps. Using the IDE, you can develop on any available Android device or create virtual devices that emulate any hardware configuration.

### 1.2.3 Video V-1: Course Overview



You may watch the video on “course overview” before moving further and answer the questions in Activity 1.1.



URL: <https://tinyurl.com/ydhf7mf6>

### 1.2.4 Check Your Progress



Check the Android version, i.e. Kernel version, in an Android phone. What are the other devices you have seen having Android operating system?

---

## 1.3 HISTORY OF ANDROID

---

In October, 2003, four computer experts, Andy Rubin, Nick Sears, Rich Miner and Chris White founded a software development organization Android Inc. in Palo Alto, California, USA. They wanted to make a Linux based operating system that can work

on digital cameras which can connect with computers. However, this plan was not as successful as they thought, so they focused on smart phones.

In August, 2005, Google purchased the Android Inc. and became the proprietor of the company. In November, 2007, Google disclosed a consortium of different mobile technology providers named Open Handset Alliance (OHA) that includes mobile hardware manufacturers (HTC, Motorola etc.), chipset manufacturers (Qualcomm, Texas Instruments etc.), and telecommunication service providers (T-Mobile etc.). There were 34 different companies in OHA consortium that agreed to provide a mobile device which does not belongs to a single company as iPhone from Apple. But for couple of years Google could not bring any mobile under the OHA consortium. In October, 2008, HTC brought first smart phone “HTC Dream” in the market which was commercially available. At the time when the first version of the Android was unveiled, only 35 Android apps were accessible. But today, millionsof Android applications are available in the market.

Android has been released in many versions since its inception. Before commercialization, many internal alpha versions were released on the name of fictional robots (like Astro Boy, Bender, R2-D2 etc.). On November 5, 2007 Google released first beta version of Android whose Software Development Kit (SDK) was released on November 12, 2007. Since then, November 5<sup>th</sup> is considered as Android’s Birthday.

### 1.3.1 Version History

All versions of Android are released under a confectionary or sweet theme; i.e. names of the Android versions are the name of confectionary product in alphabetic order. It started with Android 1.5 "Cupcake"; versions 1.0 and 1.1 (API version 1 and 2) and they were not released under explicit code names.

API level is mainly the Android version used as an alternative to the Android version name (e.g. 3.0, 4.0, 4.4, etc.) where integer numbers are applied. This number keeps on increasing with each version, for e.g. Android 1.5 is API Level 3; Android 1.6 is API Level 4, and so on. Android 11 (API Level 30) is latest version of Android that is available on some of the latest releases of mobile phones.

### 1.3.2 Video V-2: Evolution of Android



Let us watch the vide on “Evolution of Android” before moving further and answer the questions in Activity 1.2.



URL: <https://tinyurl.com/ydhf7mf6>

With their main features, different versions of Android are summarized in the following Table 1.1. You need not to worry if you are not familiar with the terms given under features. You can refer this table while reading the remaining units of this material as when required.

Name	Internal codename	Version number(s)	Initial stable release date	Supported (security fixes)	API level	References
No official codename	–	1.0	September 23, 2008	No	1	<a href="#">[9]</a> <a href="#">[14]</a>
	Petit Four	1.1	February 9, 2009	No	2	<a href="#">[9]</a> <a href="#">[14]</a> <a href="#">[15]</a>
<a href="#">Cupcake</a>		1.5	April 27, 2009	No	3	<a href="#">[14]</a> <a href="#">[16]</a>
<a href="#">Donut</a>		1.6	September 15, 2009	No	4	<a href="#">[14]</a> <a href="#">[17]</a>
<a href="#">Eclair</a>		2.0	October 27, 2009	No	5	<a href="#">[14]</a> <a href="#">[18]</a> <a href="#">[19]</a>
		2.0.1	December 3, 2009	No	6	
		2.1	January 11, 2010	No	7	<a href="#">[20]</a>
<a href="#">Froyo</a>		2.2 – 2.2.3	May 20, 2010	No	8	<a href="#">[14]</a> <a href="#">[21]</a>
<a href="#">Gingerbread</a>		2.3 – 2.3.2	December 6, 2010	No	9	<a href="#">[14]</a> <a href="#">[22]</a>
		2.3.3 - 2.3.7	February 9, 2011	No	10	
<a href="#">Honeycomb</a>		3.0	February 22, 2011	No	11	<a href="#">[14]</a> <a href="#">[23]</a>
		3.1	May 10, 2011	No	12	
		3.2 - 3.2.6	July 15, 2011	No	13	
<a href="#">Ice Cream Sandwich</a>		4.0 – 4.0.2	October 18, 2011	No	14	<a href="#">[14]</a> <a href="#">[24]</a>
		4.0.3 - 4.0.4	December 16, 2011	No	15	
<a href="#">Jelly Bean</a>		4.1 – 4.1.2	July 9, 2012	No	16	<a href="#">[14]</a> <a href="#">[25]</a>
		4.2 - 4.2.2	November 13, 2012	No	17	
		4.3 - 4.3.1	July 24, 2013	No	18	
<a href="#">KitKat</a>	Key Lime Pie	4.4 – 4.4.4	October 31, 2013	No	19	<a href="#">[14]</a> <a href="#">[26]</a>
		4.4W - 4.4W.2	June 25, 2014	No	20	
<a href="#">Lollipop</a>	Lemon Meringue Pie	5.0 – 5.0.2	November 4, 2014	No	21	<a href="#">[14]</a> <a href="#">[27]</a> <a href="#">[28]</a>
		5.1 - 5.1.1	March 2, 2015	No	22	<a href="#">[29]</a>
<a href="#">Marshmallow</a>	Macadamia Nut Cookie	6.0 – 6.0.1	October 2, 2015	No	23	<a href="#">[14]</a> <a href="#">[30]</a> <a href="#">[31]</a>
<a href="#">Nougat</a>	New York Cheesecake	7.0	August 22, 2016	No	24	<a href="#">[14]</a> <a href="#">[32]</a> <a href="#">[33]</a> <a href="#">[34]</a> <a href="#">[35]</a>
		7.1 - 7.1.2	October 4, 2016	No	25	
<a href="#">Oreo</a>	Oatmeal Cookie	8.0	August 21, 2017	No	26	<a href="#">[14]</a> <a href="#">[36]</a> <a href="#">[37]</a>
		8.1	December 5, 2017	Yes <sup>[a]</sup>	27	<a href="#">[14]</a> <a href="#">[38]</a>
<a href="#">Pie</a>		9	August 6, 2018	Yes	28	<a href="#">[14]</a> <a href="#">[39]</a>
<a href="#">Android 10</a>	Queen Cake	10	September 7, 2019	Yes	29	<a href="#">[14]</a> <a href="#">[40]</a> <a href="#">[41]</a>
<a href="#">Android 11</a>	Red Velvet Cake	11	September 8, 2020	Yes	30	<a href="#">[14]</a> <a href="#">[42]</a>
<a href="#">Android 12</a>	Snow Cone	12	TBA	Presupported	31	<a href="#">[14]</a> <a href="#">[43]</a>

### 1.3.3 Check Your Progress



Write the Android version name corresponding to following distinct features of different versions.

1. First version whose name was on the name of a bakery product.
2. First version having the capability of speech and gesture support.
3. First version that is having USB tethering and Wi-Fi hotspot.

4. This version can work on 340 MB RAM.
5. This version is built around the API level 5.
6. This version enables multi-window UI.

In the next section we will be discussing the features of Android.

## 1.4 FEATURES OF ANDROID

Android is a powerful operating system with many supporting features for mobile application developers. Few of them are listed below in Table 1.2.

Table 1.2: Features of Android

Feature	Description
<b>UI</b>	Android provides a variety of pre-built UI components such as structured layout objects and UI controls to build the graphical user interface for your app. Android also provides other UI modules for special interfaces such as dialogs, notifications, and menus with its own unique effects and animations.
<b>Connectivity</b>	Android supports connectivity technologies including for Wide Area Networks (WAN) like GSM, 3G, 4G , CDMA and 5G. Also it is supporting Wi-Fi and Ethernet as Local Area Network (LAN) technologies. Apart from that Android has support for Bluetooth as a Personal Area Network (PAN). Also newer versions support for Near Field Communication (NFC).
<b>Storage</b>	<p>Android provides several options for you to save persistent application data. The solution you choose depends on your specific needs, such as whether the data should be private to your application or accessible to other applications (and the user) and how much space your data requires.</p> <p>Your data storage options are the following: Shared Preferences, Internal Storage, External Storage, SQLite Databases, Network Connection, Cloud storage</p>
<b>Media support</b>	<p>Media format support built into the Android platform.</p> <p>Audio - MP3, MIDI, MP4 (only on Android 10+)</p> <p>Images – JPEG, GIF, PNG, BMP</p> <p>Video - MPEG-4 SP</p>

Feature	Description
<b>Messaging</b>	Short Message Service (SMS) and Multimedia Messaging Service (MMS). Google Cloud Messaging (GCM) is also a part of Android Push Messaging services.
<b>Web browser</b>	The web browser available in Android is based on the open-source Blink (previously WebKit) layout engine, coupled with Chrome's V8 JavaScript engine. It supports both Hyper Text Markup Language (HTML5) and Cascading Style Sheets (CSS3).
<b>Multi-touch</b>	A multi-touch gesture is when multiple pointers (fingers) touch the screen at the same time. Android has native support for multi-touch.
<b>Multi-tasking</b>	<p>Multitasking — running multiple tasks simultaneously.</p> <p>When an activity has been launched, the user can go to Home and launch a second activity without destroying the first activity. User can jump from one task to another and same time various applications can run simultaneously.</p>
<b>Resizable widgets</b>	<p>App Widgets are miniature application views that can be embedded in other applications (such as the Home screen) and receive periodic updates. These views are referred to as Widgets in the user interface, and you can publish one with an App Widget provider.</p> <p>Resizing allows users to adjust the height and/or the width of a widget within the constraints of the home panel placement grid. You can decide if your widget is freely resizable or if it is constrained to horizontal or vertical size changes.</p>
<b>Multi-Language</b>	It is always a good idea to make the app localized and Android supports multiple languages.
<b>Android Beam</b>	Android Beam is a device-to-device data transfer tool that uses NFC and Bluetooth to send photos, videos, contact information, links to webpages, navigate directions and more from one device to another just by bumping them together. Android framework APIs supports these features.

Next, we will be comparing Android with other existing mobile operating systems.

**✎ Check Your Progress:**

1. What is the latest version of Android?

.....

.....

.....

.....

2. When was the first version of Android launched?

.....

.....

.....

.....

3. Name any three Mobile Operating Systems.

.....

.....

.....

.....

---

## 1.5 COMPARISON OF MOBILE OPERATING SYSTEMS

---

Google's Android, Apple's iOS, Microsoft's Windows and BlackBerry Ltd's Blackberry are operating systems used primarily in mobile devices, such as smartphones and tablets. The popular mobile operating systems are very similar in some ways. Every OS supports some kind of mobile device management, but the way each OS supports is different and one of the unique features of Android is its source model. Android has its unique features when comparing with proprietary mobile operating systems. Table 1.3 shows a comparison of Android with one of the proprietary operating system, iOS.

Table 1.3: Comparison of mobile operating systems

	<b>Android</b>	<b>iOS</b>
<b>Developer</b>	Open Handset Alliance	Apple Inc.
<b>Initial release</b>	September 23, 2008	July 29, 2007

	<b>Android</b>	<b>iOS</b>
<b>Source model</b>	Open source	Closed, with open source components.
<b>Available on</b>	Many phones and tablets, LG, HTC, Samsung, Sony, Motorola, Nexus, Google Glasses	iPod Touch, iPhone, iPad, Apple TV
<b>Messaging</b>	Google Hangouts	iMessage
<b>App store</b>	Google Play	Apple app store
<b>Video chat</b>	Google Hangouts	Facetime
<b>OS family</b>	Linux	Unix-like, based on Darwin
<b>Programmed in</b>	C, C++, Java	C, C++, Objective-C, Swift
<b>Internet browsing</b>	Google Chrome	Mobile Safari
<b>Voice commands</b>	Yes	Siri
<b>Latest stable release</b>	Android 6.0.1 (Marshmallow), (October 2015)	9.3 (March 21, 2016)
<b>Device manufacturer</b>	Google, LG, Samsung, HTC, Sony, ASUS, Motorola, and many more	Apple Inc

---

## 1.6 DEVICES THAT RUN ANDROID AS THE OPERATING SYSTEM

---

Android occupies a large section of the global mobile market. Here is a list of devices, which already run Android.

- Watches
- Smart glasses



- Home Appliances – E.g.:  
Refrigerators, washing machines, oven
- Cars
- Cameras
- Smart TVs
- Game consoles
- Home automation systems
- Robots

### 1.6.1 Check Your Progress



Give reasons for Android operating system becoming very popular.

---

## 1.7 CATEGORIES OF ANDROID APPLICATIONS

---

Google Play is the premier marketplace for selling and distributing Android apps. When you publish an app on Google Play, you reach the huge installed base of Android.

Using the Google Play Developer Console, you can choose a category for your apps. Users can browse for apps by category using a computer ([play.google.com](http://play.google.com)) and the Play Store app.

There are many Android applications in the market. Some of the top categories are:

- Music
- Multimedia
- News
- Sports
- Travel
- Weather
- Books
- Finance
- Social Media

---

## 1.8 SUMMARY

---

The objective of this unit is to introduce Android and its capabilities. In the first part of this unit, we discussed about the history of Android with the names of the various versions of the Android Operating System (OS). Then we discussed the features provided by Android to create mobile applications and compared different operating systems available for mobile devices. Android is a powerful Operating System that supports many applications in Smart Phones.

---

## 1.9 ANSWERS TO CHECK YOUR PROGRESS

---

1. Android v11.0 (September 8<sup>th</sup>, 2020)
2. Android v1.0 (September 23<sup>rd</sup>, 2008)
3. Android, iOS, and Blackberry

---

## 1.10 FURTHER READINGS

---

- <https://www.android.com/>
- <https://en.wikipedia.org/wiki/Android>



ignou  
THE PEOPLE'S  
UNIVERSITY



---

## UNIT 2 ANDROID ARCHITECTURE

---

### 2.0 Introduction

#### 2.1 Objectives

#### 2.2 Android Architecture

##### 2.2.1 Check Your Progress

##### 2.2.2 Video V3: Android Architecture

##### 2.2.3 Check Your Progress

#### 2.3 Types of mobile applications

##### 2.3.1 Check Your Progress

#### 2.4 Application Fundamentals

#### 2.5 Summary

#### 2.6 Further Readings

---

## 2.0 INTRODUCTION

---

This unit gives you an overview of the basic internal architecture of Android. The first part of this unit focuses on the Android architecture and the application framework, which developers can leverage in developing Android applications. Later it discusses different types of mobile applications with their pros and cons. This unit has one video that you need to watch and three activities to complete.

---

## 2.1 OBJECTIVES

---

After studying this unit, you should be able to :



### Outcomes

- illustrate the components of Android Operating System.
- differentiate mobile application development.
- identify the components provided by the application framework.



### Terminology

<b>Linux:</b>	Name of the operating system that Android is built on
<b>Dalvik:</b>	Runtime and Virtual Machine used by the Android system for running Android applications
<b>Native app:</b>	apps that are fully programmed in the development environment specific to each operating system
<b>ART:</b>	(Android Run Time ) is the successor of Dalvik and used in latest versions of Android

## 2.2 ANDROID ARCHITECTURE

The architecture of an Android system is a collection of different layers. Each layer has a specific role and set of functionality. Each layer provides the functionality to the layer above it.

As you can see in the Figure 2.1 that Android Architecture (also called Software Stack) has the following layers:

- Linux Kernel
- Hardware Abstraction Layer (HAL)
- Native Libraries
- Android Run Time
- Android Application Framework
- Application Layer

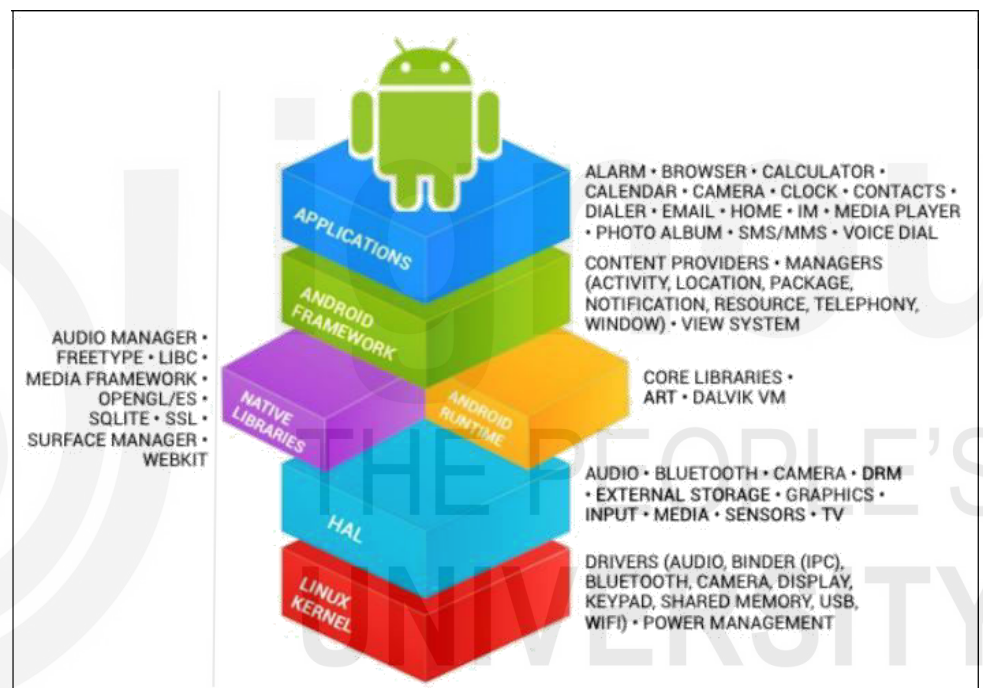


Figure 2.1: Android System Architecture

(Source: <https://source.android.com/source/index.html>)

Let us discuss each layer one by one.

### Linux Kernel

Android is designed on the top of Linux Kernel which is an open source operating system as explained in the previous unit.

Basic services like process management, memory management, security management, power management and providing hardware driver for different devices (like Bluetooth, WI-FI, Camera etc.) are managed by Linux Kernel as in Figure 2.2.



Figure 2.2: Linux kernel

(Source: <http://androidfulcrum.blogspot.com/2014/02/what-is-android-architecture.html>)

Linux Kernel never really interacts with the users and developers, but is at the heart of the whole system. Its importance stems from the fact that it provides the following functions in the Android system:

- Hardware abstraction
- Memory management programs
- Security settings
- Power management software
- Other hardware drivers (Drivers are programs that control hardware devices.)
- Support for shared libraries
- Network stack

In this section we have learned how Linux is involved with Android. Now you need to complete the given activity before reading further in this unit.

