

Chapter 1

Introduction to Android

1.1 What is Android?

- Android is an open-source operating system designed primarily for mobile devices such as smartphones and tablets. It was developed by Google and the Open Handset Alliance, a consortium of hardware, software, and telecommunication companies. Android is based on the Linux kernel and is written primarily in the Java and C++ programming languages.

Key features of Android:

- **Open Source:** Android is an open-source platform, which means its source code is freely available to developers. This allows manufacturers to customize the operating system to suit their devices and enables a vast community of developers to contribute to its development.
 - **Application Ecosystem:** Android provides access to the Google Play Store, a vast marketplace where users can download and install applications for various purposes, including productivity, entertainment, education, gaming, and more.
 - **User Interface:** Android offers a user-friendly interface that allows users to interact with their devices using touch gestures, voice commands, and other input methods.
 - **Customization:** Android provides extensive customization options for device manufacturers and users. Manufacturers can create customized user interfaces (UI skins), and users can personalize their devices with widgets, themes, and wallpapers.
 - **Multitasking:** Android supports multitasking, enabling users to run multiple applications simultaneously and switch between them seamlessly.
 - **Notifications:** Android's notification system allows apps to send timely updates and alerts to users, providing them with relevant information without the need to open the app.
 - **Connectivity:** Android devices support various connectivity options, such as Wi-Fi, Bluetooth, NFC, and mobile data, allowing users to stay connected to the internet and other devices.
 - **Integration with Google Services:** Android tightly integrates with Google's ecosystem, including services like Google Search, Google Maps, Gmail, Google Drive, and more, enhancing the overall user experience.
 - **Security:** Android incorporates multiple layers of security, including app sandboxing, permission-based access, and regular security updates, to protect user data and privacy.
 - **Frequent Updates:** Google releases regular updates to the Android platform, providing new features, performance improvements, and security patches.
- Due to its widespread adoption and open nature, Android has become the most popular mobile operating system globally, powering a vast majority of smartphones and tablets across different brands and price ranges. Additionally, Android has also expanded its reach into other devices like smart TVs, wearables, and Internet of Things (IoT) devices.

1.1.1 History of Android

- The history of Android dates back to the early 2000s when a small startup called Android Inc. was founded in October 2003 by Andy Rubin, Rich Miner, Nick Sears, and Chris White. Let's take a chronological look at the key milestones in the history of Android:
 - **Android Inc. and Early Development (2003-2005):** Android Inc. initially aimed to develop an operating system for digital cameras. However, the team later realized the potential of their technology in the rapidly growing mobile market. They shifted their focus to developing a mobile operating system based on the Linux kernel. The early vision for Android was to create an open and customizable platform that could provide a better user experience for mobile devices.
 - **Acquisition by Google (August 2005):** In August 2005, Google, led by its then-CEO Eric Schmidt, acquired Android Inc. The acquisition raised speculation about Google's plans for entering the mobile industry. At that time, Google did not reveal its intentions publicly, but the acquisition of Android Inc. marked the beginning of the Android journey within Google.
 - **Android Open Source Project (AOSP) (2007):** On November 5, 2007, Google, along with several other technology companies, including HTC, Samsung, LG, Motorola, and T-Mobile, announced the formation of the Open Handset Alliance (OHA). The OHA was a consortium of companies collaborating on the development of Android as an open-source mobile platform. The core of Android, known as the Android Open Source Project (AOSP), was made freely available to developers, allowing them to customize, modify, and distribute the operating system as they saw fit.
 - **First Android Phone (September 2008):** The first commercial Android-powered smartphone, the HTC Dream (also known as the T-Mobile G1 in the United States), was released on September 23, 2008. It ran on Android 1.0 and was the first device to showcase the capabilities of the Android platform.
 - **Android Market and App Ecosystem (October 2008):** In October 2008, the Android Market (now known as Google Play Store) was launched, providing a centralized platform for users to download and install Android applications. The app ecosystem quickly grew, and developers began creating a wide range of applications for various purposes.
 - **Version Updates and Platform Growth (2009-2013):** Over the next few years, Android underwent several major updates, each introducing new features and improvements. Notable versions during this period included Cupcake (1.5), Donut (1.6), Éclair (2.0-2.1), Froyo (2.2), Gingerbread (2.3), Honeycomb (3.0-3.2), Ice Cream Sandwich (4.0), Jelly Bean (4.1-4.3), and KitKat (4.4).
 - **Android's Dominance (2014-present):** From 2014 onwards, Android solidified its position as the leading mobile operating system globally, surpassing other platforms in terms of market share. Android continued to evolve with major updates like Lollipop (5.0-5.1), Marshmallow (6.0), Nougat (7.0-7.1), Oreo (8.0-8.1), Pie (9), Android 10, and Android 11. Google's focus on improving user experience, security, and performance has been evident in each new version.
 - **Android Beyond Smartphones (2010s):** As Android matured, it expanded its reach beyond smartphones and tablets. Android was adapted for use in other devices, such as smart TVs, smartwatches, automotive infotainment systems, and IoT devices, making it one of the most versatile operating systems.
- Today, Android continues to be a dominant force in the mobile industry, powering a vast array of devices and serving billions of users worldwide. Its open-source nature and large developer community have contributed to its ongoing success and innovation.

1.2 What is Open Handset Alliance (OHA)?

- The Open Handset Alliance (OHA) was a consortium of technology companies, led by Google, that came together to develop and promote the Android operating system. The alliance was announced in November 2007 and played a crucial role in the rapid growth and success of the Android platform.
- The primary objective of the Open Handset Alliance was to create an open and standardized mobile platform that would foster innovation and collaboration among its members. By establishing common standards and guidelines, OHA aimed to accelerate the development of mobile devices and applications, providing a better user experience for consumers.

Key points about the Open Handset Alliance (OHA):

- **Membership:** The alliance consisted of a diverse group of companies, including handset manufacturers, wireless carriers, semiconductor companies, software developers, and other technology stakeholders. Some of the notable members of OHA included Google, HTC, Samsung, LG, Motorola, Qualcomm, Intel, and T-Mobile.
 - **Open Source Philosophy:** OHA's central philosophy was to develop the Android platform as an open-source project. This meant that the source code of Android was made freely available to developers, allowing them to modify, customize, and distribute the operating system according to their needs.
 - **Android Development:** OHA members collaborated on the development of the Android platform, contributing to its features, functionalities, and bug fixes. Google provided the initial Android codebase, derived from the acquisition of a startup called Android Inc., and the members collectively worked on its evolution.
 - **Google's Role:** Although OHA had multiple members, Google played a leading role in shaping the Android ecosystem and providing strategic direction. Google's Android team managed the core development, released major updates, and maintained the Android Open Source Project (AOSP) repository.
 - **Device Ecosystem:** OHA's efforts led to the rapid growth of the Android device ecosystem. Manufacturers were able to build a wide range of Android-based smartphones and tablets, catering to different market segments and consumer preferences.
 - **Google Play Store:** As part of the Android ecosystem, OHA members contributed to the development and integration of the Google Play Store (formerly known as Android Market), which became the primary app distribution platform for Android applications.
- In 2017, Google shifted its approach to Android development. While the Open Handset Alliance was not formally disbanded, Google gradually reduced its emphasis on the OHA branding and made Android a more centralized project under its own umbrella. The principles of openness and collaboration, however, continue to be key characteristics of the Android platform.

1.2.1 History of Open Handset Alliance (OHA)?

- The Open Handset Alliance (OHA) was formed in November 2007 and played a significant role in the development and promotion of the Android operating system. Let's take a closer look at the history of OHA:
 - **Early Development of Android:** The history of the Open Handset Alliance can be traced back to 2003 when a startup called Android Inc. was founded by Andy Rubin, Rich Miner, Nick Sears, and Chris White. Android Inc. aimed to develop an operating system for digital cameras initially but later shifted its focus to mobile devices. The company worked on an open-source operating system for smartphones, with an emphasis on providing a more connected and user-friendly mobile experience.
 - **Acquisition by Google:** In August 2005, Google acquired Android Inc., bringing its team and technology under Google's umbrella. The acquisition fueled speculations about Google's intentions in the mobile industry and its plans to develop a mobile operating system.
 - **Formation of the Open Handset Alliance:** On November 5, 2007, Google, along with 33 other technology companies, announced the formation of the Open Handset Alliance. The founding members included prominent names such as HTC, Samsung, LG, Motorola, Qualcomm, Texas Instruments, and T-Mobile, among others. The alliance aimed to collaborate on the development of an open and standardized mobile platform, which later became known as Android.