### 2.2.1 Check Your Progress



Explain the role of Linux Kernel in Android

#### Hardware Abstraction Layer (HAL)

The Hardware Abstraction Layer (HAL) provides an interface for hardware vendors to define and implement the drivers for specific hardware without affecting lower level features. HAL implementations are packaged into modules, you will find these as files with .so extension and loaded by the Android system at the appropriate time. Figure 2.3 shows the components of Hardware abstraction layer (HAL).

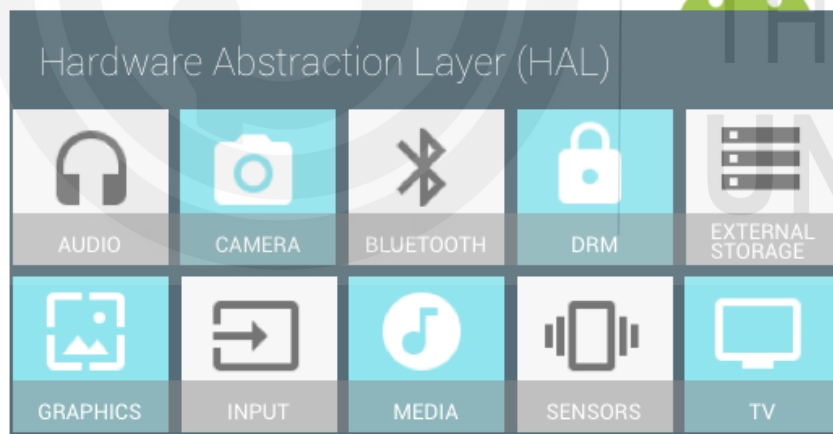


Figure 2.3: Hardware abstraction layer (HAL) components

(Source: <https://source.android.com/source/index.html>)

#### Native Libraries

Native libraries run over the HAL and it consist various C / C++ library like libc. It also includes following standard libraries:

- **Secure Sockets Layer (SSL):** It is responsible for Internet security.
- **Graphics Library:** OpenGL and SGL used to create 2D and 3D graphics.
- **WebKit:** It is open source web browser engine that gives the functionality to render the web content.

- **SQLite:** This is an open source relational database management system which is designed to be embedded in Android devices.
- **Media Library:** These libraries are used to play the audio/video media etc.

The following Figure 2.4 shows the Android library layer and its components.



**Figure 2.4: Android Libraries Layer**

(Source: <http://androidfulcrum.blogspot.com/2014/02/what-is-android-architecture.html>)

Libraries carry a set of instructions to guide the device in handling different types of data. For instance, the playback and recording of various audio and video formats is guided by the Media Framework Library. This layer enables the device to handle different types of data with the use of these libraries.

### **Dalvik Virtual Machine (DVM)**

It is a modified Java virtual machine (JVM) which is introduced for low end devices to run application objects efficiently. It is a register based virtual machine that is optimized to run multiple objects efficiently. It depends on the Linux kernel for efficiently execute the instances because memory management and thread management is part of the Linux kernel. DVM executes the Dalvik Executable Code (.dex), which is optimized to take least memory and processing resources.

### **Android Run Time**

Android Runt time (ART) includes Dalvik Virtual Machine (DVM) and Core Libraries which help your apps to run on an Android mobile device.



**Figure 2.5: Android Runtime**

(Source: <http://androidfulcrum.blogspot.com/2014/02/what-is-android-architecture.html>)

ART is the successor of Dalvik Virtual machine. ART is the managed runtime system that helps to run applications and system services. ART and its predecessor Dalvik were originally created specifically for the Android project. ART is compatible with DVM, so it helps to run Dalvik Executable codes. 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 has the following features:

- Ahead-of-time (AOT) compilation
- Improved garbage collection
- Improvements in development and debugging

## Core Libraries

Core libraries are the collection of Android specific core java libraries. You will be using these libraries when writing your Android applications in later units.

### 2.2.2 Video V3: Android Architecture



What you have read upto now regarding Android Architecture is presented in this video. After watching the video you may attemptbActivity 2.2.



URL: <https://tinyurl.com/ya7t24et>

### 2.2.3 Check Your Progress



Describe the use of Dalvik Virtual Machine and Android Runtime in Android operating systems.

## Android Application Framework

Application framework contains the classes used to create an Android application. This behaves as an abstraction layer for hardware access. It also manages application resources and user interface. Content provider, activity manager, fragment manager, telephony manager, location manager, package manager, notification manager and view system are the parts of Android Application Framework.

## Application Layer

Figure 2.6 shows the application layer which is the top layer of Android architecture. Every application (e.g. Contacts, Browsers, etc.), whether it is a native application or a third party application, runs in Application layer. Preinstalled applications provided by the vendors are called native apps and applications developed by another developer are called third party applications. In application layer, third party apps can replace the native apps. This is the beauty of Android.

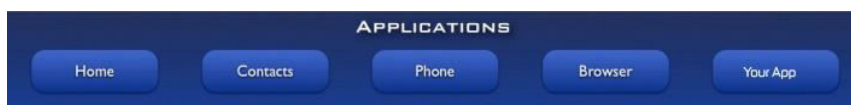


Figure 2.6: Android application layer

(Source: <http://androidfulcrum.blogspot.com/2014/02/what-is-android-architecture.html>)



---

## 2.3 TYPES OF MOBILE APPLICATIONS

---

It is obvious that mobile technologies have been evolving rapidly in recent years, with a huge amount of creativity on mobile application development. As a result, there are more prominent needs than at any other time for a mobile application client platform which can not only deliver mobile-optimized user experience, but also support the increasingly complex business logic that the application must support.

It is important to consider various factors related to the purpose of the developing the app when deciding between which types of app you should build. Therefore, it is important to check your current priorities and where you want to be in the future before selecting the app. The quality of the user experience you need your app to have, the complexity of the features you need for your app to work and the budget, helps you choose which approach is best.

Today, we often speak of three types of mobile applications according to how they are developed: native, web and hybrid. Further explanations on these types are given below.

### Native apps

Those that are fully programmed in the development environment specific to each operating system.

Native applications can leverage the full array of features and functions available through the mobile device's core operating system. Generally, they are faster and offer a significantly usable interface than the others.

Advantages:

- Smooth performance
- Good user experience
- App icon available on the device
- Can receive push notifications
- Runs inside the operating system
- Can use the platform APIs

Disadvantages:

- Developers need to know each of the platforms languages
- Source code only works on the targeted platform
- Slower to market due to multiple source codes

### Web apps

Fully developed in HTML 5, Mobile Web apps offer an attractive option for companies that do not want to invest in building native applications across four different mobile platforms. Whether getting a new application up and running or maintaining or updating an existing mobile solution with Mobile Web Apps is simple and inexpensive. Better yet, HTML5-driven mobile Web apps are cross-platform compatible and, more secure than native applications (given that very little data is stored locally on the native device.)

**Advantages:**

- Cross platform
- Single code base
- Fast to production
- Lower development cost

**Disadvantages:**

- Sluggish performance
- Require loading
- Network connection required
- Not available in the app stores
- Extremely limited API access Lives in the browser

**Hybrid apps**

Apps developed partly with the native development environment and partly in WEB language (HTML 5).

Today, technology changes so rapidly that most businesses require immense flexibility and scalability to adapt content, design and even application architecture. By deploying applications that rely on a robust combination of HTML5 Web technologies and native operating system features, you preserve a large degree of control over the content and design of the solutions we build for mobile platforms.

Many companies find that this hybrid development process empowers them to perform fast, easy, on-demand updates, without losing the inherent advantages that come from hosting a solution in the iTunes Apps Store or the Android Marketplace.

**Advantages:**

- Single source code
- Access to all platforms
- Less time for deployment
- Available in app store
- Has application icon on the device

**Disadvantages:**

- Dependent on such as phone gap
- Middleware may be slow to update
- More bug prone
- Some bug fixes need middleware updates
- Some bug fixes are outside of your control
- Slower performance
- More issues from device fragmentation

Each has its positives and negatives that can and should influence the decision when making a choice for development. Which is most appropriate will vary based on your

particular requirements. The Table 2.1 shows a summary of the features discussed in each type.

**Table 2.1: Hybrid vs. Native vs. Mobile Web**

Feature	Native	HTML5	Hybrid
Graphics	Native APIs	HTML, Canvas, SVG	HTML, Canvas, SVG
App performance	Fast	Moderate	Moderate
Distribution	App Store/Market	Web	App Store/Market
Native look and feel	Native	Emulated	Emulated
Camera	Yes	No	Yes
Push Notifications	Yes	No	Yes
File upload	Yes	Yes	Yes
Contacts, calendar	Yes	No	Yes
Connectivity	Online and offline	Mostly online	Online and offline
Development skills needed	XML, Java	HTML5, CSS, Javascript	HTML5, CSS, Javascript
Geolocation	Yes	Yes	Yes

### 2.3.1 Check Your Progress



Differentiate native, web, and hybrid mobile apps by stating the advantages, disadvantages and special features.

## 2.4 APPLICATION FUNDAMENTALS

Android apps are written in the Java programming language. The Android SDK tools compile your code along with any data and resource files into an APK: an Android package, which is an archive file with an .apk suffix. One APK file contains all the contents of an Android app and this is the file that Android-powered devices use to install the app.

- Once installed on a device, each Android app lives in its own security sandbox
- The Android operating system is a multi-user Linux system in which each app is a different user.
- By default, the system assigns each app a unique Linux user ID (the ID is used only by the system and is unknown to the app). The system sets permissions for all the files in an app so that only the user ID assigned to that app can access them.
- Each process has its own virtual machine (VM), so an app's code runs in isolation from other apps.
- By default, every app runs in its own Linux process. Android starts the process when any of the app's components need to be executed, then shuts down the

process when it's no longer needed or when the system must recover memory for other apps.

In this way, the Android system implements the principle of least privilege. That is, each app, by default, has access only to the components that it requires to do its work and no more. This creates a very secure environment in which an app cannot access parts of the system for which it is not given permission.

However, there are ways for an app to share data with other apps and for an app to access system services:

- It's possible to arrange for two apps to share the same Linux user ID, in which case they are able to access each other's files. To conserve system resources, apps with the same user ID can also arrange to run in the same Linux process and share the same VM (the apps must also be signed with the same certificate).
- An app can request permission to access device data such as the user's contacts, SMS messages, the mountable storage (SD card), camera, Bluetooth, and more. The user has to explicitly grant these permissions.

That covers the basics regarding how an Android app exists within the system.

The rest of this unit introduces you to:

- The core framework components that define your app.
- The manifest file in which you declare components and required device features for your app.
- Resources that are separate from the app code and allow your app to gracefully optimize its behaviour for a variety of device configurations.

App components are the essential building blocks of an Android app. Each component is a different point through which the system can enter your app.

Not all components are actual entry points for the user and some depend on each other, but each one exists as its own entity and plays a specific role—each one is a unique building block that helps define your app's overall behaviour.

There are four different types of app components. Each type serves a distinct purpose and has a distinct lifecycle that defines how the component is created and destroyed. Here are the four types of application components. These four components are further explained in unit 5.

## Activities

An activity represents a single screen with a user interface. For example, an email app might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. Although the activities work together to form a cohesive user experience in the email app, each one is independent of the others. As such, a different app can start any one of these activities (if the email app allows it). For example, a camera app can start the activity in the email app that composes new mail, in order for the user to share a picture.

An activity is implemented as a subclass of Activity and you can learn more about activity lifecycle in the next unit.

## Services

A service is a component that runs in the background to perform long-running operations or to perform work for remote processes. A service does not provide a user interface. For example, a service might play music in the background while the user is in a different app, or it might fetch data over the network without blocking user interaction with an activity. Another component, such as an activity, can start the service and let it run or bind to it in order to interact with it.

A service is implemented as a subclass of `Service` and you can learn more about it in the Services developer guide.

## Content providers

A content provider manages a shared set of app data. You can store the data in the file system, SQLite database, on the web, or any other persistent storage location your app can access. Through the content provider, other apps can query or even modify the data (if the content provider allows it). For example, the Android system provides a content provider that manages the user's contact information. As such, any app with the proper permissions can query part of the content provider (such as contact data) to read and write information about a particular person.

Content providers are also useful for reading and writing data that is private to your app and not shared. For example, the Note Pad sample app uses a content provider to save notes.

A content provider is implemented as a subclass of `ContentProvider` and must implement a standard set of APIs that enable other apps to perform transactions. For more information, see the Content Providers developer guide.

## Broadcast receivers

A broadcast receiver is a component that responds to system-wide broadcast announcements. Many broadcasts originate from the system—for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured. Apps can also initiate broadcasts—for example, to let other apps know that some data has been downloaded to the device and is available for them to use.

Although broadcast receivers don't display a user interface, they may create a status bar notification to alert the user when a broadcast event occurs. More commonly, though, a broadcast receiver is just a "gateway" to other components and is intended to do a very minimal amount of work. For instance, it might initiate a service to perform some work based on the event.

---

## 2.5 SUMMARY

---



The objective of this study unit was to introduce the architecture of Android. In the first part of this study unit, we discussed about the different types of layers of Android operating system and the role of each layer with functionalities. Further, it discussed about the three types of mobile applications: native, web and hybrid. The rest of the unit discussed about the fundamental components of an Android application.

---

## 2.6 FURTEHR READINGS

---

<https://source.android.com/devices/> (CC-BY)

<http://androidfulcrum.blogspot.com/2014/02/what-is-android-architecture.html> (CC-BY)

<https://source.android.com/source/index.html> (CC-BY)





---

## UNIT 3    ACTIVITY LIFECYCLE

---

- 3.0 Introduction
  - 3.1 Objectives
  - 3.2 What is an Activity in Android?
    - 3.2.1 Check Your Progress
    - 3.2.2 Check Your Progress
  - 3.3 What is an Activity Lifecycle?
    - 3.3.1 Video-V4: Android Application Fundamentals
  - 3.4 What are the Android process states?
    - 3.4.1 Check Your Progress
  - 3.9 Summary
  - 3.10 Further readings
- 

### 3.0 INTRODUCTION

---

In this unit you will be exploring the Activity Lifecycle of an Android application. After an introduction to *Activity*, *Activity life cycle* and *life cycle methods* are discussed. By watching the screen cast developed for this particular unit, you will get a better understanding of how activity lifecycle works. The demonstration on how to determine the inter-process dependencies at runtime will also be further explained in the screencast.

---

### 3.1 OBJECTIVES

---



#### Outcomes

- Sketch the activity life cycle diagram and identify the components.
- List the status of the activity life cycle and describe each status related to the working mobile application.
- Explain the process of the activity life cycle, foreground, visible, background and empty processes.

After studying this unit, you should be able to:



#### Terminology

- |                    |   |
|--------------------|---|
| <b>activity:</b>   | Activity is an application component that provides a screen with which users can interact |
| <b>life cycle:</b> | journey of an Activity passing through different stages of life                           |

---

### 3.2 WHAT IS AN ACTIVITY IN ANDROID?

---

An Activity in Android is an application component that provides a screen with which users can interact in order to do something, such as dial the phone, take a photo, send an email, or view a map as explained in the previous unit. Each activity is given a



window in which to draw its user interface. The window typically fills the screen, but may be smaller than the screen and float on top of other windows.

When you open an application, the very first screen that appears in front of you, is called a default Activity (or Main Activity). There can be more than one loosely coupled Activities in your application. Generally, as the complexity of an application increases, the need of number of Activities increases proportionally.

### 3.2.1 Check Your Progress



Give an example of an Activity of an Android application

Now you have an idea about what an Activity is. Let us look into more details about how Activities are interrelated. An Activity can start another Activity to perform some actions. If it does so then system stops the current Activity and preserves it in a stack called activity stack which will be discussed in the next section.

### Activity Stack

Typically, one activity in an application is specified as the "main" activity, which is presented to the user when launching the application for the first time. Each activity can then start another activity in order to perform different actions. Each time a new activity starts, the previous activity is stopped, but the system preserves the activity in a stack called activity stack or "back stack". It is illustrated in the Figure 3.1.

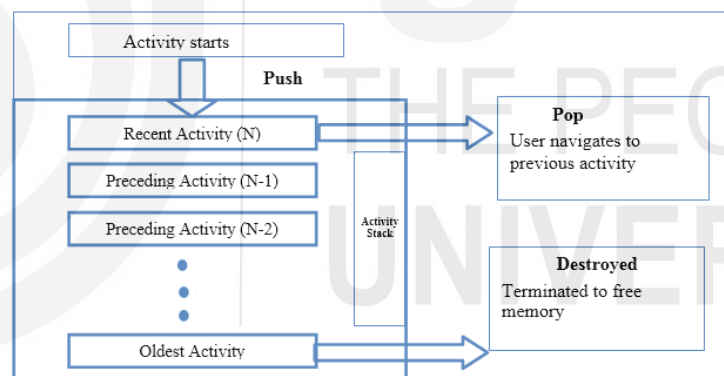


Figure 3.1: Activity Stack

When a new activity starts, it is pushed into the back stack and takes user focus. The back stack abides to the basic "last in, first out" stack mechanism, so that when the user is done with the current activity and presses the Back button, it is popped from the stack and the previous activity resumes. The old activities are destroyed by the OS to free up memory. But activity stack keeps activity reference and re-launch an activity when needed.

### 3.2.2 Check Your Progress



Draw and briefly explain, how each new activity in a task adds an item to the back stack by referring to the following link.

<https://developer.android.com/guide/components/tasks-and-back-stack.html>

Once an activity is launched, it goes through a lifecycle called Activity Lifecycle; a term that refers to the steps the activity progresses through as the user (and operating system) interacts with it.

### 3.3 WHAT IS AN ACTIVITY LIFECYCLE?

From its creation to its conclusion, an Activity goes through many stages of its life such as start, pause, resume, stop, etc. This journey of Activity passing through different stages of life called lifecycle of the Activity. Every stage of the Activity lifecycle has a specific method associated to it, called call-back method. When an Activity stops and another starts, it means a call-back method is called for each Activity. The lifecycle of an Activity is managed by the Android run time system.

Generally, an Activity can remain in following four states:

- If an activity is in the foreground of the screen (at the top of the stack), it is *active* or *running*.

If an activity has lost focus but is still visible (that is, a new non-full-sized or transparent activity has focus on top of your activity), it is paused. A paused activity is completely alive (it 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.

- If an activity is completely obscured by another activity, it is *stopped*. It still retains all state and member information, however, it is no longer visible to the user so its window is hidden and it will often be killed by the system when memory is needed elsewhere.

- 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.

- Figures 3.2 depicts the state diagrams to represent the different stages of the Activity lifecycle with different call-back methods.

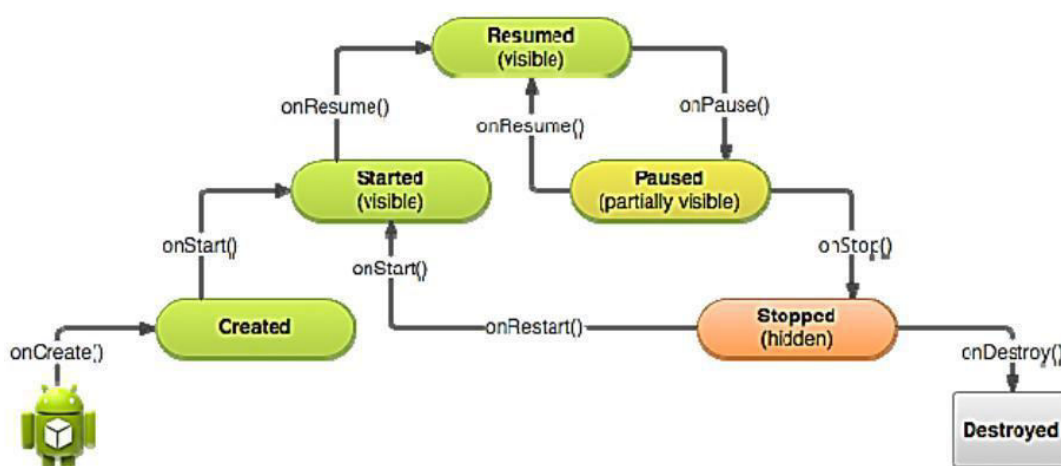


Figure 3.2: A simplified illustration of Activity Lifecycle

(Source: <http://developer.android.com/training/basics/activity-lifecycle/starting.html>)

### 3.3.1 Video-V4: Android Application Fundamentals



Let us watch this video and understand the android activity life cycle that moves to each state and respond to different call-back methods. This video will give you a detailed explanation of an Activity, Activity lifecycle, use of back-stack and the different call-back methods of Activity Lifecycle.



URL: <https://tinyurl.com/v7czgm5f>

Figure 3.3 illustrates the loops and the paths of an activity that might take place between states.

The rectangles represent the call-back methods you can implement to perform operations when the activity transitions between states. You can see that when a process is killed and the user navigates back to onCreate() method.

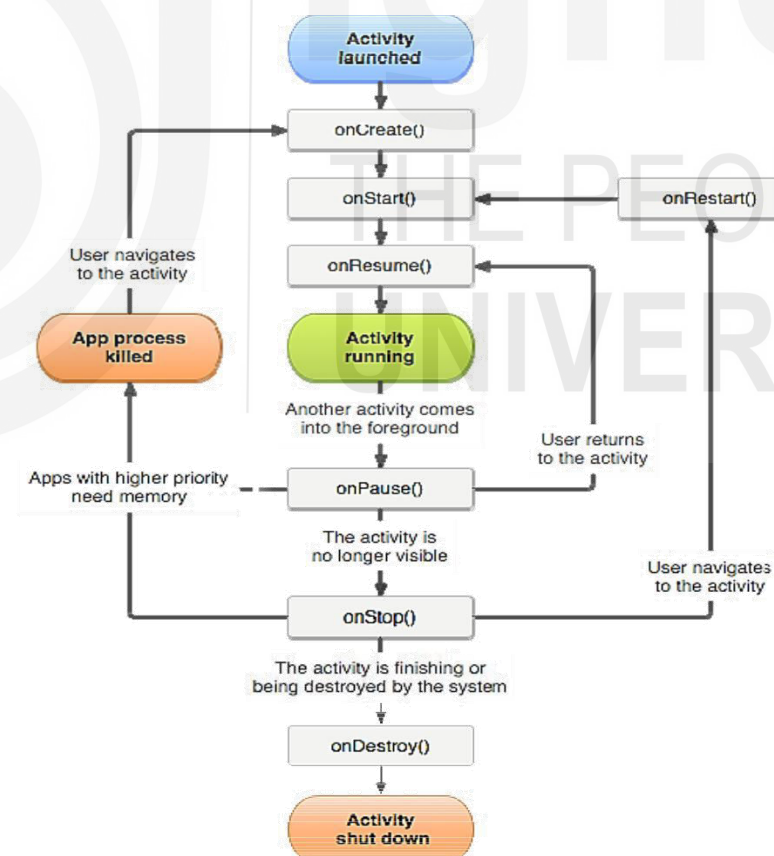


Figure 3.3: Activity Lifecycle

(Source: <http://developer.android.com/guide/components/activities.html>)

Now you have learnt the Activity Lifecycle and its states. Next, we are going to discuss how these states will be implemented as methods.

## Activity Lifecycle Methods

The Android system defines a lifecycle for activities via predefined lifecycle methods. The following Table 3.1 is showing each lifecycle call-back method with details such as name of the method, description of the method, what method is called after the specified method, method is killable or not, etc.

**Table 3.1: A summary of the activity lifecycle's call-back methods**

Methods	Description	Killable after?	Next
onCreate()	Called when the activity is first created. This is where you should do all of your normal static set up such as create views, bind data to lists, and so on.	No	onStart()
onRestart()	Called after the activity has been stopped, just prior to it being started again.	No	onStart()
onStart()	Called just before the activity becomes visible to the user.	No	onResume() or onStop()
onResume()	Called just before the activity starts interacting with the user. At this point the activity is at the top of the activity stack.	No	onPause()
onPause()	Called when the system is about to start resuming another activity.	Yes	onResume() or onStop()
onStop()	Called when the activity is no longer visible to the user. This may happen because it is being destroyed, or because another activity (either an existing one or a new one) has been resumed and is covering it.	Yes	onStart() or onDestroy()
onDestroy()	Called before the activity is destroyed. This is the final call that the activity will receive	Yes	Nothing

(Source: <http://developer.android.com/guide/components/activities.html>)

Within the lifecycle call-back methods, you can declare how your activity behaves when the user leaves and re-enters the activity. For example, if you're building a streaming video player, you might pause the video and terminate the network connection when the user switches to another app. When the user returns, you can reconnect to the network and allow the user to resume the video from the same spot.

To get the detailed information of working of an Activity, pursue the following link:

<http://developer.android.com/guide/components/activities.html>

With the understating of the Activities, Activity Lifecycle and call-back methods, we can discuss how these Activities are managed in the Android application.

### Managing Activities in the application

All Android applications started by the user are remained in memory, which makes restarting applications faster. But in reality the available memory on an Android device is limited. To manage these limited resources the Android system is allowed to terminate running processes or recycling Android components.

As stated earlier Android applications run within instances of the Dalvik virtual machine with each virtual machine being viewed by the operating system as a separate process. If the system identifies that resources on the device are reaching capacity it will take steps to terminate processes to free up its memory.

If the Android system needs to free up resources it follows logic. Every process gets a priority. If the Android system needs to terminate processes it follows the priority system. You will learn the Android process states and priority system in next section.

## 3.4 WHAT ARE THE ANDROID PROCESS STATES?

When deciding as to which process to terminate in order to free up memory, the system consider both the priority and state of all currently running processes. It is considered by Google as an important hierarchy. Processes are then terminated starting with the lowest priority and working up the hierarchy until sufficient resources have been liberated for the system to function. As outlined in Figure 3.4, a process can be in one of the following five states at any given time of priority.

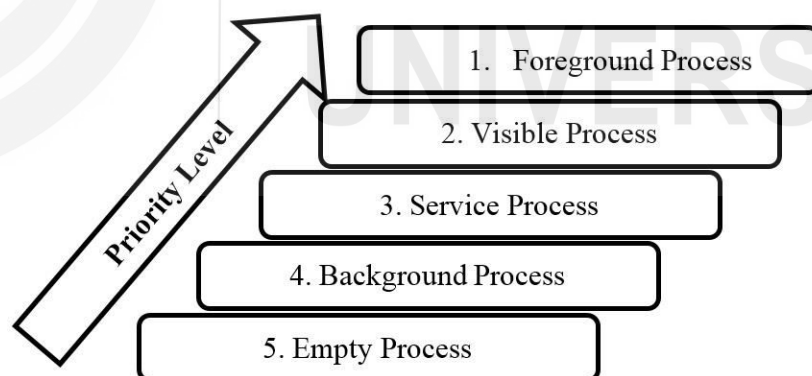


Figure: 3.4: Android process states and priority levels

Let us explain the priority levels more in details below.

### Foreground Process

These processes are assigned the highest level of priority. It is one that is required for what the user is currently doing. Various application components can cause its containing process to be considered foreground in different ways. A process is considered to be in the foreground if any of the following conditions hold:

- Hosts an activity with which the user is currently interacting.
- Hosts a Service connected to the activity with which the user is interacting.
- Hosts a Service that has indicated, via a call to `startForeground()`, that termination would be disruptive to the user experience.
- Hosts a Service executing either its `onCreate()`, `onResume()` or `onStart()` callbacks.
- Hosts a Broadcast Receiver that is currently executing its `onReceive()` method.

## Visible Process

It is a process containing an activity that is visible to the user but is not the activity with which the user is interacting. Such a process is considered extremely important and will not be killed unless doing so is required to keep all foreground processes running. This may occur, for example, if the foreground Activity is displayed as a dialog that allows the previous Activity to be seen behind it.

## Service Process

A process that contain a Service that has already been started and is currently executing is called a service process. However, these processes are not directly visible to the user. These processes are generally doing things that the user cares about such as background mp3 playback or background network data upload or download. The system will always keep such processes running unless there is not enough memory to retain all foreground and visible processes.

## Background Process

It is a process that contains one or more activities that are not currently visible to the user, and does not host a Service that qualifies for Service Process status. These processes have no direct impact on the user experience. Provided they implement their Activity life-cycle correctly, the system can kill such processes at any time to reclaim memory for one of the three previous processes types.

## Empty Process

Empty processes no longer contain any active application component. They are held in memory ready to serve as hosts for newly launched applications. Such processes are, obviously, considered to be the lowest priority and are the first to be killed to free up resources.

### 3.4.1 Check Your Progress

Select True/False statements

You can select A) True or B) False for the following statements



1. There is no guarantee that an activity will be stopped prior to being destroyed.
2. During an activity lifecycle, `onStart()` is the first callback method invoked by the system.
3. `Finish()` method is used to close an activity.
4. Once the `onStop()` method is called, the activity is no longer visible.
5. When `onPause()` method is called in an activity, another activity gets into the foreground state.

Hint check your answers with Answer guide at the end of this unit.

Now you have learnt how activities are managed. The situation of deciding the highest priority process is sometimes complex than outlined in the previous section since that processes can often be inter-dependent. Inter-Process Dependencies are explained further in the following section.

### Inter-Process Dependencies

Android system will also take into consideration that whether the process is in some way serving another process of higher priority. Likewise, when deciding as to the priority of a process the system consider inter-dependencies.

For instance, consider a service process acting as the content provider for a foreground process. The Android documentation states that a process can never be ranked lower than another process if it is currently serving a foreground process. Thus, the systems manage the activities and memory facilitating fast running applications.

(Source: <https://developer.android.com/training/articles/memory.html> )

---

## 3.5 SUMMARY

---



Mobile applications are common in day to day life. For instance, we can recall some of them as Calculator app, Date and activity schedule app, Map finding etc. All these application components provides a screen for user to interact and perform something with the mobile device. Those components are made of multiple activities providing various user interfaces. In this unit we described what an activity is and the activity lifecycle in a mobile application.

Activity lifecycle has few states and those states depend on the process states of it changes based on pre-defined priority levels. Moreover, activity lifecycle has methods which can be called upon wherever necessary. The rest of unit discussed the basics of how to build and use an activity.

---

## 3.6 FURTHER READINGS

---

- <https://developer.android.com/guide/components/activities/intro-activities>
- <https://developer.android.com/guide/components/activities/activity-lifecycle>
- <https://learncswithandroid.blogspot.com/2017/12/android-process-states.html>



---

## UNIT 4 ANDROID DEVELOPMENT ENVIRONMENT

---

- 4.0 Introduction
- 4.1 Objectives
- 4.2 Reasons for Android Development
- 4.3 Android Development Platforms, Features and Tools
  - 4.3.1 Check Your Progress
- 4.4 Configuring Android Development Environment
  - 4.4.1 Video - V5: Setting Up Android Development Environment
  - 4.4.2 Check Your Progress
  - 4.4.3 Video-V6: Install Android for Windows 10
- 4.5 Summary
- 4.6 Further readings

---

### 4.0 INTRODUCTION

---

In this unit you will be able to get familiar with available Android development platforms. You need to watch the video and install the required tools to start developing your first Android application.

---

### 4.1 OBJECTIVES

---

Upon completion of this unit you should be able to:



#### Outcomes

- explain development platforms and distinguish each against their features and capabilities.
- describe the background, and platform versions, system features, Android tools for the development environment.
- configure the Android development environment in a computer.



#### Terminology

<b>command line:</b>	commands entered as inputs without IDE
<b>SDK:</b>	Software Development Kit
<b>JDK:</b>	Java Development Kit
<b>Android Studio:</b>	Official Android platform to develop Android apps

---

### 4.2 REASONS FOR ANDROID DEVELOPMENT

---

Today, mobile telephones have fundamentally changed the way of people interact. It is evident that mobile applications will be the future of handheld devices, Television and Automobile. Moreover, developers have started embedding Android in home appliances and other devices.



There are many reasons for the popularity of Android apps, such as:

- Google provides one window solution, as Play Store, to upload and download the application either free or with minimal charges. For uploading and distributing the app, developers have no need of any approval of someone.
- Developer is the owner of his / her app and has the total control on product. However, Google has rights to unpublish any Android application in play store, if it is not complying with Google's licenses. For instance, if application contains malicious code or violate license, Google has right to unpublish the application.
- Android has open source operating system, open source software development kit (SDK) and good documentation.
- Android applications are not limited to mobile devices (Phones & Tabs), but can be run on TVs, wearable devices, vehicles and even refrigerators.
- (Source: First Thrust Towards Android, Android Programming, Course Material for Open Distance Learning, Commonwealth of Learning 2016)

---

### 4.3 ANDROID DEVELOPMENT PLATFORMS, FEATURES AND TOOLS

---

In unit 2, you have learned Android architecture and major components of the Android platform. Let's look at the Android platform and the features they are providing.

Android Studio is the official IDE for Android development, and with a single download it includes everything you need to begin developing Android apps as you can see below

- IntelliJ IDE + Android Studio plugin
- Android SDK Tools
- Android Platform-tools
- A version of the Android platform
- Android Emulator with an Android system image including Google Play Services

Android Studio provides tools for building apps on every type of Android device. Code editing, debugging, performance tooling, a flexible build system, and an instant build or deploy system are included in Android studio. Let's see what are the systems requirements to install Android studio in different operating systems.

#### System Requirements

System requirements for Windows, Mac OS and Linux are given below.

#### Windows - Microsoft® Windows® 7/8/10 (32- or 64-bit)

- 3 GB RAM minimum, 8 GB RAM recommended; plus 1 GB for the Android Emulator
- 2 GB of available disk space minimum,
- 4 GB Recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image)

- 1280 x 800 minimum screen resolution
- For accelerated emulator: 64-bit operating system and Intel® processor with support for Intel® VT-x, Intel® EM64T (Intel® 64), and Execute Disable (XD) Bit functionality

Mac - Mac® OS X® 10.10 (Yosemite) or higher, up to 10.12 (macOS Sierra)

- 3 GB RAM minimum, 8 GB RAM recommended; plus 1 GB for the Android Emulator
- 2 GB of available disk space minimum,
- 4 GB Recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image)
- 1280 x 800 minimum screen resolution

Linux - GNOME or KDE desktop

- Tested on Ubuntu® 12.04, Precise Pangolin (64-bit distribution capable of running 32-bit applications)
- 64-bit distribution capable of running 32-bit applications
- GNU C Library (glibc) 2.19 or later
- 3 GB RAM minimum, 8 GB RAM recommended; plus 1 GB for the Android Emulator
- 2 GB of available disk space minimum,
- 4 GB Recommended (500 MB for IDE + 1.5 GB for Android SDK and emulator system image)
- 1280 x 800 minimum screen resolution

For accelerated emulator: Intel® processor with support for Intel® VT-x, Intel® EM64T (Intel® 64), and Execute Disable (XD) Bit functionality, or AMD processor with support for AMD Virtualization™ (AMD-V™).

(Source: <https://source.android.com/source/requirements.html>, CC:BY: 2.5)

## Command Line Tools

The Android SDK tools available from the SDK Manager provide additional command-line tools to help you during your Android development. The tools are classified into two groups: SDK tools and platform tools. SDK tools are platform independent and are required no matter which Android platform you are developing on. Platform tools are customized to support the features of the latest Android platform.

## Additional Command Line Tools

The Android SDK tools available from the SDK Manager provide additional command-line tools to help you during your Android development. The tools are classified into two groups: SDK tools and platform tools. SDK tools are platform independent and are required no matter which Android platform you are developing on. Platform tools are customized to support the features of the latest Android platform.

## **SDK Tools**

The SDK tools are installed with the SDK starter package and are periodically updated. The SDK tools are required if you are developing Android applications. The most important SDK tools include the Android SDK Manager (Android sdk), the AVD Manager (Android AVD) the emulator (emulator), and the Dalvik Debug Monitor Server (DDMS). A short summary of some frequently-used SDK tools is provided below.

## **Virtual Device Tools**

### **a) Android Virtual Device Manager**

The AVD Manager provides a graphical user interface in which you can create and manage Android Virtual Devices (AVDs) that run in the Android Emulator.

### **b) Android Emulator (emulator)**

A QEMU(short for quick emulator) based device emulation tool that you can use to debug and test your applications in an actual Android run-time environment.

### **c) mksdcard**

Helps you create a disk image that you can use with the emulator, to simulate the presence of an external storage card (such as an SD card).

## **Development Tools**

Hierarchy Viewer (hierarchyviewer) - Provides a visual representation of the layout's View hierarchy with performance information for each node in the layout, and a magnified view of the display to closely examine the pixels in your layout.

## **SDK Manager**

SDK Manager lets you manage SDK packages, such as installed platforms and system images.

sqlite3 - Lets you access the SQLite data files created and used by Android applications.

## **Debugging Tools**

The debugging tools are further explained in the later units of this material.

### **a) Android Monitor**

Android Monitor is integrated into Android Studio and provides logcat, memory, CPU, GPU, and network monitors for app debugging and analysis.

### **b) adb**

Android Debug Bridge (adb) is a versatile command line tool that lets you communicate with an emulator instance or connected Android-powered device. It also provides access to the device shell.

### **c) Dalvik Debug Monitor Server (DDMS)**

DDMS lets you debug Android apps.