- **Release of the First Android Phone:** On September 23, 2008, the first commercial Android-powered smartphone, the HTC Dream (also known as the T-Mobile G1 in the United States), was released. It ran on Android 1.0 and marked the beginning of Android's presence in the mobile market.
- **Growth of the Android Ecosystem:** With the foundation of the Open Handset Alliance, the Android ecosystem experienced rapid growth. More manufacturers joined the alliance, leading to a diverse range of Android-powered devices across different price points and form factors.
- **Major Android Versions:** Over the years, Android underwent several major updates, each bringing new features and improvements. Some notable Android versions include Cupcake (1.5), Donut (1.6), Éclair (2.0-2.1), Froyo (2.2), Gingerbread (2.3), Honeycomb (3.0-3.2), Ice Cream Sandwich (4.0), Jelly Bean (4.1-4.3), KitKat (4.4), Lollipop (5.0-5.1), Marshmallow (6.0), Nougat (7.0-7.1), Oreo (8.0-8.1), Pie (9), Android 10, and Android 11.

1.3 Version of Android OS

1. Android 1.0 (Alpha, Beta)

- Android 1.0 was the first version released commercially, but before the official release, there were alpha and beta versions circulated among developers.
- It was launched on September 23, 2008.
- Features included basic smartphone functionalities like web browsing, email, and the ability to install third-party applications.
- The user interface was basic and lacked many features present in later versions.

2. Android 1.5 Cupcake

- Released in April 2009.
- Introduced an on-screen keyboard, video recording, and playback support.
- Added widgets for the home screen, allowing users to customize their interfaces.

3. Android 1.6 Donut

- Released in September 2009.
- Introduced improved search functionality, including the ability to search within apps.
- Added support for CDMA networks, enabling Android to work on more carriers.

4. Android 2.0/2.1 Éclair

- Android 2.0 was released in October 2009, followed by 2.1 in January 2010.
- Introduced support for multiple accounts and HTML5 support in the browser.
- Added features like live wallpapers, Microsoft Exchange email support, and improved camera functionalities.

5. Android 2.2 Froyo (Frozen Yogurt)

- Released in May 2010.
- Introduced performance optimizations and a Just-In-Time (JIT) compiler for improved speed.
- Added support for Adobe Flash in the web browser.

6. Android 2.3 Gingerbread

- Released in December 2010.
- Brought a refined user interface and improved text input and copy-paste functionality.
- Added support for NFC (Near Field Communication) and SIP (Session Initiation Protocol) VoIP.

7. Android 3.0/3.1/3.2 Honeycomb

- Honeycomb was designed specifically for tablets and larger-screen devices.
- Introduced a revamped user interface with a holographic design.
- Optimized for multitasking and featured an improved web browser and updated Google apps.

8. Android 4.0 Ice Cream Sandwich

- Released in October 2011.
- Unified the smartphone and tablet interfaces, providing a consistent user experience.
- Introduced virtual buttons in the system bar for navigation.

9. Android 4.1/4.2/4.3 Jelly Bean

- Jelly Bean debuted in July 2012 (4.1) and received subsequent updates.
- Introduced "Project Butter" for smoother performance and responsiveness.
- Added features like Google Now, enhanced notifications, and multiple user accounts on tablets.

10. **Android 4.4 KitKat**

- Released in October 2013.
- Focused on improving performance and efficiency on lower-end devices.
- Introduced a new immersive mode, updated phone app, and support for wireless printing.

11. **Android 5.0/5.1 Lollipop**

- Lollipop was released in November 2014 (5.0) and received updates.
- Introduced the "Material Design" language for a more visually appealing and consistent UI.
- Added features like ART runtime, screen pinning, and improved battery life through "Project Volta."

12. **Android 6.0 Marshmallow**

- Released in October 2015.
- Introduced "Doze" mode for better battery optimization during device inactivity.
- Added runtime permissions, USB Type-C support, and Google Now on Tap.

13. **Android 7.0/7.1 Nougat**

- Nougat was released in August 2016 (7.0) and received updates.
- Introduced split-screen multitasking, improved notifications, and direct reply functionality.
- Included Daydream VR support and app shortcuts.

14. **Android 8.0/8.1 Oreo**

- Released in August 2017 (8.0) and received updates.
- Introduced picture-in-picture mode, notification dots, and adaptive icons.
- Focused on improving system performance and background app limitations.

15. **Android 9 Pie**

- Released in August 2018.
- Introduced gesture navigation, Digital Wellbeing features, and Adaptive Battery.
- Focused on AI-powered enhancements and overall user experience.

16. **Android 10**

- Released in September 2019.
- Introduced system-wide dark mode, enhanced privacy controls, and improved gesture navigation.
- Added support for 5G connectivity and foldable devices.

17. **Android 11**

- Released in September 2020.
- Introduced chat bubbles, improved media controls, and improved 5G support.
- Focused on enhancing privacy and providing more control over app permissions.

18. **Android 12**

- Android 12 launched in October 2021 and was first rolled out to Google Pixel devices.
- The most significant update in Android 12 is the introduction of "Material You," a complete overhaul of the Material Design standard.
- Material You brings a fresh look and customization options to the Android interface, extending to apps, Google services, Chromebooks, Smart Displays, and wearables.
- Android 12 also focuses on improving performance, security, and privacy, with enhanced controls over data access and a new isolated section for AI features.

19. **Android 13**

- Android 13 was launched in August 2022.
- It introduces a new interface design for tablets and foldable phones, focusing on improving the large-screen experience.
- Enhancements include a new app optimization framework, multitasking features, and a ChromeOS-like desktop-style taskbar.
- Android 13 may support a new multipurpose product that can function as a Smart Display and detach as a tablet.
- On regular phones, the update is less significant, with minor visual refinements and improvements to clipboard, QR code scanning, privacy, security, and performance.

20. **Android 14**

- Android 14's first developer preview was announced in February 2023.
- It is expected to have two developer preview releases and four beta versions leading to the final release in late summer or early fall.
- Early versions focus on system performance, privacy, and optimizations for tablets and foldable phones.
- Surface-level enhancements include a new system for visual alerts around notifications and an improved back gesture.
- A new feature allows users to "clone" an app and run two separate versions simultaneously for different profiles.
- More features and updates are likely to be revealed in the coming months as Google continues its development of Android 14.

21. Each Android version builds upon the previous one, offering new features and improvements, while maintaining compatibility with older apps and devices. Google continues to release regular updates to improve security, performance, and user experience on Android devices ¹.

¹<https://www.computerworld.com/article/3235946/android-versions-a-living-history-from-1-0-to-today.html>

1.4 Features and Benefits of Android

1. **Open Source:** Android is an open-source platform, allowing developers to access the source code freely, modify it, and contribute to its development. This fosters innovation and collaboration within the Android community.
2. **Diverse App Ecosystem:** Android provides access to the Google Play Store, offering a vast selection of applications for various purposes, including productivity, entertainment, communication, gaming, and more.
3. **User-Friendly Interface:** Android offers an intuitive and user-friendly interface that allows users to navigate and interact with their devices easily.
4. **Customization Options:** Android provides extensive customization options for device manufacturers, allowing them to create customized user interfaces (UI skins) and experiences tailored to their devices.
5. **Multitasking:** Android supports multitasking, enabling users to run multiple applications simultaneously and switch between them seamlessly.
6. **Widgets:** Android supports interactive widgets on the home screen, allowing users to access specific app functionalities without opening the app itself.
7. **Notifications:** Android's notification system provides timely updates and alerts to users, enhancing communication and keeping users informed.
8. **Connectivity:** Android devices support various connectivity options, including Wi-Fi, Bluetooth, NFC, and mobile data, allowing users to stay connected to the internet and other devices.
9. **Google Services Integration:** Android tightly integrates with Google's ecosystem, offering access to services like Google Search, Google Maps, Gmail, Google Drive, and more, enhancing productivity and convenience.
10. **Seamless Cloud Integration:** Android devices offer seamless integration with cloud services, making it easy for users to back up and access their data across devices.
11. **Regular Updates:** Google releases regular updates to the Android platform, providing new features, performance improvements, and security patches.
12. **Support for Multiple Devices:** Android is not limited to smartphones; it extends its reach to tablets, smartwatches, smart TVs, and other devices, creating a unified ecosystem.
13. **Voice Assistants:** Android devices come with Google Assistant, a powerful voice-activated digital assistant that helps users perform tasks using voice commands.
14. **Enhanced Security:** Android includes multiple layers of security, such as app sandboxing, permission-based access, and regular security updates, to protect user data and privacy.
15. **Cost and Device Variety:** Android-powered devices come in a wide range of price points and hardware specifications, making them accessible to a broad audience.
16. **Developer-Friendly:** Android provides a developer-friendly environment with various tools, APIs, and resources to build and publish apps effectively.
17. **Global Market Share:** Android is the most popular mobile operating system globally, which means developers can reach a vast user base.

These features and advantages have contributed to Android's widespread adoption and position as the leading mobile operating system, powering billions of devices worldwide.

1.5 Comparing Android with Other Platforms

When comparing Android with other mobile platforms, such as iOS, Windows Phone (now discontinued), or BlackBerry OS (now transitioned to Android), several factors come into play. Here's a detailed analysis of how Android stacks up against other platforms:

1. Open Source vs. Closed Ecosystem:

- Android is an open-source platform, allowing developers to access and modify the source code freely. This fosters innovation and encourages a vast community of developers to contribute to its development.
- iOS, on the other hand, is a closed ecosystem developed and controlled by Apple. The source code is not open to the public, and only Apple-approved developers can contribute to the platform.

2. App Ecosystem:

- Android has a diverse app ecosystem, with millions of apps available on the Google Play Store, catering to various needs and interests.
- iOS also offers a vast app ecosystem, with millions of apps available on the Apple App Store. While the total number of apps is similar, some exclusive apps may be available only on one platform.

3. Customization Options:

- Android provides extensive customization options for device manufacturers, allowing them to create customized user interfaces (UI skins) and tailor the Android experience to their devices.
- iOS offers limited customization, with a consistent user interface across all Apple devices.

4. Device Variety and Price Range:

- Android is available on devices from numerous manufacturers, providing a wide range of choices in terms of design, features, and price points, making it accessible to a broad audience.
- iOS is limited to Apple's own devices, which are generally priced at a higher range, making them less accessible to budget-conscious consumers.

5. Fragmentation:

- Android's open nature and device diversity can lead to fragmentation, where different devices run on different versions of Android and have varying hardware capabilities. This can pose challenges for developers to optimize their apps for the entire Android ecosystem.
- iOS, with a more closed ecosystem and limited device variations, tends to have less fragmentation, making it easier for developers to optimize their apps.

6. Update Availability:

- Android updates can be slow to reach all devices, especially for devices from different manufacturers, carriers, and regions. This can lead to delayed access to the latest features and security patches.
- iOS updates are typically available for all supported devices simultaneously, providing a more consistent user experience across the platform.

7. Integration with Ecosystem and Services:

- Android tightly integrates with Google's ecosystem and services, such as Google Search, Google Maps, Gmail, and Google Drive, offering seamless access to these services.
- iOS integrates with Apple's ecosystem and services, such as Siri, Apple Maps, iCloud, and iMessage, providing a seamless user experience for Apple users.

8. Voice Assistants

- Android devices come with Google Assistant, a powerful voice-activated digital assistant that provides a wide range of functionalities and integrates well with Google's services.
- iOS devices come with Siri, Apple's voice assistant, which also provides a range of features and integrates with Apple's ecosystem.

9. Security and Privacy

- Both Android and iOS have strong security measures, but iOS is often considered more secure due to its closed ecosystem and stringent app review process.
- Android has improved its security with features like Google Play Protect and regular security updates, but its open nature can still pose security challenges.

10. Market Share

- Android has a larger global market share compared to iOS, making it the most popular mobile operating system worldwide.
- iOS, while having a smaller market share, tends to have a more affluent user base, leading to higher app revenue for developers.

In conclusion, the choice between Android and other mobile platforms depends on factors like customization options, device variety, ecosystem integration, security, and user preferences. Android's open nature, device diversity, and global market share make it a popular choice for a wide range of users, while other platforms like iOS offer a more closed ecosystem, consistent user experience, and strong app revenue potential for developers.

Android Architecture

Android is an open source, Linux-based software stack created for a wide array of devices and form factors. Figure 1 shows the major components of the Android platform.

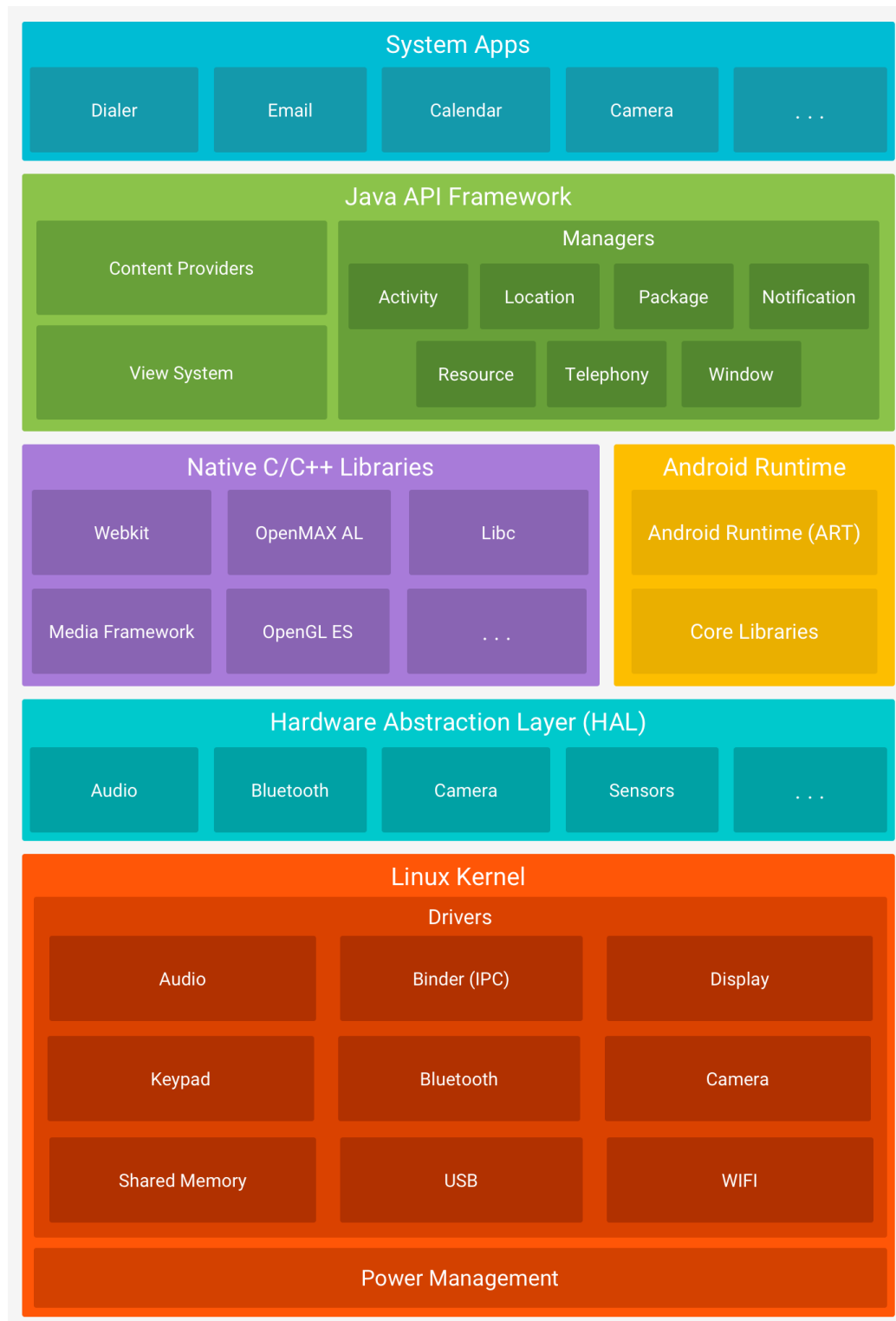


Figure 1. The Android software stack.