### **d) Device Monitor**

Android Device Monitor is a stand-alone tool that provides a graphical user interface for several Android application debugging and analysis tools.

#### **e) Systrace**

This tool lets you analyze the execution of your application in the context of system processes, to help diagnose display and performance issues.

#### **f) traceview**

Provides a graphical viewer for execution logs saved by your application.

#### **g) Tracer for OpenGL ES**

Allows you to capture OpenGL ES(the standard for Embedded Accelerated 3D Graphics) commands and frame-by-frame images to help you understand how your app is executing graphics commands.

### **Build Tools**

#### **a) apksigner**

Signs APKs and checks whether APK signatures will be verified successfully on all platform versions that a given APK supports.

#### **b) JOBB**

Allows you to build encrypted and unencrypted APK expansion files in Opaque Binary Blob (OBB) format.

#### **c) ProGuard**

Shrinks, optimizes, and obfuscates your code by removing unused code and renaming classes, fields, and methods with semantically obscure names.

#### **d) zipalign**

Optimizes APK files by ensuring that all uncompressed data starts with a particular alignment relative to the start of the file.

### **Image Tools**

#### **a) Draw 9-patch**

Allows you to easily create a NinePatch (class permits drawing a bitmap in nine or more sections) graphic using a WYSIWYG (What You See Is What You Get) editor. It also previews stretched versions of the image, and highlights the area in which content is allowed.

#### **b) Etc1tool**

A command line utility that lets you encode PNG images to the ETC1 compression standard and decode ETC1 compressed images back to PNG.

### **Platform Tools**

The platform tools are typically updated every time you install a new SDK platform. Each update of the platform tools is backward compatible with older platforms. Usually, you directly use only one of the platform tools—the Android Debug Bridge (adb). Android Debug Bridge is a versatile tool that lets you manage the state of an

emulator instance or Android-powered device. You can also use it to install an Android application (APK) file on a device.

The other platform tools, such as `aidl`, `aapt`, `dexdump`, and `dx`, are typically called by the Android build tools, so you rarely need to invoke these tools directly. As a general rule, you should rely on the build tools to call them as needed.

Note: The Android SDK provides additional shell tools that can be accessed through `adb`, such as `bmgr` and `logcat`.

#### **a ) bmgr**

A shell tool you can use to interact with the Backup Manager on Android devices supporting API Level 8 or greater.

#### **b) logcat**

Provides a mechanism for collecting and viewing system debug output.

(Source: <https://developer.android.com/studio/command-line/index.html> CC:BY: 2.5)

Now, you have and learnt the systems requirements (hardware/software features) to set up the Android development platform and the Android command line tools. It is vital to determine the specific features of each Android version and how it has been developed to performing better. In unit 1 we learnt the history of Android and how each version of Android evolved. Next, we will summarize the Android platform versions with their unique features.

### **Android platform versions and specific features**

There are rapid developments and new versions for the Android and Table 4.1 summarize the specific features of different Android platform versions up to now.

**Table 4.1 summary the specific features of different Android platform versions**

Android Platform version	Specific Features
Android 1.6 - Donut	Quick search box Screen size diversity Android market
Android 2.1 - Eclair	Google maps navigation Home screen customization Speech-to- Text
Android 2.2- Froyo	Voice actions Portable Hotspot Performance
Android 2.3- Gingerbread	Gaming APIs

	Near Field Communication (NFC) Battery Management
Android 3.0- Honeycomb	Tablet-friendly design System bar Quick settings
Android 4.0- Ice cream Sandwich	Custom home screen Data usage control Android Beam
Android 4.1- Jelly Bean	Google now Actionable Notifications Account switching
Android 4.4- Kitkat	Voice: OK Google Immersive Design Smart Dialer
Android 5.0- Lollipop	Material Design Multiscreen Notifications
Android 6.0- Marshmallow	Now on tap Permissions Battery works smarter
Android 7.0 - Nougat	Multi Locale language settings Multi-window view Quick switch between apps Data Saver Notification Controls Display Size

You can read more information online about the relative numbers of devices that are running different versions in the following link.

<https://developer.android.com/about/dashboards/index.html>

Thus, creating apps in Android for various mobile devices are increasing day by day. Since developing native apps is expensive, the demand for cross platform app development tools is also increasing. It is essential to know cross platforms and tools for mobile application development to develop apps for enhancing the market capacity.

You can watch the online presentation given in the link below to get familiar with the cross platforms for mobile application development.

[bit.ly/XPlatformMobileDev](http://bit.ly/XPlatformMobileDev)

### 4.3.1 Check Your Progress



**Explore most popular cross platforms and name three major cross platforms.**

**Briefly describe one of the major cross platform with the features and uses of them.**

Hint: Check your answers with Answer guide at the end of this unit.

Now you are aware of the native and cross platform for mobile application development. We will now explore how the development environment is set up and configured.

---

## 4.4 CONFIGURING ANDROID DEVELOPMENT ENVIRONMENT

---

To develop an Android application, first you have to setup the Android development environment.

First download and install the Android Studio, Java and then download and install every individual tool (like Java, SDK Manager, DDMS tool, AVD Manager, etc.)

After installing Android tooling, it is possible to integrate development with various IDEs like, IntelliJ IDEA, Eclipse variants etc. Otherwise you can use any other java IDE or editor tool like notepad to compose the code.

### 4.4.1 Video -V5: Setting Up Android Development Environment



URL: <https://tinyurl.com/y9kcehov>



Let us now watch this video of setting up Android development environment set-by-step approach. This will help you at set up your Android development environment before you start programming with Android.

Next, we will be using Android Studio to develop the application. Android Studio is a native Android IDE that is fully dedicated to Android development. ADT Plugin needs to be installed to make it ready for Android development. Eclipse is an open source Java IDE that is compatible for several platforms. In early days Eclipse was the mostly used IDE by developers for Android applications.

## Download and install Java Development Kit (JDK)

Most of the programming of Android is done using the Java programming language. To download latest Java Development Kit, follow the link given below.

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

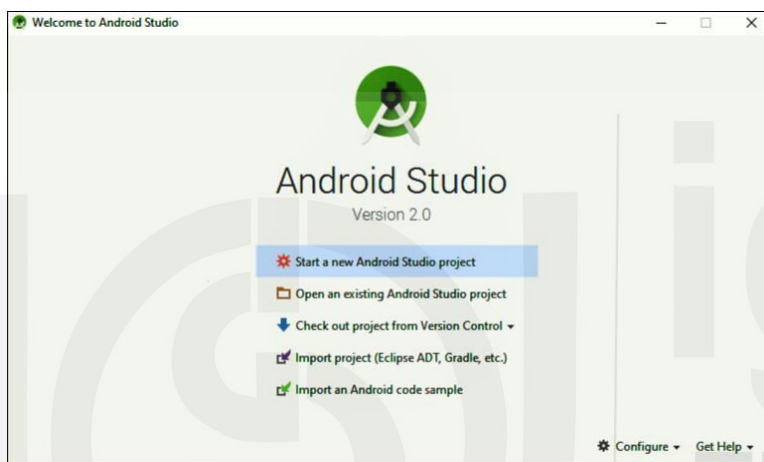
Now, extract the downloaded zip file and double click on the .exe and follow the instructions.

## Download and install Android studio

You can download the latest Android Studio from the following link:

<http://developer.android.com/sdk/index.html>

Download the executable file from the above mentioned link, double click on the file



and follow the installation instructions. To install the latest “Android Studio” you need to install compatible Java SE Development Kit first.

Online reading: You can pursue the following link to grab the installation instructions for Android Studio.

<http://developer.android.com/sdk/installing/index.html>

After installing and when you will start the Android Studio first time, the very first window will look like Figure 4.1.

**Figure 4.1: First time opening window of Android Studio**

When you click on the first option to start a new Android Studio Project, multiple windows will pop up one after another to setup a new project.

Online reading: Android Studio has numerous features. In this course, you are going to explore and use many features of Android Studio. For the prior reading to understand the Android studio, follow the following web link:

<http://developer.android.com/tools/studio/index.html>

Once you install Android Studio, it is easy to keep the Android Studio IDE and Android SDK tools up to date with automatic updates and the Android SDK Manager.

Following web page provides update instructions of IDE and Android SDK tools.



<https://developer.android.com/studio/intro/update.html#channels>

#### 4.4.2 Check Your Progress



State the most common tools used in Android application development

#### 4.4.3 Video-V6: Install Android for Windows 10



URL: <https://storage.googleapis.com/androiddevelopers/videos/studio-install-windows.mp4>

This video will show you how to install and configure the Android development environment using Android Studio in Windows 10.

Furthermore, the following link has videos that shows each step of the recommended setup procedure for Mac and Linux.

<https://developer.android.com/studio/install.html>

#### Lab Exercise:

Download and install JDK and the latest Android Studio version to configure the development environment.

Now, you are ready to go for making an Android App.

---

### 4.5 SUMMARY

---



The first step to start on Android based application development is to set up the development environment. Therefore, it is essential to know how to configure development environment and installing tools for Android application development.

In this unit, you have learned Android development platforms, tools and cross platforms. At the last section of the unit, you learnt about setting up the Android development environment by installing the latest JDK and Android Studio. It is also important to keep the Android studio IDE and Android SDK tools up to date at the development environment.

---

### 4.6 FURTHER READINGS

---

<https://developer.android.com/studio/intro/studio-config?authuser=1>

<https://developer.android.com/studio/install>

---

## UNIT 5    ANDROID APPLICATION FUNDAMENTALS

---

- 5.0 Introduction
- 5.1 Objectives
- 5.2 Basic App Components
  - 5.2.1 Video -V6: Android Development
  - 5.2.2 Check Your Progress
- 5.3 Additional Components
- 5.4 Resources
  - 5.4.1 Check Your Progress
- 5.5 Android Manifest
- 5.6 File conventions
  - 5.6.1 Check Your Progress
- 5.7 Summary
- 5.8 Further Readings

---

### 5.0    INTRODUCTION

---

In this unit we describe of the basic app components, additional components, resources and the manifest file using Android Studio, which is the Open Source platform provided for application developers. This unit also provides you an overview of the interaction and the association between the components and the application.

---

### 5.1    OBJECTIVES

---



#### Outcomes

- Explain the activity Components
- Outline the manifest file
- Explain the introduction to AVD
- Create an Android Virtual Device to simulate a device and to display on the development computer



#### Terminology

- |                  |   |
|------------------|---|
| <b>Manifest</b>  | file containing metadata for a group files that are part of a |
| <b>File:</b>     | coherent unit   |
| <b>Bound</b>     | Server in a client-server interface                           |
| <b>Services:</b> |   |
| <b>RPC:</b>      | Remote Procedure Call   |
| <b>API:</b>      | Application Programming Interface                             |

After studying this Unit, you should be able to:

---

## 5.2 BASIC APP COMPONENTS

---

As you already learnt in Unit 2, there are four types of app components and they are:

- Activities
- Services
- Content Providers
- Broadcast Receivers

### Activities

An activity is the entry point for interacting with the user. As explained in unit 2, it represents a single screen with a user interface. An activity facilitates the following key interactions between system and app:

- Keeping track of what the user currently cares about (what is on screen) to ensure that the system keeps running the process that is hosting the activity.
- Knowing that previously used processes contain things the user may return to (stopped activities), and thus more highly prioritize keeping those processes around.
- Helping the app handle having its process is killed so the user can return to activities with their previous state restored.
- Providing a way for apps to implement user flows between each other, and for the system to coordinate these flows. (The most classic example here being share.)

An activity is a single, focused thing that the user can do. Almost all activities interact with the user, so the Activity class takes care of creating a window for you in which you can place your UI with `setContentView(View)`. While activities are often presented to the user as full-screen windows, they can also be used in other ways: as floating windows (via a theme with `windowIsFloating` set) or embedded inside of another activity (using `ActivityGroup`).

There are two methods almost all subclasses of Activity will implement:

- `onCreate(Bundle)` is where you initialize your activity. Most importantly, here you will usually call `setContentView(int)` with a layout resource defining your UI, and using `findViewById(int)` to retrieve the widgets in that UI that you need to interact with programmatically.
- `onPause()` is where you deal with the user leaving your activity. Most importantly, any changes made by the user should at this point be committed (usually to the `ContentProvider` holding the data).

To be of use with `Context.startActivity()`, all activity classes must have a corresponding `<activity>` declaration in their package's `AndroidManifest.xml` as shown in Program 5.1. An activity must be implemented as a subclass of the Activity class.

```
public class MainActivity extends Activity {  
    /** Called when the activity is first created. */  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
    }  
}
```

```

}
/** Called when the activity is about to be visible.*/
@Override
protected void onStart() {
    super.onStart();
}
/** Called when the activity has become visible. */
@Override
protected void onResume() {
    super.onResume();
}
/** Called when another activity is taking focus. */
@Override
protected void onPause() {
    super.onPause();
}
/** Called when the activity is no longer visible. */
@Override
protected void onStop() {
    super.onStop();
}
/** Called just before the activity is destroyed. */
@Override
public void onDestroy() {
    super.onDestroy();
}

```

Program 5.1 Methods in Activity Class

Source(<http://www.androdevelopment.com/android-activities/>)

## Services

A service is a general-purpose entry point for keeping an app running in the background for all kinds of reasons. As stated in unit 2, it is a component that runs in the background to perform longrunning operations or to perform work for remote processes. Another component, such as an activity, can start the service and let it run or bind to it in order to interact with it. There are two very distinct semantics services that tell the system about how to manage an app: Started services tell the system to keep them running until their work is completed. This could be to sync some data in the background or play music even after the user leaves the app. Syncing data in the background or playing music also represents two different types of started services that modify how the system handles them:

- Music playback is something the user is directly aware of, so the app tells the system this by saying it wants to be foreground with a notification to tell the user about it; in this case the system knows that it should try really hard to keep that service's process running, because the user will be unhappy if it goes away.
- A regular background service is not something the user is directly aware as running, so the system has more freedom in managing its process. It may allow it to be killed (and then restarting the service sometime later) if it needs RAM for things that are of more immediate concern to the user.

Bound services run because some other app (or the system) has said that it wants to make use of the service. This is basically the service providing an API to another process. The system thus knows there is a dependency between these processes, so if process A is bound to a service in process B, it knows that it needs to keep process B

(and its service) running for A. Further, if process A is something the user cares about, then it also knows to treat process B as something the user also cares about. Because of their flexibility (for better or worse), services have turned out to be a really useful building block for all kinds of higher-level system concepts. Live wallpapers, notification listeners, screen savers, input methods, accessibility services, and many other core system features are all built as services that applications implement and the system binds to when they should be running.

To create a service, you must create a subclass of `Service` or use one of its existing subclasses. In your implementation, you must override some callback methods that handle key aspects of the service lifecycle and provide a mechanism that allows the components to bind to the service, if appropriate. You need to have prior knowledge of applying object oriented concepts to do this. These are the most important callback methods that you should override:

### **onStartCommand()**

The system invokes this method by calling `startService()` when another component (such as an activity) requests that the service be started. When this method executes, the service is started and can run in the background indefinitely. If you implement this, it is your responsibility to stop the service when its work is complete by calling `stopSelf()` or `stopService()`. If you only want to provide binding, you don't need to implement this method.

### **onBind()**

The system invokes this method by calling `bindService()` when another component wants to bind with the service (such as to perform RPC). In your implementation of this method, you must provide an interface that clients use to communicate with the service by returning an `IBinder`. You must always implement this method; however, if you don't want to allow binding, you should return `null`.

### **onCreate()**

The system invokes this method to perform one-time setup procedures when the service is initially created (before it calls either `onStartCommand()` or `onBind()`). If the service is already running, this method is not called.

### **onDestroy()**

The system invokes this method when the service is no longer used and is being destroyed. Your service should implement this to clean up any resources such as threads, registered listeners, or receivers. This is the last call that the service receives.

If a component starts the service by calling `startService()` (which results in a call to `onStartCommand()`), the service continues to run until it stops itself with `stopSelf()` or another component stops it by calling `stopService()`.

If a component calls `bindService()` to create the service and `onStartCommand()` is not called, the service runs only as long as the component is bound to it. After the service is unbound from all of its clients, the system destroys it.

Traditionally, there are two classes you can extend to create a started service named `Service` and `IntentService`.

The `Service` is the base class for all services (shown in Program 5.2). When you extend this class, it is important to create a new thread in which the service can complete all of its work; the service uses your application's main thread by default, which can slow the performance of any activity that your application is running.

```
public class HelloService extends Service {
    private Looper mServiceLooper;
    private ServiceHandler mServiceHandler;

    // Handler that receives messages from the thread
    private final class ServiceHandler extends Handler {
        public ServiceHandler(Looper looper) {
            super(looper);
        }
        @Override
        public void handleMessage(Message msg) {

            // Normally we would do some work here, like download a //file. For our
            // sample, we just sleep for 5 seconds.
            try {
                Thread.sleep(5000);
            } catch (InterruptedException e) {
                // Restore interrupt status.
                Thread.currentThread().interrupt();
            }

            // Stop the service using the startId, so that we don't stop //the
            // service in the middle of handling another job
            stopSelf(msg.arg1);
        }
    }

    @Override
    public void onCreate() {
        // Start up the thread running the service. Note that we //create
        // a separate thread because the service normally runs //in the
        // process's main thread, which we don't want to //block. We also make
        // it background priority so CPU-//intensive work will not disrupt our UI.

        HandlerThread thread
        = new HandlerThread("ServiceStartArguments",
            Process.THREAD_PRIORITY_BACKGROUND);
        thread.start();

        // Get the HandlerThread's Looper and use it for our //Handler
        mServiceLooper = thread.getLooper();
        mServiceHandler = new ServiceHandler(mServiceLooper);
    }
    @Override
    public int onStartCommand(Intent intent, int flags, int
        startId) {
```

```
        Toast.makeText(this, "service  
starting", Toast.LENGTH_SHORT).show();

// For each start request, send a message to start a // job and  
deliver the start ID so we know which // request we're  
stopping when we finish the job
        Message msg = mServiceHandler.obtainMessage();
        msg.arg1 = startId;
        mServiceHandler.sendMessage(msg);

// If we get killed, after returning from here, restart
        return START_STICKY;
    }
    @Override
    public IBinder onBind(Intent intent) {
        // We don't provide binding, so return null
        return null;
    }
    @Override
    public void onDestroy() {
        Toast.makeText(this, "service  
done", Toast.LENGTH_SHORT).show();
    }
}
```

#### Program 5.2 Service class declaration

Source(<https://developer.android.com/guide/components/services.html>)

The `IntentService` is a subclass of `Service` that uses a worker thread to handle all of the start requests, one at a time (Shown in Program 5.3). This is the best option if you don't require that your service handle multiple requests simultaneously. Implement `onHandleIntent()`, which receives the intent for each start request so that you can complete the background work.

```
public class HelloIntentService extends IntentService {
    /** A constructor is required, and must call the super
     * IntentService(String) constructor with a name for the worker
     * thread. */
    public HelloIntentService() {
        super("HelloIntentService");
    }

    /** The IntentService calls this method from the default worker
     * thread with the intent that started the service. When this
     * method returns, IntentService stops the service, as
     * appropriate. */
    @Override
    protected void onHandleIntent(Intent intent) {
        // Normally we would do some work here, like download a //file. For our
        // sample, we just sleep for 5 seconds.
        try {
```

```

        Thread.sleep(5000);
    } catch (InterruptedException e) {
        // Restore interrupt status.
        Thread.currentThread().interrupt();
    }
}
}

```

Program 5.3 Intent Service subclass declaration

Source (<https://developer.android.com/guide/components/services.html>)

## Broadcast receivers

As already stated in unit 2, broadcast receiver is a component that enables the system to deliver events to the app outside of a regular user flow, allowing the app to respond to system-wide broadcast announcements. Because broadcast receivers are another well-defined entry into the app, the system can deliver broadcasts even to apps that aren't currently running. So, for example, an app can schedule an alarm to post a notification to tell the user about an upcoming event and by delivering that alarm to a BroadcastReceiver of the app, there is no need for the app to remain running until the alarm goes off. Many broadcasts originate from the system for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured.

A broadcast receiver is implemented as a subclass (shown in program 5.4) of BroadcastReceiver and each broadcast is delivered as an Intent object.

```

public class MyReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        // Implement action for received broadcast.
    }
}

```

Program 5.4 Broadcast receiver subclass

Source(<https://developer.android.com/samples/AppShortcuts/src/com.example.android.appshortcuts/MyReceiver.html?hl=pt-br>)

## Content providers

As already stated in unit 2, content provider manages a shared set of app data that you can store in the file system, in a SQLite database, on the web, or on any other persistent storage location that your app can access. There are a few particular things this allows the system to do in managing an app:

- Assigning a Uniform Resource Identifier (URI) does not require that the app remain running, so URIs can persist after their owning apps have exited. The system only needs to make sure that an owning app is still running when it has to retrieve the app's data from the corresponding URI.



- These URIs also provide an important fine-grained security model. For example, an app can place the URI for an image it has on the clipboard, but leave its content provider locked up so that other apps cannot freely access it. When a second app attempts to access that URI on the clipboard, the system can allow that app to access the data via a temporary URI permission grant so that it is allowed to access the data only behind that URI, but nothing else in the second app.
- A unique aspect of the Android system design is that any app can start another app's component. For example, if you want the user to capture a photo with the device camera, there's probably another app that does that and your app can use it instead of developing an activity to capture a photo yourself. You don't need to incorporate or even link to the code from the camera app. Instead, you can simply start the activity in the camera app that captures a photo. When complete, the photo is even returned to your app so you can use it. To the user, it seems as if the camera is actually a part of your app.
- When the system starts a component, it starts the process for that app if it's not already running and instantiates the classes needed for the component. For example, if your app starts the activity in the camera app that captures a photo, that activity runs in the process that belongs to the camera app, not in your app's
- process. Therefore, unlike apps on most other systems, Android apps don't have a single entry point (there's no `main()` function).
- Because the system runs each app in a separate process with file permissions that restrict access to other apps, your app cannot directly activate a component from another app. However, the Android system can. To activate a component in another app, deliver a message to the system that specifies your *intent* to start a particular component. The system then activates the component for you.

### 5.2.1 Video -V6: Android Development



URL: <https://tinyurl.com/y9xuh62v>



In this video, we will be introducing the components of the Android Application Fundamentals. You may watch the video while reading this unit in order to understand the content better. After watching the video answer the question in Activity 5.1.

### 5.2.2 Check Your Progress



Identify the element that is not part of the basic application component of an Android application.

- ☐ Activities
- ☐ Services
- ☐ Content Providers
- ☐ ScreenCast Receivers
- ☐ Broadcast Receivers

## 5.3 ADDITIONAL COMPONENTS

Other than basic four app components there are few additional components as well. In this section we will discuss these additional components.

### Application Class

The Application class in Android is Base class for maintaining global application state which contains all other components such as activities and services. The Application class, or any subclass of the Application class, is instantiated before any other class when the process for your application/package is created.

### Defining Your Application Class

If we do want a custom application class, we start by creating a new class which extends `Android.app.Application` as the following Program 5.5:

```
import android.app.Application;

public class MyCustomApplication extends Application {
    // Called when the application is starting, before any // other
    // application objects have been created.
    // Overriding this method is totally optional!
    @Override
    public void onCreate() {
        super.onCreate();
    }
    // Required initialization logic here!
    // Called by the system when the device configuration changes
    // while your component is running.
    // Overriding this method is totally optional!
    @Override
    public void onConfigurationChanged(Configuration
    newConfig) {
        super.onConfigurationChanged(newConfig);
    }
    // This is called when the overall system is running // low on
    // memory, and would like actively running // processes to tighten
    // their belts.
    // Overriding this method is totally optional!
    @Override
    public void onLowMemory() {
        super.onLowMemory();
    }
}
```

Program 5.5 Application class

Source(<https://guides.codepath.com/android/Understanding-the-Android-Application-Class>)

And specify the android:name property in the <application> node in AndroidManifest.xml as in code snippet given below.

```
<application
    android:name=".MyCustomApplication"
    android:icon="@drawable/icon"
    android:label="@string/app_name"
    ...>
```

Source(<https://guides.codepath.com/android/Understanding-the-Android-Application-Class>)

That is all what you need to get started with your custom application.

## Fragment

A Fragment is a piece of an application's user interface or behavior that can be placed in an Activity. Interaction with fragments is done through `FragmentManager`, which can be obtained via `Activity.getFragmentManager()` and `Fragment.getFragmentManager()`.

The `Fragment` class can be used many ways to achieve a wide variety of results. In its core, it represents a particular operation or interface that is running within a larger Activity. A Fragment is closely tied to the Activity it is in, and cannot be used apart from one. Though `Fragment` defines its own lifecycle, that lifecycle is dependent on its activity: if the activity is stopped, no fragments inside of it can be started; when the activity is destroyed, all fragments will be destroyed.

All subclasses of `Fragment` must include a public no-argument constructor. The framework will often re-instantiate a fragment class when needed, in particular during state restore, and needs to be able to find this constructor to instantiate it. If the no-argument constructor is not available, a runtime exception will occur in some cases during state restore.

## Fragment Lifecycle

Though a `Fragment`'s lifecycle is tied to its owning activity, it has its own wrinkle on the standard activity lifecycle. It includes basic activity lifecycle methods such as `onResume()`, but also important are methods related to interactions with the activity and UI generation.

The core series of lifecycle methods that are called to bring a fragment up to resumed state (interacting with the user) are:

1. **`onAttach(Activity)`** called once the fragment is associated with its activity.
2. **`onCreate(Bundle)`** called to do initial creation of the fragment.
3. **`onCreateView(LayoutInflater, ViewGroup, Bundle)`** creates and returns the view hierarchy associated with the fragment.
4. **`onActivityCreated(Bundle)`** tells the fragment that its activity has completed its own `Activity.onCreate()`.
5. **`onViewStateRestored(Bundle)`** tells the fragment that all of the saved state of its view hierarchy has been restored.

6. **onStart()** makes the fragment visible to the user (based on its containing activity being started).
7. **onResume()** makes the fragment begin interacting with the user (based on its containing activity being resumed).

As a fragment is no longer being used, it goes through a reverse series of callbacks:

1. **onPause()** fragment is no longer interacting with the user either because its activity is being paused or a fragment operation is modifying it in the activity.
2. **onStop()** fragment is no longer visible to the user either because its activity is being stopped or a fragment operation is modifying it in the activity.
3. **onDestroyView()** allows the fragment to clean up resources associated with its View.
4. **onDestroy()** called to do final cleanup of the fragment's state.
5. **onDetach()** called immediately prior to the fragment no longer being associated with its activity.

## Fragment Layout

Fragments can be used as part of your application's layout, allowing you to better modularize your code and more easily adjust your user interface to the screen it is running on. As an example, we can look at a simple program consisting of a list of items, and display of the details of each item.

An activity's layout XML can include <fragment> tags to embed fragment instances inside of the layout. For example, here is a simple layout that embeds one fragment shown in code snippet below:

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/a
ndroid"
    android:layout_width="match_parent" android:layout_height="m
atch_parent">
    <fragment class="com.example.android.apis.app.FragmentLayout
$TitlesFragment"
        android:id="@+id/titles"
        android:layout_width="match_parent" android:layout_h
eight="match_parent"/>
</FrameLayout>
```

Source : (<https://developer.android.com/reference/android/app/Fragment.html>)

The layout is installed in the activity in the normal way as shown in code snippet below:

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

## View

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.

## Using Views

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.

**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)`.

*Note:* The Android framework is responsible for measuring, laying out and drawing views. You should not call methods that perform these actions on views yourself unless you are actually implementing a *ViewGroup*.

## Intents and Intent Filters

An Intent is a messaging object you can use to request an action from another app component. Although intents facilitate communication between components in several ways, there are three fundamental use-cases:

- **To start an activity:** An Activity represents a single screen in an app. You can start a new instance of an Activity by passing an Intent to `startActivity()`. The Intent describes the activity to start and carries any necessary data. If you want to receive a result from the activity when it finishes, call `startActivityForResult()`. Your activity receives the result as a separate Intent object in your activity's `onActivityResult()` callback. For more information, see the Activities guide.

- **To start a service:** A Service is a component that performs operations in the background without a user interface. You can start a service to perform a one-time operation (such as download a file) by passing an Intent to `startService()`. The Intent describes the service to start and carries any necessary data. If the service is designed with a client-server interface, you can bind to the service from another component by passing an Intent to `bindService()`. For more information, see the Services guide
- **To deliver a broadcast:** A broadcast is a message that any app can receive. The system delivers various broadcasts for system events, such as when the system boots up or the device starts charging. You can deliver a broadcast to other apps by passing an Intent to `sendBroadcast()`, `sendOrderedBroadcast()`, or `sendStickyBroadcast()`.

## Intents Types

There are two types of intents:

- **Explicit intents** specify the component to start by name (the fully-qualified class name). You'll typically use an explicit intent to start a component in your own app, because you know the class name of the activity or service you want to start. For example, start a new activity in response to a user action or start a service to download a file in the background.
- **Implicit intents** do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it. For example, if you want to show the user a location on a map, you can use an implicit intent to request that another capable app show a specified location on a map.

When you create an explicit intent to start an activity or service, the system immediately starts the app component specified in the Intent object.

When you create an implicit intent, the Android system finds the appropriate component to start by comparing the contents of the intent to the *intent filters* declared in the manifest file of other apps on the device. If the intent matches an intent filter, the system starts that component and delivers it the Intent object. If multiple intent filters are compatible, the system displays a dialog so the user can pick which app to use.

---

## 5.4 RESOURCES

---

This sits on top of the asset manager of the application (accessible through `getAssets()`) and provides a high-level API for getting typed data from the assets.

The Android resource system keeps track of all non-code assets associated with an application. You can use this class to access your application's resources. You can generally acquire the Resources instance associated with your application with `getResources()`.

The Android SDK tools compile your application's resources into the application binary at build time. To use a resource, you must install it correctly in the source tree

(inside your project's `res/` directory) and build your application. As part of the build process, the SDK tools generate symbols for each resource, which you can use in your application code to access the resources.

Using application resources makes it easy to update various characteristics of your application without modifying code, and—by providing sets of alternative resources—enables you to optimize your application for a variety of device configurations (such as for different languages and screen sizes). This is an important aspect of developing Android applications that are compatible on different types of devices.

You should always externalize resources such as images and strings from your application code, so that you can maintain them independently. Externalizing your resources also allows you to provide alternative resources that support specific device configurations such as different languages or screen sizes, which becomes increasingly important as more Android-powered devices become available with different configurations. In order to provide compatibility with different configurations, you must organize resources in your project's `res/` directory, using various sub-directories that group resources by type and configuration.

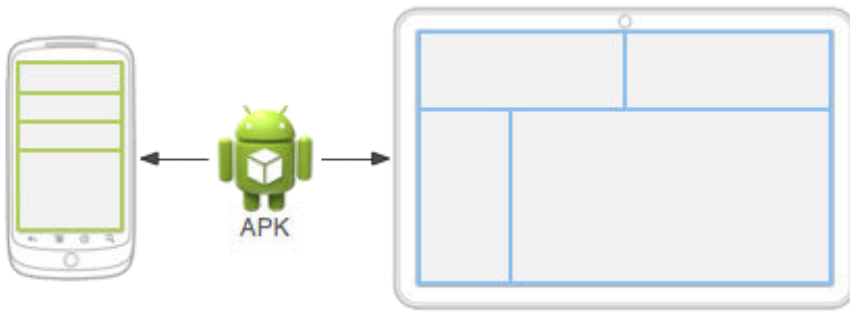
For any type of resource, you can specify *default* and multiple *alternative* resources for your application:

- Default resources are those that should be used regardless of the device configuration or when there are no alternative resources that match the current configuration.
- Alternative resources are those that you've designed for use with a specific configuration. To specify that a group of resources are for a specific configuration, append an appropriate configuration qualifier to the directory name.

For example, while your default UI layout is saved in the `res/layout/` directory, you might specify a different layout to be used when the screen is in landscape orientation, by saving it in the `res/layout-land/` directory. Android automatically applies the appropriate resources by matching the device's current configuration to your resource directory names.



**Figure 5.1** Two different devices, each using the default layout (the app provides no alternative layouts).



**Figure 5.2** Two different devices, each using a different layout provided for different screen sizes.

Figure 5.1: illustrates how the system applies the same layout for two different devices when there are no alternative resources available. Figure 2 shows the same application when it adds an alternative layout resource for larger screens.

The following section provide a complete guide to how you can organize your application resources, specify alternative resources, access them in your application, and more:

### Providing Resources

What kinds of resources you can provide in your app, where to save them, and how to create alternative resources for specific device configurations.

### Accessing Resources

How to use the resources you've provided, either by referencing them from your application code or from other XML resources.

### Handling Runtime Changes

How to manage configuration changes that occur while your Activity is running.

### Localization

A bottom-up guide to localizing your application using alternative resources. While this is just one specific use of alternative resources, it is very important in order to reach more users.

### Complex XML Resources

An XML format for building complex resources like animated vector drawables in a single XML file.

### Resource Types

A reference of various resource types you can provide, describing their XML elements, attributes, and syntax. For example, this reference shows you how to create a resource for application menus, drawables, animations, and more.

#### 5.4.1 Check Your Progress



Explain how you can organize your application resources.

---

## 5.5 ANDROID MANIFEST

---



Every application must have an `AndroidManifest.xml` file (with precisely that name) in its root directory. The manifest file provides essential information about your app to the Android system, which the system must have before it can run any of the app's code.

Among other things, the manifest file does the following:

- It names the Java package for the application. The package name serves as a unique identifier for the application.
- It describes the components of the application, which include the activities, services, broadcast receivers, and content providers that compose the application. It also names the classes that implement each of the components and publishes their capabilities, such as the Intent messages that they can handle. These declarations inform the Android system of the components and the conditions in which they can be launched.
- It determines the processes that host the application components.
- It declares the permissions that the application must have in order to access protected parts of the API and interact with other applications. It also declares the permissions that others are required to have in order to interact with the application's components.
- It lists the Instrumentation classes that provide profiling and other information as the application runs. These declarations are present in the manifest only while the application is being developed and are removed before the application is published.
- It declares the minimum level of the Android API that the application requires.
- It lists the libraries that the application must be linked against.

### Manifest file structure

The code snippet below shows the general structure of the manifest file and every element that it can contain. Each element, along with all of its attributes, is fully documented in a separate file.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest>
    <uses-permission/>
    <permission/>
    <permission-tree/>
    <permission-group/>
    <instrumentation/>
    <uses-sdk/>
    <uses-configuration/>
    <uses-feature/>
    <supports-screens/>
    <compatible-screens/>
    <supports-gl-texture/>
</manifest>
```

```

<application>

    <activity>
        <intent-filter>
            <action/>
            <category/>
            <data/>
        </intent-filter>
        <meta-data/>
    </activity>

    <activity-alias>
        <intent-filter> . . . </intent-filter>
        <meta-data/>
    </activity-alias>

<service>
    <intent-filter> . . . </intent-filter>
    <meta-data/>
</service>

<receiver>
    <intent-filter> . . . </intent-filter>
    <meta-data/>
</receiver>

<provider>
    <grant-uri-permission/>
    <meta-data/>
    <path-permission/>
</provider>
<uses-library/>
</application>
</manifest>

```

Source: (<https://developer.android.com/guide/topics/manifest/manifest-intro.html>)

## 5.6 File conventions

This section describes the conventions and rules that apply generally to all of the elements and attributes in the manifest file.

### Elements

Only the `<manifest>` and `<application>` elements are required. They each must be present and can occur only once. Most of the other elements can occur many times or not at all. However, at least some of them must be present before the manifest file becomes useful.

If an element contains anything at all, it contains other elements. All of the values are set through attributes, not as character data within an element.

Elements at the same level are generally not ordered. For example, the `<activity>`, `<provider>`, and `<service>` elements can be intermixed in any sequence. There are two key exceptions to this rule:

- An `<activity-alias>` element must follow the `<activity>` for which it is an alias.
- The `<application>` element must be the last element inside the `<manifest>` element. In other words, the `</application>` closing tag must appear immediately before the `</manifest>` closing tag.

## Attributes

In a formal sense, all attributes are optional. However, there are some attributes that must be specified so that an element can accomplish its purpose. Use the documentation as a guide. For truly optional attributes, it mentions a default value or states what happens in the absence of a specification.

Except for some attributes of the root `<manifest>` element, all attribute names begin with an `android:` prefix. For example, `android:alwaysRetainTaskState`. Because the prefix is universal, the documentation generally omits it when referring to attributes by name.

## Declaring class names

Many elements correspond to Java objects, including elements for the application itself (the `<application>` element) and its principal components: activities (`<activity>`), services (`<service>`), broadcast receivers (`<receiver>`), and content providers (`<provider>`).