Linux kernel

The foundation of the Android platform is the Linux kernel. For example, the Android Runtime (ART) ([#art](#)) relies on the Linux kernel for underlying functionalities such as threading and low-level memory management.

Using a Linux kernel lets Android take advantage of key security features (<https://source.android.com/security/overview/kernel-security.html>) and lets device manufacturers develop hardware drivers for a well-known kernel.

Hardware abstraction layer (HAL)

The hardware abstraction layer (HAL) (<https://source.android.com/devices/architecture/hal>) provides standard interfaces that expose device hardware capabilities to the higher-level Java API framework ([#api-framework](#)). The HAL consists of multiple library modules, each of which implements an interface for a specific type of hardware component, such as the camera (<https://source.android.com/devices/camera/index.html>) or Bluetooth (<https://source.android.com/devices/bluetooth.html>) module. When a framework API makes a call to access device hardware, the Android system loads the library module for that hardware component.

Android runtime

For devices running Android version 5.0 (API level 21) or higher, each app runs in its own process and with its own instance of the Android Runtime (ART) (<https://source.android.com/devices/tech/dalvik/index.html>). ART is written to run multiple virtual machines on low-memory devices by executing Dalvik Executable format (DEX) files, a bytecode format designed specifically for Android that's optimized for a minimal memory footprint. Build tools, such as d8 ([/studio/command-line/d8](#)), compile Java sources into DEX bytecode, which can run on the Android platform.

Some of the major features of ART include the following:

- Ahead-of-time (AOT) and just-in-time (JIT) compilation
- Optimized garbage collection (GC)
- On Android 9 (API level 28) and higher, conversion ([/about/versions/pie/android-9.0#art-aot-dex](#)) of an app package's DEX files to more compact machine code

- Better debugging support, including a dedicated sampling profiler, detailed diagnostic exceptions and crash reporting, and the ability to set watchpoints to monitor specific fields

Prior to Android version 5.0 (API level 21), Dalvik was the Android runtime. If your app runs well on ART, then it can work on Dalvik as well, but the reverse might not be true (/guide/practices/verifying-apps-art).

Android also includes a set of core runtime libraries that provide most of the functionality of the Java programming language, including some Java 8 language features (/guide/platform/j8-jack), that the Java API framework uses.

Native C/C++ libraries

Many core Android system components and services, such as ART and HAL, are built from native code that requires native libraries written in C and C++. The Android platform provides Java framework APIs to expose the functionality of some of these native libraries to apps. For example, you can access OpenGL ES (/develop/ui/views/graphics/opengl/about-opengl) through the Android framework's Java OpenGL API (/reference/android/opengl/package-summary) to add support for drawing and manipulating 2D and 3D graphics in your app.

If you are developing an app that requires C or C++ code, you can use the Android NDK (/ndk) to access some of these native platform libraries (/ndk/guides/stable_apis) directly from your native code.

Java API framework

The entire feature-set of the Android OS is available to you through APIs written in the Java language. These APIs form the building blocks you need to create Android apps by simplifying the reuse of core, modular system components and services, which include the following:

- A rich and extensible view system (/guide/topics/ui/overview) you can use to build an app's UI, including lists, grids, text boxes, buttons, and even an embeddable web browser
- A resource manager (/guide/topics/resources/overview), providing access to non-code resources such as localized strings, graphics, and layout files

- A [notification manager](/guide/topics/ui/notifiers/notifications) (/guide/topics/ui/notifiers/notifications) that enables all apps to display custom alerts in the status bar
- An [activity manager](/guide/components/activities/intro-activities) (/guide/components/activities/intro-activities) that manages the lifecycle of apps and provides a common [navigation back stack](/guide/components/tasks-and-back-stack) (/guide/components/tasks-and-back-stack)
- [Content providers](/guide/topics/providers/content-providers) (/guide/topics/providers/content-providers) that enable apps to access data from other apps, such as the Contacts app, or to share their own data

Developers have full access to the same [framework APIs](/reference/packages) (/reference/packages) that Android system apps use.

System apps

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

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

Content and code samples on this page are subject to the licenses described in the [Content License](/license) (/license). Java and OpenJDK are trademarks or registered trademarks of Oracle and/or its affiliates.

Last updated 2023-05-04 UTC.

1.6 Android Architecture

The architecture of Android is a layered and flexible design that allows the operating system to run on various hardware platforms while providing a consistent user experience. It is based on the Linux kernel and follows a modular structure, comprising multiple layers that interact to deliver the functionalities of Android. Let's delve into the details of the Android architecture:

- **Linux Kernel:**

- The foundation of the Android architecture is the Linux kernel, which handles core system functionalities, such as memory management, process management, device drivers, and networking.
- The Linux kernel abstracts the hardware components and provides an interface for higher-level components to interact with the hardware.

- **Inter-process Communication (IPC):**

- Android employs Interprocess Communication mechanisms to facilitate communication between different components and apps running in different processes.

- **Hardware Abstraction Layer (HAL):**

- Above the Linux kernel is the Hardware Abstraction Layer (HAL), which acts as an interface between the Android framework and the underlying hardware drivers.
- The HAL provides a standardized API that allows the Android framework to interact with various hardware components, such as camera, sensors, audio, and display, regardless of the underlying hardware implementation.

- **Android Runtime (ART):**

- Android originally used the Dalvik Virtual Machine (DVM) for executing Android applications. However, since Android 5.0 (Lollipop), it switched to the Android Runtime (ART) for improved performance and efficiency.
- ART is a runtime environment that compiles Android application code into native machine code at the time of installation, rather than interpreting it at runtime as in the case of DVM. This ahead-of-time (AOT) compilation results in faster app performance and lower resource consumption.

- **Native Libraries:**

- Android includes a set of native libraries written in C/C++, which provide essential functionality to the system and apps.
- These libraries cover a wide range of functionalities, including graphics rendering (OpenGL), media processing, SQLite database management, and more.

- **Java API Framework:**

- The Java API Framework is a collection of higher-level libraries and services built on top of the native libraries, providing the core functionalities of Android to developers.
- It offers a rich set of APIs for app development, enabling developers to access system features, interact with hardware, and create user interfaces.
- The Java API Framework includes components such as Activity Manager, Content Provider, Location Manager, and Notification Manager, among others.

- **Application Layer:**

- At the top layer of the Android architecture is the application layer, where user apps and system apps reside.
- This layer is responsible for providing the user interface and delivering the app's functionalities to the user.

- **System Apps and User Apps:**

- Android comes with a set of pre-installed system apps, such as the dialer, messaging, calendar, contacts, and settings apps.
- User apps are the apps that users download and install from the Google Play Store or other sources. These apps interact with the Android Framework to access system resources and provide various functionalities to users.

- **Application Components:**

- Android apps are composed of various components, including Activities, Services, Content Providers, and Broadcast Receivers, each serving specific purposes in the app's lifecycle and functionality.

In summary, the Android architecture is built upon the Linux kernel, with the Android Framework and native libraries providing essential functionalities. The layered design allows Android to run on diverse hardware platforms, while the modular structure facilitates app development and enables consistent user experiences across different devices

1.7 Android Development Tools

Android development tools refer to the software and frameworks used to build applications for Android devices. Here is an overview of some key Android development tools:

- **Android Studio:** Android Studio is the official integrated development environment (IDE) for Android app development. It provides a comprehensive set of tools and features to streamline the development process. Some key features include:
 - Code editor with intelligent code completion, code analysis, and refactoring capabilities.
 - Visual layout editor for designing app interfaces using drag-and-drop components.
 - Debugger for identifying and fixing issues in the code.
 - Profiler for analyzing app performance and memory usage.
 - Emulator for testing apps on different virtual devices.
 - Built-in support for version control systems like Git.
- **Android SDK (Software Development Kit):** The Android SDK is a set of tools and libraries that developers use to create Android apps. It includes:
 - Android API libraries for accessing various platform features, such as UI components, sensors, network connectivity, and more.
 - Command-line tools for tasks like building and deploying apps.
 - Emulator for testing apps on virtual devices.
 - Platform tools for debugging and managing devices.
- **Gradle:**
 - Gradle is an open-source build automation system used to build Android apps.
 - It handles dependencies, configures builds, and packages the app. Build files are written in Java, Groovy, or Kotlin.
 - It helps compile code, package resources, and create APK (Android Package) files.
 - Developers can customize build configurations using the Gradle script, enabling tasks like code signing, obfuscation, and more.
- **Android Virtual Device (AVD) Manager or Android Emulator:**
 - The Android Emulator is a tool within Android Studio that allows developers to test their apps on virtual Android devices.
 - Developers can create and configure different virtual devices with varying specifications, screen sizes, and Android versions for comprehensive testing.
- **Android Debug Bridge (ADB):**
 - ADB is a command-line tool that facilitates communication between a computer with emulated and physical Android devices.
 - It's used for various tasks, such as installing and debugging apps, transferring files, accessing device shell, and more.
- **Android Studio Profilers:**
 - Set of profiling tools built into Android Studio to measure CPU, memory and network usage of running apps. Helps optimize performance.
 - The Android Profiler in Android Studio helps developers monitor app performance and diagnose issues.
 - It offers insights into CPU usage, memory allocation, network activity, and more, allowing developers to optimize their apps for better efficiency.

- **Firebase:**

- Firebase is a comprehensive mobile development platform by Google that provides various tools and services for building, testing, and scaling Android apps.
- It includes features like authentication, real-time databases, cloud storage, analytics, crash reporting, and more.

- **Profiler**

- The Android Profiler in Android Studio helps developers monitor app performance and diagnose issues.
- It offers insights into CPU usage, memory allocation, network activity, and more, allowing developers to optimize their apps for better efficiency.

- **Android Jetpack**

- A collection of libraries and tools to accelerate Android development with recommended architecture guidance. Components for UI, navigation, notifications etc.

So in summary, Android Studio, Gradle, SDK, AVD and other tools form the core toolchain for building, testing and debugging Android apps efficiently. These are just a few of the many tools available for Android development. The combination of these tools and resources empowers developers to create high-quality, efficient, and user-friendly Android applications.

1.8 Android Studio

Android Studio is the official integrated development environment (IDE) for Android app development. It's designed to provide developers with a powerful and user-friendly platform for creating, testing, and optimizing Android applications. Here's a detailed overview of Android Studio and its features:

1. User Interface and Features:

- **Code Editor:** Android Studio offers a code editor with features like syntax highlighting, auto-completion, and code analysis. It supports multiple programming languages, including Java and Kotlin, the official languages for Android development.
- **Visual Layout Editor:** The visual layout editor enables developers to design app interfaces using a drag-and-drop approach. Developers can see real-time visual representations of their layouts as they build them.
- **Debugger:** The built-in debugger helps identify and fix issues in the code by allowing developers to set breakpoints, inspect variables, and step through code execution.
- **Profiler:** The Profiler tool helps analyze app performance by providing insights into CPU, memory, and network usage. This helps developers optimize their apps for better efficiency.
- **Logcat:** Logcat displays logs generated by the Android system and apps. It's crucial for debugging and understanding app behavior during development.
- **Version Control Integration:** Android Studio seamlessly integrates with version control systems like Git. This allows developers to manage code changes, collaborate with team members, and track project history.

2. Project Structure and Management:

- **Project Explorer:** The project explorer provides an organized view of project files, resources, and directories. Developers can easily navigate through their project's structure.
- **Gradle Build System:** Android Studio uses Gradle, a powerful build automation tool, to manage the build process of Android apps. Developers can customize build configurations and tasks using Gradle scripts.
- **Layout Editor:** The layout editor lets developers design XML layouts using a visual interface. This is particularly useful for creating complex layouts or tweaking UI elements manually.

3. Emulator and Device Testing:

- **Android Emulator:** Android Studio includes a powerful emulator that allows developers to test their apps on virtual Android devices. Emulators can simulate different devices with varying specifications and Android versions.
- **Physical Devices:** Developers can also test their apps on physical Android devices connected to their computers. Android Studio provides tools to deploy and debug apps on real devices.

4. Integration with Google Services:

- **Firebase Integration:** Android Studio offers seamless integration with Firebase, Google's mobile development platform. Firebase provides tools for authentication, real-time databases, cloud storage, analytics, and more.

5. Extensibility and Plugins:

- **Plugins:** Android Studio supports plugins that extend its functionality. Developers can install plugins for various purposes, such as code analysis, design enhancements, and integration with third-party libraries.

1.8.1 Significance of Android Studio for App Development

Android Studio has become the go-to integrated development environment (IDE) for Android app development due to its comprehensive set of features, tools, and resources that streamline the app creation process. Its importance in the world of app development is hard to overstate. Here are the key reasons why Android Studio is highly valued by developers:

1. **Official IDE:** Android Studio is the official IDE provided by Google for Android app development. This endorsement lends credibility and reliability to the platform, ensuring that developers are using a tool directly supported by the creators of the Android ecosystem.
2. **Feature-Rich Environment:** Android Studio offers a wide range of tools and features that cater to various aspects of app development, including coding, design, testing, debugging, and profiling. This comprehensive environment eliminates the need for developers to use separate tools for different tasks.
3. **Robust Code Editor:** The code editor in Android Studio provides intelligent code completion, syntax highlighting, real-time error checking, code analysis, and refactoring capabilities. These features significantly enhance developer productivity and code quality.
4. **Visual Design Tools:** Android Studio's visual layout editor allows developers to create app interfaces using a visual drag-and-drop approach. This feature simplifies the UI design process and helps developers see the results of their design decisions in real time.
5. **Efficient Debugging:** The debugger within Android Studio helps identify and fix issues in the code by enabling developers to set breakpoints, inspect variables, and step through code execution. This feature is essential for producing stable and error-free apps.
6. **Performance Profiling:** Android Studio's profiler tool offers insights into app performance, including CPU, memory, and network usage. Developers can optimize their apps to ensure efficient resource utilization, resulting in smoother user experiences.
7. **Emulator and Device Testing:** The built-in Android Emulator lets developers test their apps on virtual devices with various specifications and Android versions. This helps identify potential compatibility issues and ensures that the app works well across different devices.
8. **Gradle Build System:** Android Studio utilizes the Gradle build system, which allows developers to customize build configurations, manage dependencies, and automate repetitive tasks. This system streamlines the app building process and improves efficiency.
9. **Integration with Google Services:** Android Studio seamlessly integrates with Google services like Firebase, which provides a range of tools for authentication, real-time databases, cloud storage, analytics, and more. This integration simplifies the process of adding powerful features to apps.
10. **Community Support and Resources:** Being the official IDE, Android Studio has a vast community of developers, forums, tutorials, and resources available online. This makes it easier for developers to seek assistance, share knowledge, and stay up to date with the latest practices.
11. **Continuous Improvement:** Google regularly updates Android Studio to introduce new features, enhance existing ones, and address issues. Developers can expect ongoing support and improvements, ensuring that the IDE remains up to date with the evolving needs of the Android ecosystem.

1.8.2 Install Android Studio

Installing Android Studio involves a few simple steps. Here's a guide to help you install Android Studio on your computer:

1. Check System Requirements:

- Before you begin, ensure that your computer meets the system requirements for Android Studio.
- You can find the system requirements on the official Android Studio website.

2. Download Android Studio:

- Visit the official Android Studio download page at <https://developer.android.com/studio>.
- Click on the "Download Android Studio" button.
- Choose the appropriate version for your operating system (Windows, macOS, or Linux).

3. Install Android Studio

- **For Windows**
 - Run the downloaded installer file.
 - Follow the on-screen instructions. You can choose the installation location and whether to create desktop shortcuts.
 - Android Studio will be installed on your computer.
- **For macOS:**
 - Open the downloaded DMG file.
 - Drag the Android Studio icon to the "Applications" folder.
 - Android Studio will be installed on your Mac.
- **For Linux:**
 - Extract the downloaded archive to your desired location.
 - Navigate to the extracted directory and run the "studio.sh" script in the "bin" folder.
 - Follow the on-screen instructions to complete the installation.