If you define a subclass, as you almost always would for the component classes (Activity, Service, BroadcastReceiver, and ContentProvider), the subclass is declared through a `name` attribute. The name must include the full package designation. For example, a Service subclass might be declared as follows:

```
<manifest . . . >
  <application . . . >
    <service android:name="com.example.project.SecretService"
      . . . >
      . . .
    </service>
    . . .
  </application>
</manifest>
```

Source(<https://developer.android.com/guide/topics/manifest/manifest-intro.html>)

However, if the first character of the string is a period, the application's package name (as specified by the `<manifest>` element's package attribute) is appended to the string. The following assignment is the same as that shown above:

```
<manifest package="com.example.project" . . . >
  <application . . . >
    <service android:name=".SecretService" . . . >
      . . .
```

```

        </service>
        . . .
    </application>
</manifest>

```

Source(<https://developer.android.com/guide/topics/manifest/manifest-intro.html>)

When starting a component, the Android system creates an instance of the named subclass. If a subclass isn't specified, it creates an instance of the base class.

### Multiple values

If more than one value can be specified, the element is almost always repeated, rather than multiple values being listed within a single element. For example, an intent filter can list several actions:

```

<intent-filter . . . >
    <actionandroid:name="android.intent.action.EDIT"/>
    <actionandroid:name="android.intent.action.INSERT"/>
    <actionandroid:name="android.intent.action.DELETE"/>
    . . .
</intent-filter>

```

Source:(<https://developer.android.com/reference/android/content/Intent.html>)

### Resource values

Some attributes have values that can be displayed to users, such as a label and an icon for an activity. The values of these attributes should be localized and set from a resource or theme. Resource values are expressed in the following format:

**@[<i>package</i>:]<i>type</i>/<i>name</i>**

You can omit the *package* name if the resource is in the same package as the application. The *type* is a type of resource, such as *string* or *drawable*, and the *name* is the name that identifies the specific resource. Here is an example:

```

<activityandroid:icon="@drawable/smallPic" . . . >

```

Source (<https://developer.android.com/guide/topics/manifest/manifest-intro.html>)

The values from a theme are expressed similarly, but with an initial **?** instead of **@**:

**?[<i>package</i>:]<i>type</i>/<i>name</i>**

### String values

Where an attribute value is a string, you must use double backslashes (**\\**) to escape characters, such as **\\n** for a newline or **\\uxxxx** for a Unicode character.

### 5.6.1 Check Your Progress



Explain the role of AndroidManifest.xml file in an Android application.

---

## 5.7 File conventions

---



First part of this unit gave you an in-depth knowledge about the four main types of mobile application components. This unit also introduced you to the methods that were facilitated by Android and the elements used in any Android application development such as fragments and intents. You will need to refer to this unit when you are writing your application to understand the process beneath each method while executing.

---

## 5.8 FURTHER READINGS

---

- <https://developer.android.com/guide/components/fundamentals>
- <https://developer.android.com/guide/topics/manifest/manifest-intro>

---

## UNIT 6 ANDROID DEVELOPMENT

---

- 6.0 Introduction
- 6.1 Objectives
- 6.2 Creating Your First Program
  - 6.2.1 Check Your Progress
- 6.3 Building and running the application
  - 6.3.1 Check Your Progress
- 6.4 Summary
- 6.5 Further readings

---

### 6.0 INTRODUCTION

---

Now you are aware that Android applications can be developed using Android Studio. You have also learnt fundamentals of Android App design. How to build a simple user interface and handle user input will be described further in this unit.

Particularly, you will apply Java features in the context of core Android components (such as Activities and basic UI elements) by applying common tools (such as Android Studio) needed to develop Java programs and useful Android apps.

---

### 6.1 OBJECTIVES

---

After studying this unit, you should be able to :



#### Outcomes

- Develop and run an Android application.
- Run the developed application in both on the actual device and in the emulator.



#### Terminology

- AVD:** Android Virtual Emulator
- UI:** User Interface
- XML:** eXtensible Markup Language

---

### 6.2 CREATING YOUR FIRST PROGRAM

---

This unit shows you how to create a new Android project with Android Studio and describes some of the files in the project.

In Android Studio, create a new project

- In the New Project screen, enter the following values:

Application Name: "My First App"  
Company Domain: "example.com"

Android Studio fills in the package name and project location for you, but you can edit these if you'd like.

- Click Next.
- In the Target Android Devices screen, keep the default values and click Next.

The Minimum Required SDK is the earliest version of Android that your app supports, which is indicated by the API level. To support as many devices as possible, you should set this to the lowest version available that allows your app to provide its core feature set. If any feature of your app is possible only on newer versions of Android and it's not critical to the core feature set, enable that feature only when running on the versions that support it.

- In the Add an Activity to Mobile screen, select Empty Activity and click Next.
- In the Customize the Activity screen, keep the default values and click Finish.

After some processing, Android Studio opens and displays a "My First App" app with default files. You will add functionality to some of these files in the following lessons.

Now take a moment to review the most important files. First, be sure that the Project window is open (select View > Tool Windows > Project) and the Android view is selected from the drop-down list at the top. You can then see the following files:

**app > java > com.example.myfirstapp > MainActivity.java**

This file appears in Android Studio after the New Project wizard finishes. It contains the class definition for the activity you created earlier. When you build and run the app, the **Activity** starts and loads the layout file that says "Hello World!"

**app > res > layout > activity\_main.xml**

This XML file defines the layout of the activity. It contains a **TextView** element with the text "Hello world!"

**app > manifests > AndroidManifest.xml**

The manifest file describes the fundamental characteristics of the app and defines each of its components. You'll revisit this file as you follow these lessons and add more components to your app.

**Gradle Scripts > build.gradle**

Android Studio uses Gradle to compile and build your app. There is a build.gradle file for each module of your project, as well as a build.gradle file for the entire project. Usually, you're only interested in the build.gradle file for the module.


### 6.2.1 Check Your Progress



- Create a simple "Hello World Program"

## 6.3 BUILDING AND RUNNING THE APPLICATION

By default, Android Studio sets up new projects to deploy to the Emulator or a physical device with just a few clicks. With Instant Run, you can push changes to methods and existing app resources to a running app without building a new APK, so code changes are visible almost instantly.

To build and run your app, click Run . Android Studio builds your app with Gradle, asks you to select a deployment target (an emulator or a connected device), and then deploys your app to it. You can customize some of this default behaviour, such as selecting an automatic deployment target, by changing the run configuration.

If you want to use the Android Emulator to run your app, you need to have an Android Virtual Device (AVD) ready. If you haven't already created one, then after you click Run, click Create New Emulator in the Select Deployment Target dialog. Follow the Virtual Device Configuration wizard to define the type of device you want to emulate. For more information, see [Create and Manage Virtual Devices](#).

### Select and build a different module

If your project has multiple modules beyond the default app module, you can build a specific module as follows:

- Select the module in the Project panel, and then click Build > Make Module *module-name*.

Android Studio builds the module using Gradle. Once the module is built, you can run and debug it if you've built a module for a new app or new device, or use it as a dependency if you've built a library or Google Cloud module.

### To run a built app module:

- Click Run > Run and select the module from the Run dialog.


### Change the run/debug configuration

The run/debug configuration specifies the module to run, package to deploy, activity to start, target device, emulator settings, logcat options, and more. The default run/debug configuration launches the default project activity and uses the Select Deployment Target dialog for target device selection. If the default settings don't suit your project or module, you can customize the run/debug configuration, or even create a new one, at the project, default, and module levels. To edit a run/debug configuration, select **Run > Edit Configurations**.

### Change the build variant

By default, Android Studio builds the debug version of your app, which is intended only for testing, when you click Run. You need to build the release version to prepare your app for public release.

To change the build variant Android Studio uses, select **Build > Select Build**



**Variant** in the menu bar (or click Build Variants  in the windows bar), and then select a build variant from the drop-down menu. By default, new projects are set up with a debug and release build variant.



Using *product flavours*, you can create additional build variants for different versions of your app, each having different features or device requirements.

## Generate APKs


To create an APK for your app, follow these steps:

- Select the build variant you want to build from the **Build Variants**  window.
- Click **Build > Build APK** in the menu bar.
- To instead build the APK and immediately run it on a device, click **Run**  in the toolbar.

All built APKs are saved in *project-name/module-name/build/outputs/apk/*. You can also locate the generated APKs by clicking the link in the pop-up dialog that appears once the build is complete, as shown in figure 2

## Monitor the build process

You can view details about the build process by clicking **View > Tool Windows**

**>Gradle Console** (or by clicking **Gradle Console**  in the tool window bar). The console displays each task that Gradle executes in order to build your app, as shown in Figure 6.1.

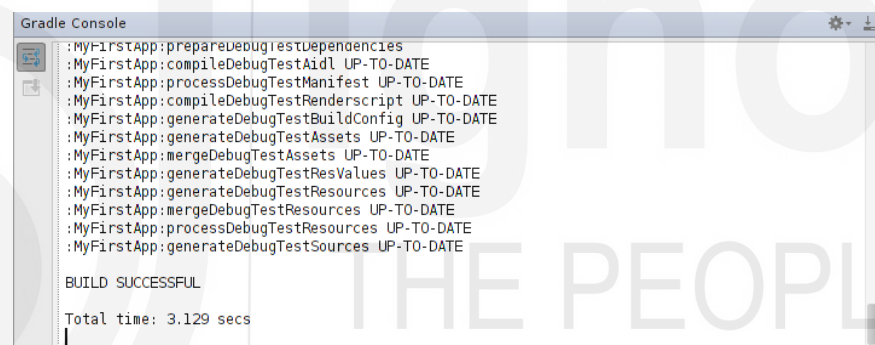


Figure 6.1 The Gradle Console in Android Studio.

If your build variants use product flavours, Gradle also invokes tasks to build those product flavours. To view the list of all available build tasks, click **View > Tool**

**Windows > Gradle** (or click **Gradle**  in the tool window bar).

If an error occurs during the build process, the *Messages* window appears to describe the issue. Gradle may recommend some command-line options to help you resolve the issue, such as `--stacktrace` or `--debug`. To use command-line options with your build process:

- Open the **Settings** or **Preferences** dialog:
- Navigate to **Build, Execution, Deployment > Compiler**.
- In the text field, next to *Command-line Options*, enter your command-line options.
- Click **OK** to save and exit.


Gradle will apply these command-line options the next time you try building your app.

## Running on the Emulator

Before you run your app on an emulator, you need to create an Android Virtual Device (AVD) definition. An AVD definition defines the characteristics of an

Android phone, tablet, Android Wear, or Android TV device that you want to simulate in the Android Emulator.

Create an AVD Definition as follows:

- Launch the Android Virtual Device Manager by selecting **Tools > Android > AVD Manager**, or by clicking the AVD Manager icon  in the toolbar.
- In the Your Virtual Devices screen, click Create Virtual Device.
- In the Select Hardware screen, select a phone device, such as Nexus 6, and then click Next.
- In the System Image screen, choose the desired system image for the AVD and click Next. (if you don't have a particular system image installed, you can get it by clicking the download link.)

Verify the configuration settings (for your first AVD, leave all the settings as they are), and then click **Finish**.

## Create and Manage Virtual Devices

An Android Virtual Device (AVD) definition lets you define the characteristics of an Android phone, tablet, Android Wear, or Android TV device that you want to simulate in the Android Emulator. The AVD Manager helps you easily create and manage AVDs

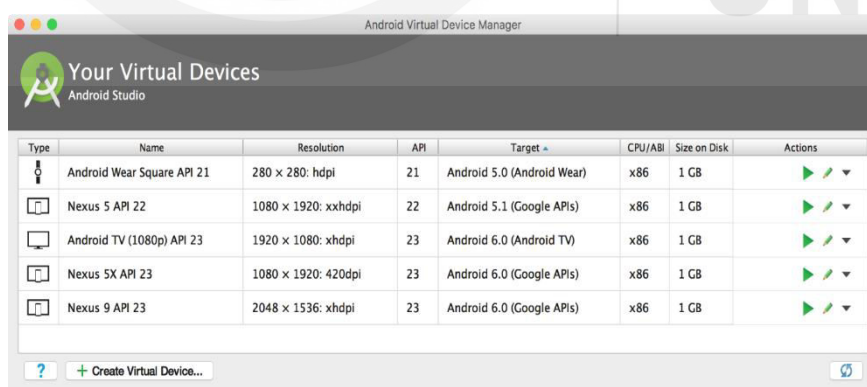
### VIEWING AND MANAGING YOUR AVDS

The AVD Manager lets you manage your AVDs all in one place.

To run the AVD Manager, do one of the following:

- In Android Studio, select **Tools > Android > AVD Manager**.
- Click **AVD Manager**  in the toolbar.

The AVD Manager appears as shown in Figure 6.2.



**Figure 6.2 AVD manager**

It displays any AVDs you have already defined. When you first install Android Studio, it creates one AVD. If you defined AVDs for Android Emulator 24.0.x or lower, you need to recreate them.

From this page, you can:

- Define a new [AVD](#) or [hardware profile](#).
- Edit an existing [AVD](#) or [hardware profile](#).
- Delete an [AVD](#) or [hardware profile](#).
- [Import or export](#) hardware profile definitions.
- [Run](#) an AVD to start the emulator.
- [Stop](#) an emulator.
- [Clear](#) data and start fresh, from the same state as when you first ran the emulator.
- [Show](#) the associated AVD .ini and .img files on disk.
- [View](#) AVD configuration details that you can include in any bug reports to the Android Studio team.

## Creating an AVD

You can create a new AVD from the beginning, or duplicate an AVD and change some properties.

To create a new AVD:

- From the Your Virtual Devices page of the AVD Manager, click Create Virtual Device.
- Alternatively, run your app from within Android Studio. In the Select Deployment Target dialog, click Create New Emulator.

Then Select Hardware page appears as shown in Figure 6.3

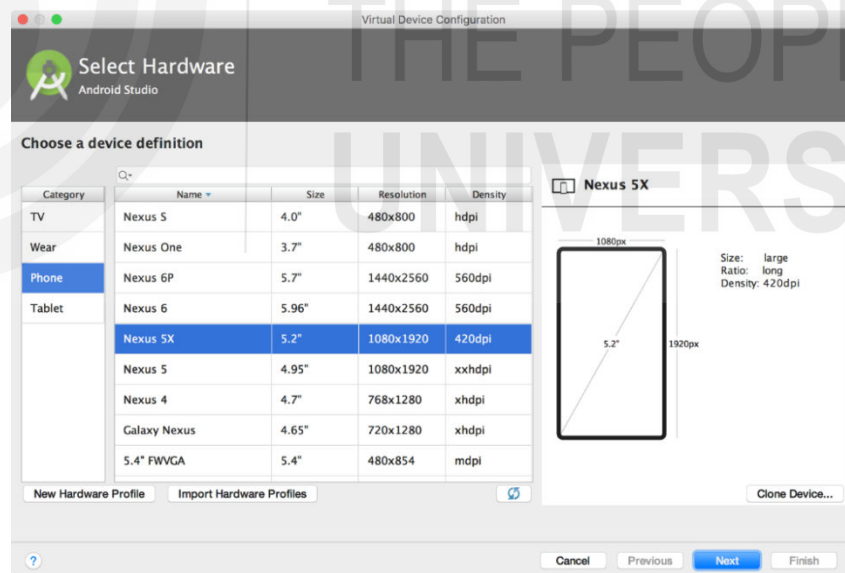
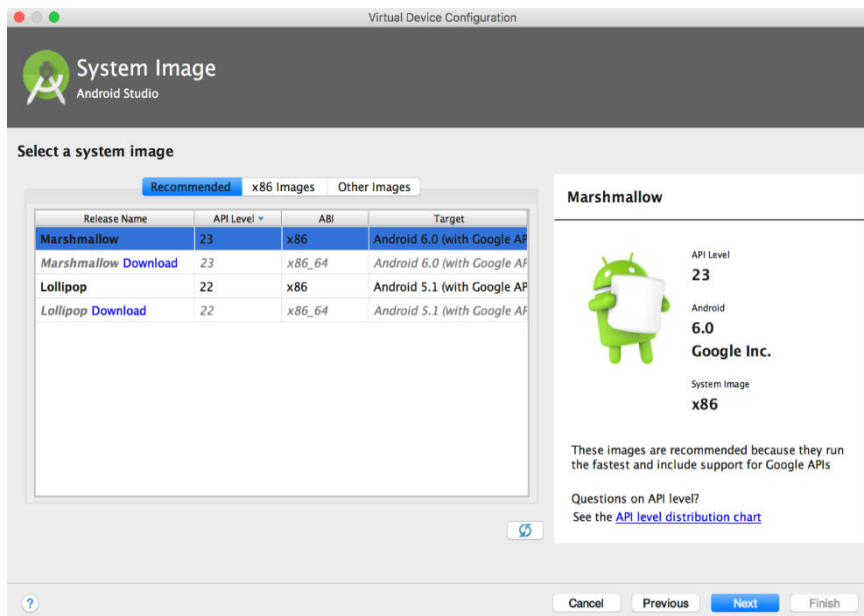


Figure 6.3: Virtual Device Configuration(Select Hardware)

- Select a hardware profile, and then click 'Next'.

If you don't see the hardware profile you want, you can create or import a hardware profile. Then System Image page appears as shown in Figure 6.4.



**Figure 6.4: Virtual Device Configuration(System Image)**

- Select the system image for a particular API level, and then click 'Next'.

The Recommended tab lists recommended system images. The other tabs include a more complete list. The right pane describes the selected system image. x86 images run the fastest in the emulator.

If you see Download next to the system image, you need to click it to download the system image. You must be connected to the internet to download it.

The API level of the target device is important, because your app won't be able to run on a system image with an API level that's less than that required by your app, as specified in the minSdkVersion attribute of the app manifest file. For more information about the relationship between system API level and minSdkVersion, see Versioning Your Apps.

If your app declares a <uses-library> element in the manifest file, the app requires a system image in which that external library is present. If you want to run your app on an emulator, create an AVD that includes the required library. To do so, you might need to use an add-on component for the AVD platform; for example, the Google APIs add-on contains the Google Maps library.

The Verify Configuration page appears as shown in Figure 6.5.

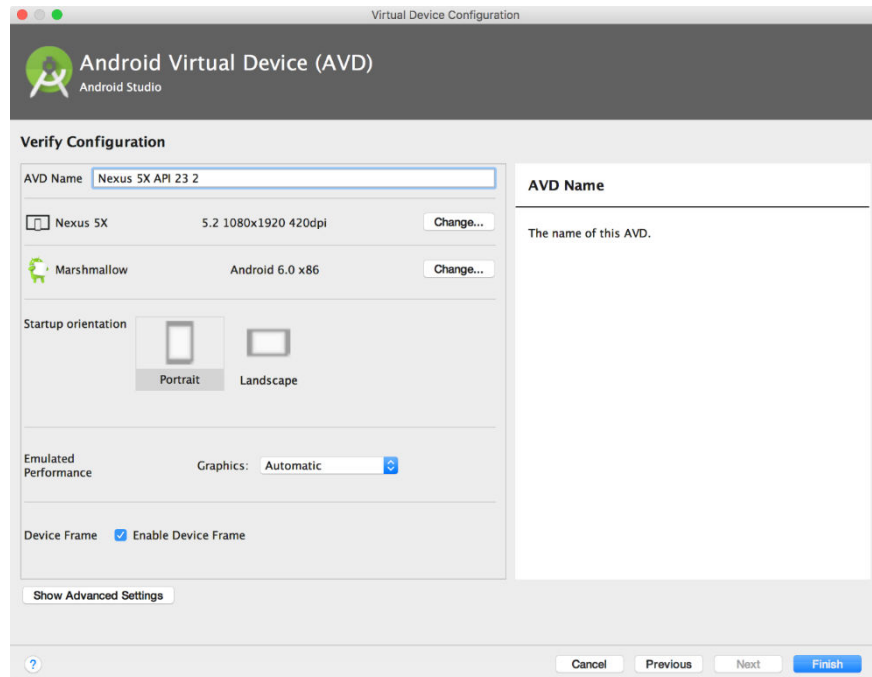


Figure 6.5: Virtual Device Configuration(AVD)

- Change AVD properties as needed, and then click Finish.
- Click Show Advanced Settings to show more settings, such as the skin.

The new AVD appears in the Your Virtual Devices page or the Select Deployment Target dialog.

### To create an AVD starting with a copy:

- From the Your Virtual Devices page of the AVD Manager, right-click an AVD and select Duplicate.

Or click Menu ▼ and select Duplicate.

The Verify Configuration page appears (See previous image).

Click Change or Previous if you need to make changes on the System Image and Select Hardware pages.

Make your changes, and then click Finish.

The AVD appears in the Your Virtual Devices page.

### Running and Stopping an Emulator, and Clearing Data

From the *Your Virtual Devices* page, you can perform the following operations on an emulator:

- To run an emulator that uses an AVD, double-click the AVD. Or click Launch ►.
- To stop a running emulator, right-click an AVD and select **Stop**. Or click Menu ▼ and select **Stop**.
- To clear the data for an emulator, and return it to the same state as when it was first defined, right-click an AVD and select **Wipe Data**. Or click Menu ▼ and select **Wipe Data**.

## Running on the actual device

When building an Android app, it's important that you always test your application on a real device before releasing it to users. This page describes how to set up your development environment and Android-powered device for testing and debugging on the device.

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. If you don't yet have a device, check with the service providers in your area to determine which Android-powered devices are available.

When building an Android app, it's important that you always test your application on a real device before releasing it to users. This page describes how to set up your development environment and Android-powered device for testing and debugging on the device.

**Note:** When developing on a device, keep in mind that you should still use the Android emulator to test your application on configurations that are not equivalent to those of your real device. Although the emulator does not allow you to test every device feature, it does allow you to verify that your application functions properly on different versions of the Android platform, in different screen sizes and orientations, and more.

### Enable Developer mode


- Android-powered devices have a host of developer options that you can access on the phone, which let you
- Enable debugging over USB.
- Quickly capture bug reports onto the device.
- Show CPU usage on screen.
- Draw debugging information on screen such as layout bounds, updates on GPU views and hardware layers, and other information.
- Plus many more options to simulate app stresses or enable debugging options.



Figure 6.6: Enabling Developer Mode in Mobile

To access these settings, open the Developer options in the system Settings as shown in Figure 6.6. On Android 4.2 and higher, the Developer options screen is hidden by default. To make it visible, go to Settings > About phone and tap Build number seven times. Return to the previous screen to find Developer options at the bottom.

### Running app on the actual device

- In Android Studio, select your project and click **Run**  from the toolbar.
- In the **Select Deployment Target** window, select your device, and click **OK**.

Android Studio installs the app on your connected device and starts it.

### 6.3.1 Check Your Progress



- Build a simple app and run the application on
  1. Emulator
  2. Actual Device

---

## 6.4 SUMMARY

---



In this unit, you learnt how to develop maintainable mobile apps that include the core Android components discussed in the previous unit. You need to watch the provided video on how to create an app from scratch using Android Studio. We also discussed about creating the Android Virtual Devices to run, testing and debugging the application.

Configuring and saving the launch configuration and associating an AVD with your project as discussed here will make your debugging and testing task easier. You also learnt about different project files that are created during the development process of an Android application as well as different development tools, Android application components and adding permissions to an application.

---

## 6.4 FURTHER READINGS

---

- <https://developer.android.com/training/basics/firstapp?authuser=1>
- <https://developer.android.com/studio/run/managing-avds>



---

## UNIT 7 DEVICE COMPATIBILITY

---

### 7.0 Introduction

#### 7.1 Objectives

#### 7.2 Application availability to devices

##### 7.2.1 Check Your Progress

#### 7.3 Device Features

#### 7.4 Platform Version

##### 7.4.1 Check Your Progress

#### 7.5 Screen Configuration

##### 7.5.1 Video- V7: Device Compatibility

##### 7.5.2 Check Your Progress

#### 7.6 Summary

#### 7.7 Further readings

---

## 7.0 INTRODUCTION

---

Android is designed to run on many different types of devices, from phones to tablets and televisions. The range of devices provides a huge potential audience for the Android applications. In order to be successful on all these devices, it provides a flexible user interface that adapts to different screen configurations.

Android provides a dynamic app framework that can provide configuration-specific app resources in static files such as different XML layouts for different screen sizes. Android loads the appropriate resources based on the current device configuration. With some additional app resources, developer can publish a single application package (APK) that provides an optimized user experience on a variety of devices.

This unit will be focused on device compatibility. There are two types of compatibility *device compatibility* and *app compatibility*.

Hardware manufacturer can build a device that runs the Android operating system. Yet, a device is "**Android compatible**" only if it can correctly run apps written for the *Android execution environment* and each device must pass the Compatibility Test Suite (CTS) in order to be considered compatible.

Though Android runs on a wide range of device configurations, some features are not available on all devices. For example, some devices may not include a compass sensor. If your app's core functionality requires the use of a compass sensor, then your app is compatible only with devices that include a compass sensor.

---

## 7.1 OBJECTIVES

---

After studying this unit, you should be able to:



### Outcomes

- identify compatibility of an application with different devices.
- discuss screen configuration for various device sizes and resolutions
- evaluate device compatibility of an application



**Terminology****Widget:**

an application, or a component of an interface, that enables a user to perform a function or access a service

**platform:**

where any piece of software is executed

**screen density:**

quantity of pixels within a physical area of the screen

**resolution:**

The total number of physical pixels on a screen

---

## 7.2 APPLICATION AVAILABILITY TO DEVICES

---

Android supports a variety of features your app can control through platform APIs. Some features are hardware-based such as a compass sensor, some are software-based such as app widgets, and some features dependent on the platform version. You have to control application availability to the devices based on the features of your application because not every device supports every feature.

To achieve the largest user-base possible for an app, developer should strive to support as many device configurations as possible using a single APK. In most situations, it can do by disabling optional features at runtime and providing app resources with alternatives for different configurations.

Device characteristics are Device features, Platform version and Screen configuration.

### 7.2.1 Check Your Progress



State the importance of device configuration to maintain device compatibility

---

## 7.3 DEVICES FEATURES

---

In order to manage your app's availability based on device features, Android defines *feature IDs* for any hardware or software feature that may not be available on all devices.

For instance, you can prevent users from installing your app when their devices do not provide a given feature by declaring it with a `<uses-feature>` element in your app's manifest file.

*Example: Declare the compass sensor*

If your app does not make sense on a device that lacks a compass sensor, you can declare the compass sensor as required with the following manifest tag as shown in the code snippet below.

```
<manifest ... >
    <uses-feature android:name="android.hardware.sensor.compass"
                  android:required="true"/>
    ...
</manifest>
```

Google Play Store compares the features that your app requires to the features available on each user's device to determine whether your app is compatible with that device. If the device does not provide all the features your app requires, the user cannot install your app.

However, if your app's primary functionality does not require a device feature, you should set the required attribute to `"false"` and check for the device feature at runtime. If the app feature is not available on the current device, gracefully degrade the corresponding app feature. For example, you can query whether a feature is available by calling `hasSystemFeature()` like this:

```
PackageManager pm = getPackageManager();
if (!pm.hasSystemFeature(PackageManager.FEATURE_SENSOR_COMPASS))
{
    // This device does not have a compass, turn off the //compass
    feature
    disableCompassFeature();
}
```

## 7.4 PLATFORM VERSION

Different devices may run different versions of the Android platform, such as Android 4.0 or Android 6.0. Each successive platform version often adds new APIs not available in the previous version. To indicate which set of APIs are available, each platform version specifies an API level.

For instance, Android 1.0 is API level 1 and Android 6.0 is API level 23.

The API level allows you to declare the minimum version with which your app is compatible, using the `<uses-sdk>` manifest tag and its `minSdkVersion` attribute.

*For example:*

The Calendar Provider APIs were added in Android 4.0 (API level 14). If your app cannot function without these APIs, you should declare API level 14 as your app's minimum supported version like this:

```
<manifest ... >
    <uses-
sdkandroid:minSdkVersion="14"android:targetSdkVersion="19"/>
    ...
</manifest>
```

The `minSdkVersion` attribute declares the minimum version with which your app is compatible and the `targetSdkVersion` attribute declares the highest version on which you have optimized your app. Each successive version of Android provides compatibility for apps that were built using the APIs from previous platform versions, so your app should always be compatible with future versions of Android while using the documented Android APIs.

However, if your app uses APIs added in a more recent platform version, but does not require them for its primary functionality, you should check the API level at runtime and gracefully degrade the corresponding features when the API level is too low.

In this case, set the [minSdkVersion](#) to the lowest value possible for your app's primary functionality, then compare the current system's version, `SDK_INT`, to one the codename constants in

`Build.VERSION_CODES` that corresponds to the API level you want to check.

*For example:*

```
if (Build.VERSION.SDK_INT < Build.VERSION_CODES.HONEYCOMB) {
    // Running on something older than API level 11, so disable
    // the drag/drop features that use ClipboardManager APIs
    disableDragAndDrop();
}
```

### 7.4.1 Check Your Progress



Discuss different devices and different versions of the Android platform

Now we will see how Android runs on devices of various sizes.

## 7.5 SCREEN CONFIGURATION

Android runs on devices of various sizes, from phones to tablets and TVs. In order to categorize devices by their screen type, Android defines characteristics for each device:

- **Screen size** - The physical size of the screen. Actual physical size, measured as the screen's diagonal.
- For simplicity, Android groups all actual screen sizes into four generalized sizes: small, normal, large, and extra-large.
- **Screen density** -. The quantity of pixels within a physical area of the screen; usually referred to as dpi (dots per inch). For example, a "low" density screen has fewer pixels within a given physical area, compared to a "normal" or "high" density screen.
- For simplicity, Android groups all actual screen densities into six generalized densities: low, medium, high, extra-high, extra-extra-high, and extra-extra-extra-high.
- **Resolution**- The total number of physical pixels on a screen. When adding support for multiple screens, applications do not work directly with resolution; applications should be concerned only with screen size and density, as specified by the generalized size and density groups.

- ldpi (low) ~120dpi
- mdpi (medium) ~160dpi
- hdpi (high) ~240dpi
- xhdpi (extra-high) ~320dpi
- xxhdpi (extra-extra-high) ~480dpi

## Density-independent pixel (dp)

A virtual pixel unit that you should use when defining UI layout, to express layout dimensions or position in a density-independent way.

The density-independent pixel is equivalent to one physical pixel on a 160 dpi screen, which is the baseline density assumed by the system for a "medium" density screen.

At runtime, the system transparently handles any scaling of the dp units, as necessary, based on the actual density of the screen in use. The conversion of dp units to screen pixels is simple:  $px = dp * (dpi / 160)$ .

For example, on a 240 dpi screen, 1 dp equals 1.5 physical pixels. You should always use dp units when defining your application's UI, to ensure proper display of your UI on screens with different densities.

By default, your app is compatible with all screen sizes and densities, because the system makes the appropriate adjustments to your UI layout and image resources as necessary for each screen. However, you should optimize the user experience for each screen configuration by adding specialized layouts for different screen sizes and optimized bitmap images for common screen densities.

## Use wrap\_content and match\_parent

To ensure that your layout is flexible and adapts to different screen sizes, you should use "wrap\_content" and "match\_parent" for the width and height of some view components. If you use "wrap\_content", the width or height of the view is set to the minimum size necessary to fit the content within that view, while "match\_parent" makes the component expand to match the size of its parent view. By using the "wrap\_content" and "match\_parent" size values instead of hard-coded sizes, your views either use only the space required for that view or expand to fill the available space, respectively.

*For example:*

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <LinearLayout android:layout_width="match_parent"
        android:id="@+id/linearLayout1"
        android:gravity="center"
        android:layout_height="50dp">
        <ImageView android:id="@+id/imageView1"
            android:layout_height="wrap_content"
            android:layout_width="wrap_content"
            android:src="@drawable/logo"
```

```

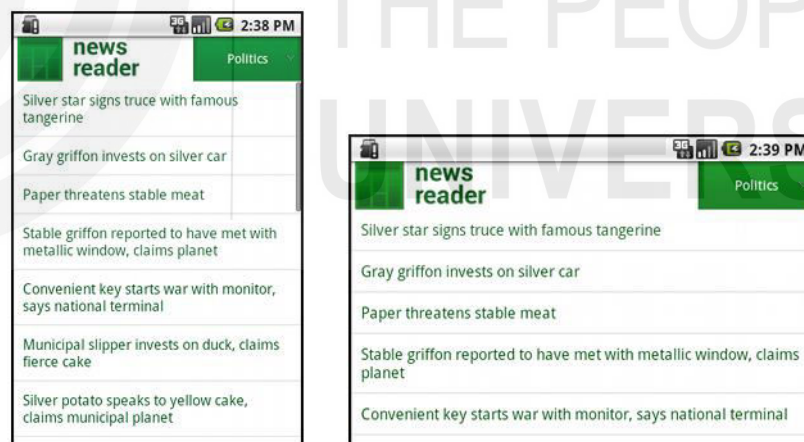
        android:paddingRight="30dp"
        android:layout_gravity="left"
        android:layout_weight="0"/>
        <View android:layout_height="wrap_content"
            android:id="@+id/view1"
            android:layout_width="wrap_content"
            android:layout_weight="1"/>
        <Button android:id="@+id/categorybutton"
            android:background="@drawable/button_bg"
            android:layout_height="match_parent"
            android:layout_weight="0"
            android:layout_width="120dp"
            style="@style/CategoryButtonStyle"/>
    </LinearLayout>

    <fragment android:id="@+id/headlines"
        android:layout_height="fill_parent"
        android:name="com.example.android.newsreader.Head
linesFragment"
        android:layout_width="match_parent"/>
</LinearLayout>

```

Notice how the sample uses "wrap\_content" and "match\_parent" for component sizes rather than specific dimensions. This allows the layout to adapt correctly to different screen sizes and orientations.

For example, figure 7.1 shows what this layout looks like in portrait and landscape mode. Notice that the sizes of the components adapt automatically to the width and height:



**Figure 7.1:** News Reader sample app in portrait (left) and landscape (right)  
(Source: <https://developer.android.com/training/multiscreen/screensizes.html>)

### 7.5.1 Video- V7: Device Compatibility



Let us watch the video on device compatibility and do the activity 7.3.



URL: <https://tinyurl.com/ydc19trg>



Calculate resolution values for your mobile device using density independent pixels.

---

## 7.6 SUMMARY

---



In this unit, we discussed device compatibility and application availability to devices based on the device characteristics. These characteristics include device features, Platform version and Screen configuration. Furthermore, Android defines characteristics for each device such as screen size, density and resolution.

---

## 7.7 FURTHER READINGS

---

<https://developer.android.com/guide/app-compatibility>

[https://developer.android.com/guide/practices/screens\\_support](https://developer.android.com/guide/practices/screens_support)

ignou  
THE PEOPLE'S  
UNIVERSITY



---

## UNIT 8 USER INTERFACE DESIGN

---

- 8.1 Introduction
- 8.2 Objectives
- 8.3 UI Overview
- 8.4 User Interface Layout
  - 8.4.1 Video – V8: Creating GUI for Android Application
- 8.5 Input Controls
- 8.6 Fundamentals of designing user interfaces using XML
  - 8.6.1 Check Your Progress
  - 8.6.2 Check Your Progress
  - 8.6.3 Check Your Progress
  - 8.6.4 Check Your Progress
- 8.7 Design a UI with Layout Editor
- 8.8 Managing Touch Events in a ViewGroup
- 8.9 Best Practices for User Interface
- 8.10 Unit summary
- 8.11 Answers to check your progress
- 8.12 Further readings

---

### 8.0 INTRODUCTION

---

This unit will focus on theory of User Interface (UI) Design and at the end of this unit you will be able to build a user interface using Android layouts for different types of devices. Hence, this unit helps you to create an application that is smooth and responsive by using best practices for graphical user interface (GUI) design. .

---

### 8.1 OBJECTIVES

---

Upon completion of this unit you should be able to:



#### Outcomes

- design Graphical User Interface(GUI) using Extended Markup Language(XML)
- use best practices for GUI design
- apply layouts to improve application performance



#### Terminology

- attributes:** a piece of information which describes the properties of a field
- layout:** arrangement or the plan
- adapter:** converts one type of software to another type
- view:** object that user can interact.



---

## 8.2 UI OVERVIEW

---

All user interface elements in an Android app are built using *View* and *ViewGroup* objects.

Android provides a collection of both *View* and *ViewGroup* subclasses that offer you common input controls (such as buttons and text fields) and various layout models (such as a linear or relative layout).

- **View** - is an object that draws something on the screen that the user can interact. UI widgets such as buttons or text fields.
- **ViewGroup** - is an object that holds other *View* objects in order to define the layout of the interface. Invisible view containers that define how the child views are laid out, such as in a grid or a vertical list.

---

## 8.3 USER INTERFACE LAYOUT

---

The user interface for each component of your app is defined using a hierarchy of *View* and *ViewGroup* objects, as shown in figure 8.1.

Each view group is an invisible container that organizes child views, while the child views may be input controls or other widgets that draw some part of the UI. This hierarchy tree can be as simple or complex as you need it to be (but simplicity is best for performance).

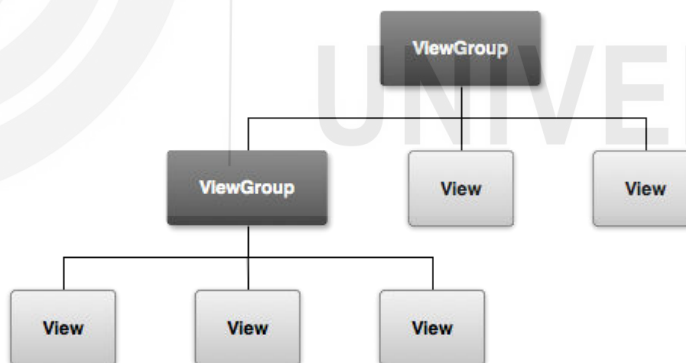


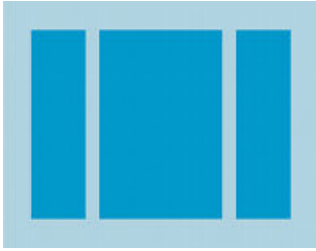
Figure 8.1. *ViewGroup* objects form branches in the layout with *View* objects

(Source: <https://developer.android.com/guide/topics/ui/overview.html>)

### Common Layouts

Each subclass of the *ViewGroup* class provides a unique way to display the views you nest within it. Some of the common layout types that are built into the Android platform are given below in figure 8.2. You can nest one or more layouts within another layout to achieve your UI design.

- Linear Layout



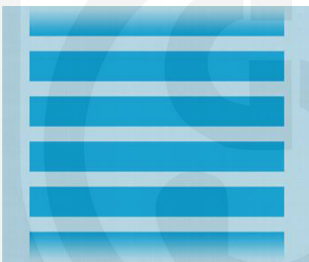
LinearLayout is a view group that aligns all its children in a single direction, vertically or horizontally. You can specify the layout direction with the `android:orientation` attribute. A layout that organizes its children into a single horizontal or vertical row. It creates a scrollbar if the length of the window exceeds the length of the screen.

- Relative Layout



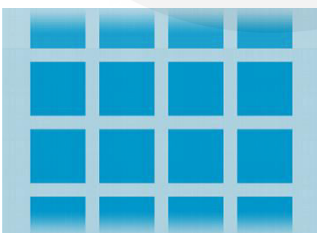
RelativeLayout is a viewgroup that displays child views in relative positions. The position of each view can be specified as relative to sibling elements (such as to the left-of or below another view) or in positions relative to the parent RelativeLayout area (such as aligned to the bottom, left or center).

- List View



ListView is a viewgroup that displays a list of scrollable items. The list items are automatically inserted to the list using an Adapter that pulls content from a source such as an array or database query and converts each item result into a view that is placed into the list.

- Grid View



GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid. The grid items are automatically inserted to the layout using a ListAdapter.

**Figure 8.2: common layouts**

(Source:<https://developer.android.com/guide/topics/ui/layout/linear.html>)

### 8.3.1 Video – V8: Creating GUI for Android Application



You may watch this screencast to see how to build a simple user interface while studying this unit.



URL: <https://tinyurl.com/y9hkqpjb>

### Building Layouts with an Adapter

When the content for your layout is dynamic or not pre-determined, you can use a layout that subclasses `AdapterView` to populate the layout with views at runtime. A subclass of the `AdapterView` class uses an `Adapter` to bind data to its layout. The `Adapter` behaves as an intermediary between the data source and the `AdapterView` layout the `Adapter` retrieves the data (from a source such as an array or a database query) and converts each entry into a view that can be added into the `AdapterView` layout.

Common layouts backed by an adapter include:

- List View - Displays a scrolling single column list
- Grid View - Displays a scrolling grid of columns and rows.

A layout defines the visual structure for a user interface, such as the UI for an activity or app widget. You can declare a layout in two ways:

- Declare UI elements in XML - Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.
- Instantiate layout elements at runtime - Your application can create View and ViewGroup objects programmatically.

Now we will see how to design the UI using XML.

---

## 8.4 INPUT CONTROLS

---

Input controls are the interactive components in your app's user interface. Android provides a wide variety of controls you can use in your UI, such as buttons, text fields, seek bars, checkboxes, zoom buttons, and toggle buttons. Adding an input control to your UI is as simple as adding an XML element to your XML layout.

*Example: layout with a text field and button*

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal">
    <EditText android:id="@+id/edit_message"
        android:layout_weight="1"
        android:layout_width="0dp"
```

```

android:layout_height="wrap_content"
android:hint="@string/edit_message"/>
<Button android:id="@+id/button_send"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/button_send"
android:onClick="sendMessage"/>
</LinearLayout>