4. Set Up Android Studio:

- Launch Android Studio from the location where you installed it. Android Studio will prompt you to import settings from a previous version if applicable. You can choose to import or not.
- Choose whether to participate in the Android Studio user experience program and click "OK."

5. Install Android SDK Components:

- Android Studio will guide you through the "Setup Wizard." You'll be prompted to install the Android SDK components necessary for app development.
- This includes SDK platforms, tools, and system images. Follow the prompts to install the recommended components.

6. Configure Android Emulator:

- If you plan to use the Android Emulator for testing, configure it by installing the system images for the Android versions you intend to test. The Setup Wizard will guide you through this process.

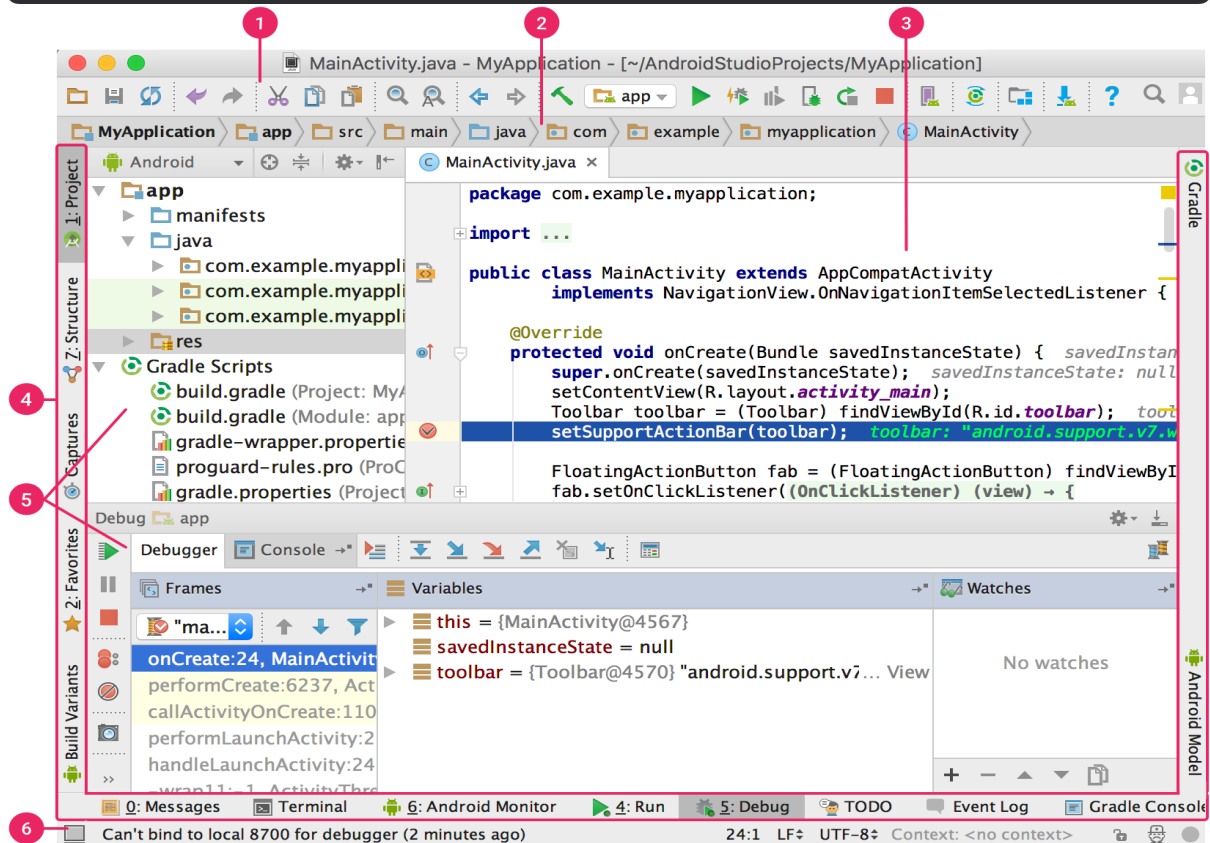
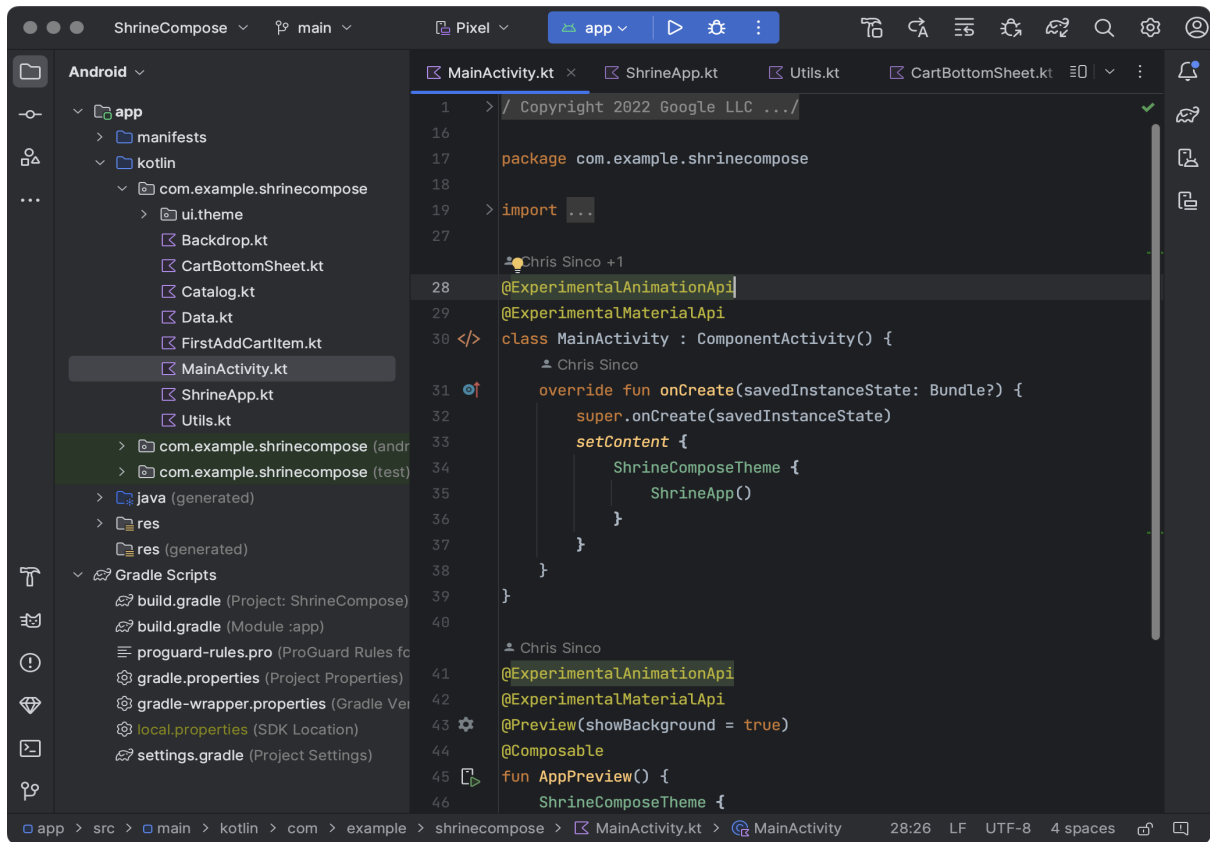
7. Complete Setup: Once you've installed the required components, Android Studio will be fully set up and ready for use.

8. Start Using Android Studio:

- Launch Android Studio from your computer's applications or the location where you installed it.
- Create a new Android project or open an existing project.
- You're now ready to start developing Android applications using Android Studio.

1.8.3 Complete Overview of Android Studio Main Window

The Android studio main window is consists of several logical areas. Let's discuss each area in detail.