```

### Common controls:

Table 8.1 gives a list of some common controls that you can use in your app.

**Table 8.1: common controls**

Control Type	Description	Related Classes
Button	A push-button that can be pressed, or clicked, by the user to perform an action.	Button
Text field	An editable text field. You can use the <code>AutoCompleteTextView</code> widget to create a text entry widget that provides auto-complete suggestions	<code>EditText</code> , <code>AutoCompleteTextView</code>
Checkbox	An on/off switch that can be toggled by the user. You should use checkboxes when presenting users with a group of selectable options that are not mutually exclusive.	<code>CheckBox</code>
Radio button	Similar to checkboxes, except that only one option can be selected in the group.	<code>RadioGroup</code> , <code>RadioButton</code>
Toggle button	An on/off button with a light indicator.	<code>ToggleButton</code>
Spinner	A drop-down list that allows users to select one value from a set.	<code>Spinner</code>

## 8.5 FUNDAMENTALS OF DESIGNING USER INTERFACES USING XML

To declare your layout, you can instantiate `View` objects in code and start building a tree, but the easiest and most effective way to define layout is with an XML file. XML offers a human-readable structure for the layout, similar to HTML.

### Write the XML

Using Android's XML vocabulary, you can quickly design UI layouts and the screen elements they contain, in the same way you create web pages in HTML with a series of nested elements.

Each layout file must contain exactly one root element, which must be a `View` or `ViewGroup` object. Once you have defined the root element, you can add additional

layout objects or widgets as child elements to build a View hierarchy that defines your layout.

*Example: XML layout that uses a vertical LinearLayout to hold a TextView and a Button*

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">
    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="I am a TextView" />
    <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="I am a Button" />
</LinearLayout>
```

After you have declared your layout in XML, save the file with the .xml extension, in your Android project's res/layout/ directory, so it will properly compile.

### Load the XML Resource

When you compile your application, each XML layout file is compiled into a View resource. You should load the layout resource from your application code, in your Activity.onCreate() callback implementation. Do so by calling setContentView(), passing it the reference to your layout resource in the form of: R.layout.layout\_file\_name.

*Example: XML layout saved as main\_layout.xml*

*You need to load it for your Activity.*

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_layout);
}
```

The onCreate() callback method in your Activity is called by the Android framework when your Activity is launched.

### Attributes

Every View and ViewGroup object supports their own variety of XML attributes. Some attributes are specific to a View object (for example, TextView supports the textSize attribute), but these attributes are also inherited by any View objects that may extend this class. Few of these attributes are given below.

- ID attribute

Any View object may have an integer ID associated with it, to uniquely identify the View within the tree. When the application is compiled, this ID is referenced as an

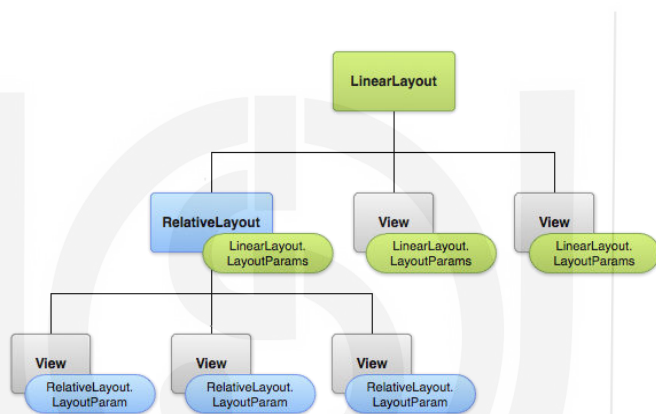
integer, but the ID is typically assigned in the layout XML file as a string, in the `id` attribute. This is an XML attribute common to all View objects (defined by the View class) and you will use it very often.

The syntax for an ID, inside an XML tag is:

```
android:id="@+id/my_button"
```

- **Layout Parameters**  
XML layout attributes named `layout_something` define layout parameters for the View that are appropriate for the ViewGroup in which it resides. Every ViewGroup class implements a nested class that extends `ViewGroup.LayoutParams`. This subclass contains property types that define the size and position for each child view, as appropriate for the view group.

Figure 8.3 shows the hierarchy of layouts associated with each view.



**Figure 8.3. Hierarchy with layout parameters associated with each view.**

(Source: <https://developer.android.com/guide/topics/ui/declaring-layout.html#CommonLayouts>)

- **Layout Position**  
The geometry of a view is that of a rectangle. A view has a location, expressed as a pair of *left* and *top* coordinates, and two dimensions, expressed as a width and a height. The unit for location and dimensions is the pixel. It is possible to retrieve the location of a view by invoking the methods `getLeft()` and `getTop()`. In addition, several convenience methods are offered to avoid unnecessary computations, namely `getRight()` and `getBottom()`.
- **Size, Padding and Margins**  
The size of a view is expressed with a width and a height. A view possesses two pairs of width and height values. The first pair is known as *measured width* and *measured height*. These dimensions define how big a view wants to be within its parent. The measured dimensions can be obtained by calling `getMeasuredWidth()` and `getMeasuredHeight()`.

The next section will focus on common layout for Android application.

### 8.5.1 Check Your Progress



Create a Linear Layout following the given steps

**Step 1:** In Android Studio, from the res/layout directory, open the content\_main.xml file.

The Blank Activity template you chose when you created this project includes the content\_my.xml file with a RelativeLayout root view and a TextView child view.

**Step 2:** In the Preview pane, click the Hide icon  to close the Preview pane.

In Android Studio, when you open a layout file, you are first shown the Preview pane. Clicking elements in this pane opens the WYSIWYG tools in the Design pane. For this lesson, you are going to work directly with the XML.

**Step 3:** Delete the <TextView> element.

**Step 4:** Change the <RelativeLayout> element to <LinearLayout>.

**Step 5:** Add the android:orientation attribute and set it to "horizontal".

**Step 6:** Remove the android:padding attributes and the tools:context attribute.

The result looks like this:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:showIn="@layout/activity_main">
```

As with every [View](#) object, you must define certain XML attributes to specify the EditText object's properties.

### 8.5.2 Check Your Progress



Add a Text Field to created layout following the given steps.

**Step 1:** In the content\_my.xml file, within the <LinearLayout> element, define an <EditText> element with the id attribute set to @+id/edit\_message.

**Step 2:** Define the layout\_width and layout\_height attributes as wrap\_content.

**Step 3:** Define a hint attribute as a string object named edit\_message.

*The <EditText> element should read as follows:*



```
<EditText android:id="@+id/edit_message"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:hint="@string/edit_message" />
```

### 8.5.3 Check Your Progress



Add a Button to created layout following the given steps.

**Step 1:** In Android Studio, from the res/layout directory, edit the content\_my.xml file.

**Step 2:** Within the <LinearLayout> element, define a <Button> element immediately following the <EditText> element.

**Step 3:** Set the button's width and height attributes to "wrap\_content" so the button is only as big as necessary to fit the button's text label.

**Step 4:** Define the button's text label with the android:text attribute; set its value to the button\_send string resource you defined in the previous section.

Your <LinearLayout> should look like this:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:showIn="@layout/activity_my">
    <EditText android:id="@+id/edit_message"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="@string/edit_message"/>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_send"/>
</LinearLayout>
```

The layout is currently designed so that both the EditText and Button widgets are only as big as necessary to fit their content, as given in figure 8.3.

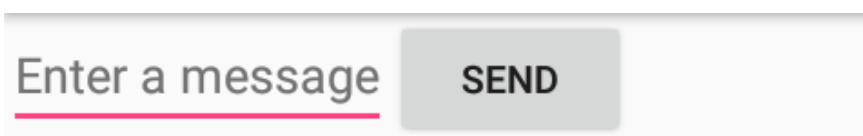


Figure 8.4. The EditText and Button widgets.

(Source: <https://developer.android.com/training/basics/firstapp/building-ui.html>)



### 8.5.4 Check Your Progress



What is the importance of XML-based layouts?

Explain LinearLayout in Android.

Now we will see how to design the same UI with the Layout Editor.

---

## 8.6 DESIGN A UI WITH LAYOUT EDITOR

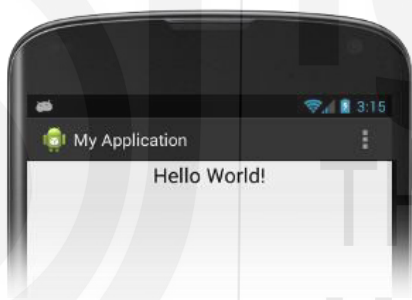
---

Android Studio offers an advanced layout editor that allows you to drag and-drop widgets into your layout and preview your layout while editing the XML.

Within the layout editor, you can switch between the Text view, where you edit the XML file as text, and the Design view. Just click the appropriate tab at the bottom of the window to display the desired editor.

### Editing in the Text View

While editing in the Text view, you can preview the layout on devices by opening the Preview pane and you can modify the preview by changing various options at the pane, including the preview device, layout theme, platform version and more. figure 8.5 gives how your application preview.



**Figure 8.5** Previewing your app

(Source: <https://developer.android.com/studio/write/layout-editor.html>)

Next, we will focus on how you can switch to graphical editor and edit user interfaces in a design view.

### Editing in the Design View

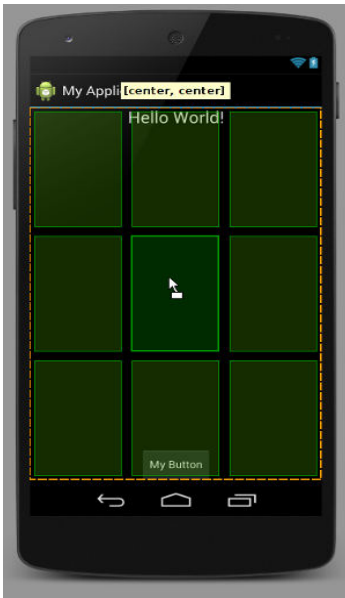
You can switch to the graphical editor by clicking Design at the bottom of the window. While editing in the Design view, you can show and hide the widgets available to drag-and-drop by clicking Palette on the window.

Clicking Designer reveals a panel with a layout hierarchy and a list of properties for each view in the layout.

When you drag a widget into the graphical layout for your app, the display changes to help you place the widget. What you see depends on the type of layout.

*Example: Dragging a widget into a FrameLayout*

It displays a grid to help you place the widget, as shown in figure 8.6.



**Figure 8.6** Grid layout to place a widget.

(Source: <https://developer.android.com/studio/write/layout-editor.html#design-view>)

In the next section, we will see how to manage touch events in an android application.

---

## 8.7 MANAGING TOUCH EVENTS IN A VIEWGROUP

---

Handling touch events in a ViewGroup takes special care, because it is common for a ViewGroup to have children that are targets for different touch events than the ViewGroup itself. To make sure that each view correctly receives the touch events intended for it, override the `onInterceptTouchEvent()` method.

### Intercept Touch Events in a ViewGroup

The `onInterceptTouchEvent()` method is called whenever a touch event is detected on the surface of a ViewGroup, including on the surface of its children.

If `onInterceptTouchEvent()` returns true, the `MotionEvent` is intercepted, meaning it will not be passed on to the child, but rather to the `onTouchEvent()` method of the parent.

The `onInterceptTouchEvent()` method gives a parent the chance to see any touch event before its children do.

In the following snippet, the class `MyViewGroup` extends `ViewGroup`. `MyViewGroup` contains multiple child views. If you drag your finger across a child view horizontally, the child view should no longer get touch events, and `MyViewGroup` should handle touch events by scrolling its contents.

However, if you press buttons in the child view, or scroll the child view vertically, the parent shouldn't intercept those touch events, because the child is the intended target. In those cases, `onInterceptTouchEvent()` should return false, and `MyViewGroup.onTouchEvent()` won't be called.

```

public class MyViewGroup extends ViewGroup {

    private int mTouchSlop;
    ...
    ViewConfiguration vc
    = ViewConfiguration.get(view.getContext());
    mTouchSlop = vc.getScaledTouchSlop();

    ...

    @Override
    public boolean onInterceptTouchEvent(MotionEvent ev) {

        /* This method JUST determines whether we want to intercept the motion.
        * If we return true, onTouchEvent will be called and we do the actual scrolling
        there. */

        final int action = MotionEventCompat.getActionMasked(ev);

        // Always handle the case of the touch gesture being complete.
        if (action == MotionEvent.ACTION_CANCEL || action
            == MotionEvent.ACTION_UP) {
            // Release the scroll.
            mIsScrolling = false;
            return false;

            // Do not intercept touch event, let the child handle it
        }

        switch (action) {
            case MotionEvent.ACTION_MOVE: {
                if (mIsScrolling) {
                    // We're currently scrolling, so yes, intercept the
                    // touch event!
                    return true;
                }

                // If the user has dragged her finger horizontally more
                // than the touch slop, start the scroll
                // left as an exercise for the reader

                final int xDiff = calculateDistanceX(ev);

                // Touch slop should be calculated using ViewConfiguration
                // constants.
                if (xDiff > mTouchSlop) {
                    // Start scrolling!
                    mIsScrolling = true;
                    return true;
                }
                break;
            }
            ...
        }

        // In general, we don't want to intercept touch events.
        // They should be handled by the child view.
    }
}

```

```

        return false;
    }

    @Override
    public boolean onTouchEvent (MotionEvent ev) {

        // Here we actually handle the touch event (e.g. if the
        // action is ACTION_MOVE, scroll this container).
        // This method will only be called if the touch event was // intercepted
        inonInterceptTouchEvent

        ...}}

```

In this section, we discussed managing touch Events in a ViewGroup. Next we will see what are the best practices of UI.

---

## 8.8 BEST PRACTICES FOR USER INTERFACE

---

Android provides a flexible framework for UI design that allows your app to display different layouts for different devices, create custom UI widgets, and even control aspects of the system UI outside your app's window.

- Designing for Multiple Screens - A user interface that's flexible enough to fit perfectly on any screen and create different interaction patterns that are optimized for different screen sizes.
- Adding the App Bar - Use the support library's toolbar widget to implement an app bar that displays properly on a wide range of devices.
- Showing Pop-Up Messages - Use the support library's Snackbar widget to display a brief pop-up message.
- Creating Custom View - Build custom UI widgets that are interactive and smooth.
- Creating Backward-Compatible UIs - Use UI components and other APIs from the more recent versions of Android while remaining compatible with older versions of the platform.
- Implementing Accessibility - Make apps accessible to users with vision impairment or other physical disabilities.
- Managing the System UI - Hide and show status and navigation bars across different versions of Android, while managing the display of other screen components.
- Creating Apps with Material Design - Implement material design on Android.

---

## 8.9 SUMMARY

---



In this unit, we discussed the fundamentals of user interface design. In addition, we discussed how to build a user interface using Android layouts and the best practices for user interface design. Android provides a flexible framework for UI design. So in this unit we discussed how to display an application in different layouts for different devices and how to create custom UI widgets.