1. Toolbar

- One can find the Toolbar section at the top of the android studio.
- It contains a wide range of functions including creating a new project which is inside the File Section, Reformat your code which is inside the Code Section, Rebuild your project which is inside the Build section, running your android app which is inside the Run section, and many more.

2. Navigation Bar

- The navigation bar helps the users to navigate throughout the project and open files for editing.
- It gives a more compressed view of the structure visible in the Project window.
- For example in the above diagram when we click on the MainActivity.java file in the editor window section then the navigation bar shows us where this file present inside the android studio. (Here app > src > main > java > com > example > meetandroidstudio > MainActivity).
- Thus it helps us to navigate and modify the required file easily.

3. Editor Window

- In this window, the users can create, write, and modify their code.
- This editor window changes depending on the current file type. For example, if you are viewing a layout file (Here activity_main.xml file) then the editor displays the layout editor.
- Similarly, if you are writing the backend code then the editor displays the Java/Kotlin file (Here MainActivity.java file) depending upon the language you have chosen during the creation of the project.

4. Tool Window Bar

- The tool window bar operates around the outside of the IDE window and includes the buttons that allow users to expand or collapse individual tool windows. For example, in the above diagram, we have expanded the Project and the Build button.

5. Tool Windows

- The tool windows give access to specific tasks like project management, search, version control, and more.
- Access specific tasks like project management, search, version control, and more. You can expand them and collapse them.
- It is strongly related to the Tool Window Bar section. When you expand the Tool Window Bar then they are visible inside the Tool Windows section. So the user can also expand and collapse them.

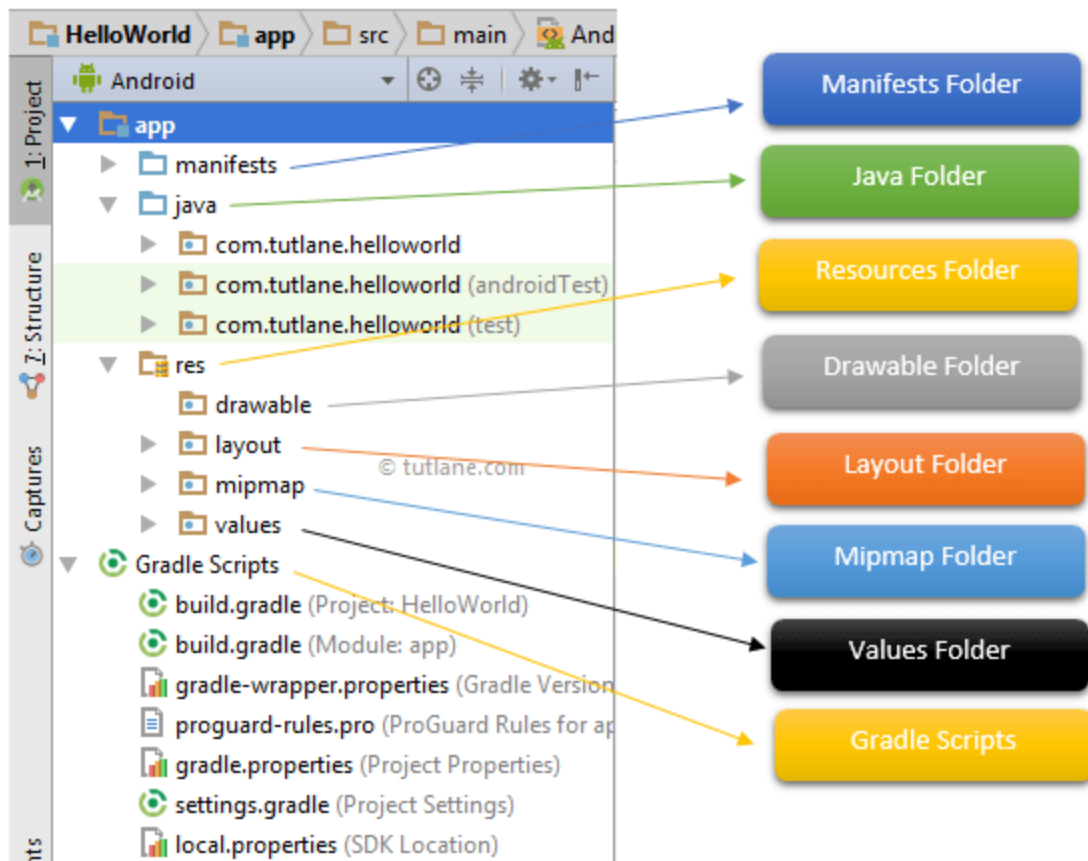
6. Status Bar

- The status bar displays the current status of the project and the IDE itself, as well as any warnings or messages during the execution of the project.

Android Directory Structure

To implement android apps, **Android Studio** is the official IDE (Integrated Development Environment) which is freely provided by Google for android app development.

Once we setup android development environment (/tutorial/android/android-development-environment-setup) using android studio and if we create a sample application using android studio, our project folder structure will be like as shown below. In case if you are not aware of creating an application using an android studio please check this Android Hello World App (/tutorial/android/android-hello-world-app-example).



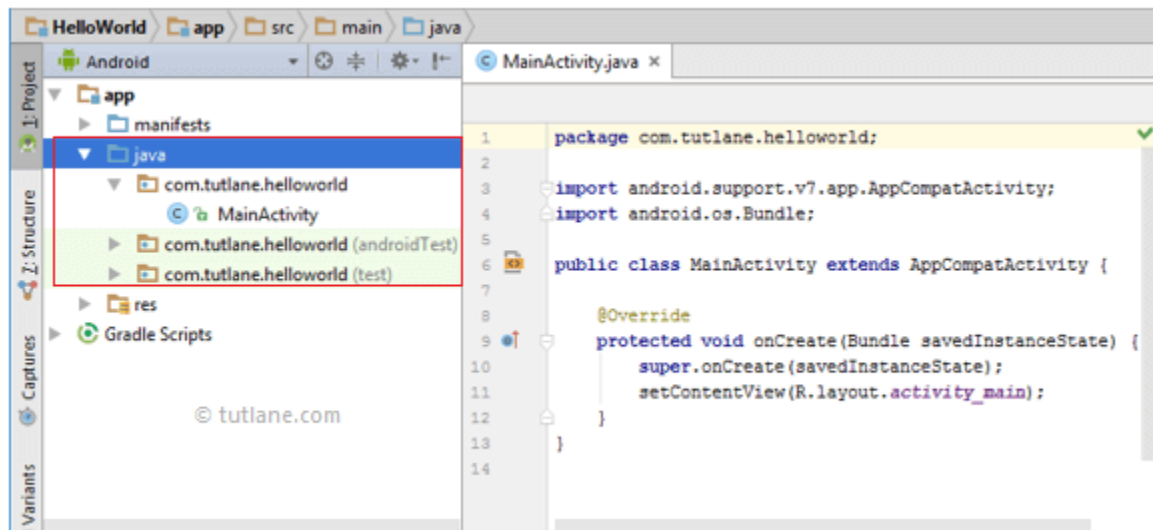
The Android project structure on the disk might differ from the above representation. To see the actual file structure of the project, select **Project** from the **Project** dropdown instead of **Android**.

The android app project will contain different types of app modules, source code files, and resource files. We will explore all the folders and files in the android app.

Java Folder

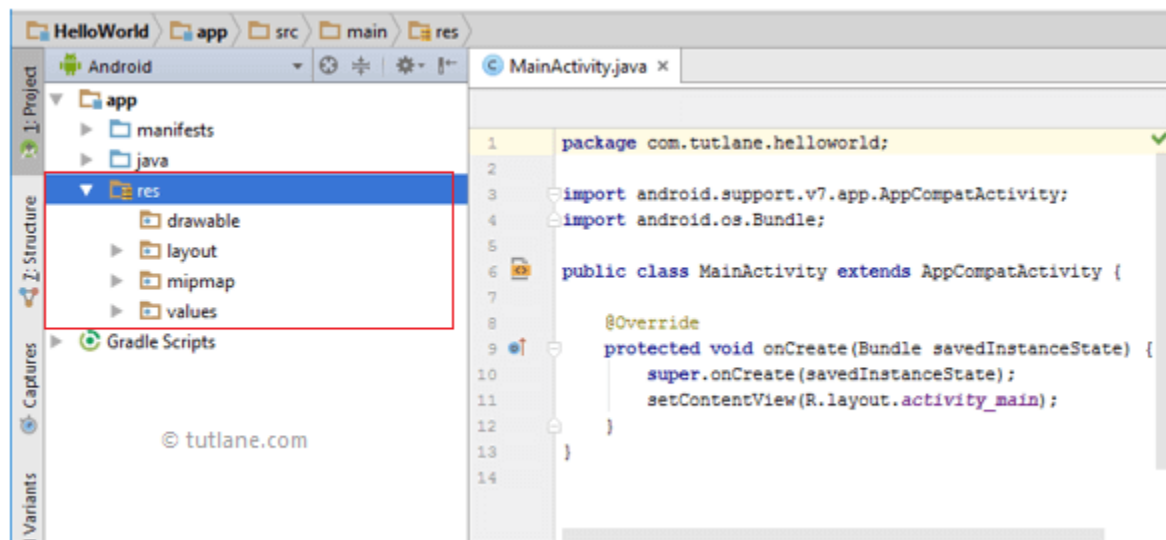
This folder will contain all the java source code (**.java**) files which we'll create during the application development, including JUnit test code. Whenever we create any new project/application, by default the class file **MainActivity.java** will be created automatically under the package name "**com.tutlane.helloworld**".

like as shown below.



res (Resources) Folder

It's an important folder that will contain all non-code resources, such as bitmap images, UI strings, XML layouts like as shown below.



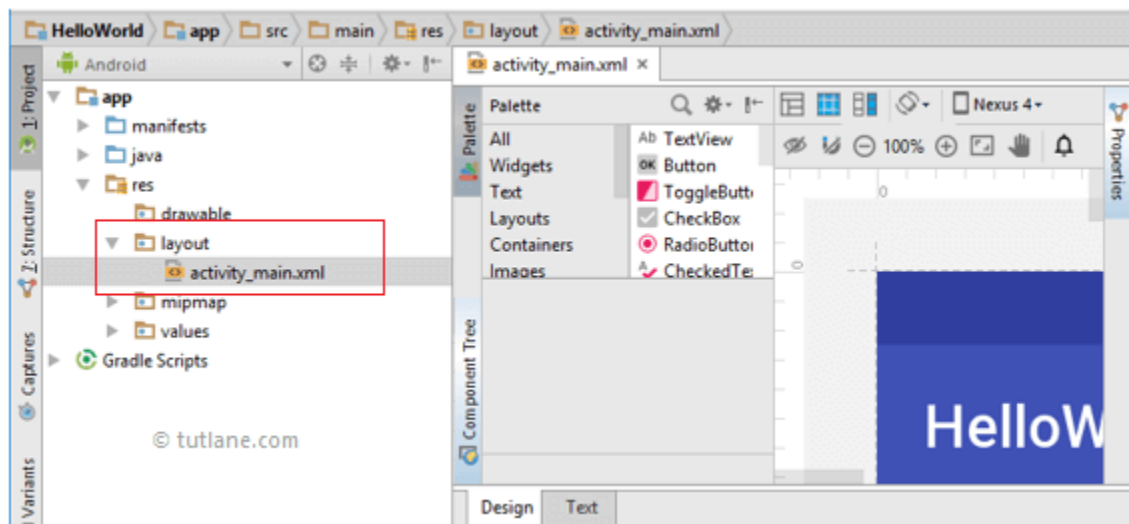
The res (**Resources**) will contain a different type of folders that are

Drawable Folder (res/drawable)

It will contain the different types of images as per the requirement of application. It's a best practice to add all the images in a **drawable** folder other than app/launcher icons for the application development.

Layout Folder (res/layout)

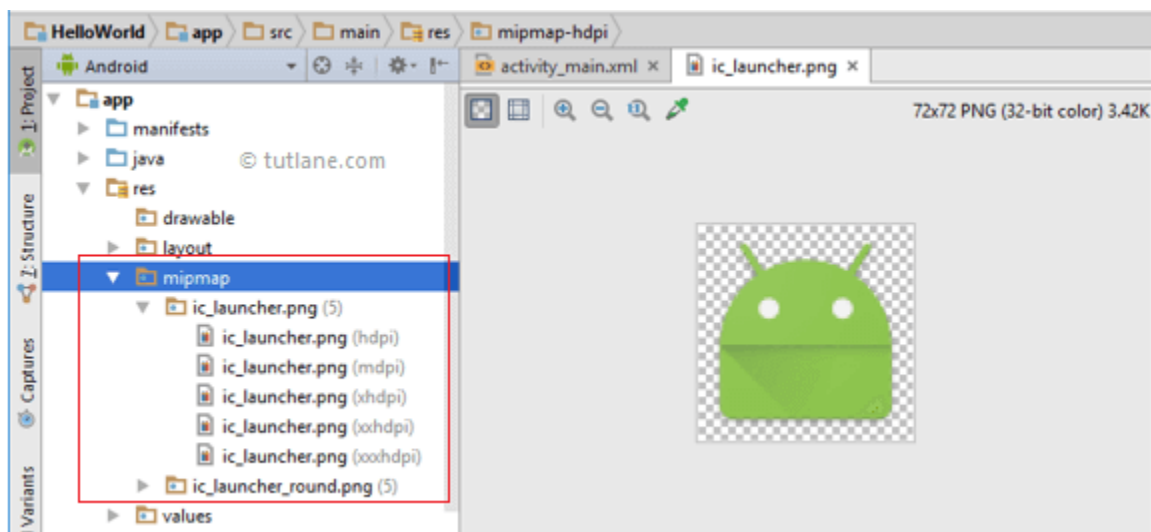
This folder will contain all XML layout files which we used to define the user interface of our application. Following is the structure of the **layout** folder in the android application.



Mipmap Folder (res/mipmap)

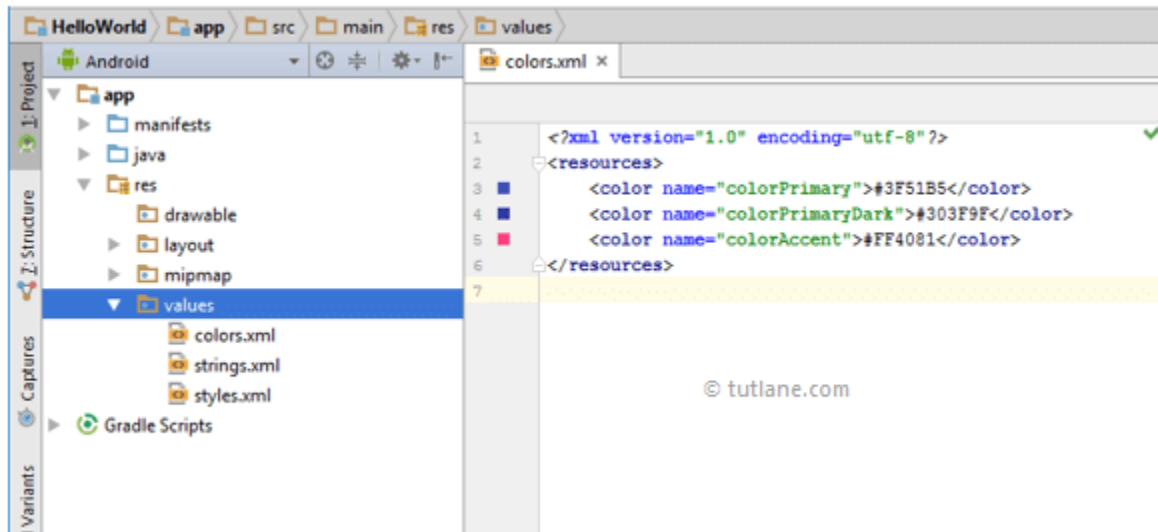
This folder will contain app / launcher icons that are used to show on the home screen. It will contain different density type of icons such as hdpi, mdpi, xhdpi, xxhdpi, xxxhdpi, to use different icons based on the size of the device.

Following is the structure of the **mipmap** folder in the android application.



Values Folder (res/values)

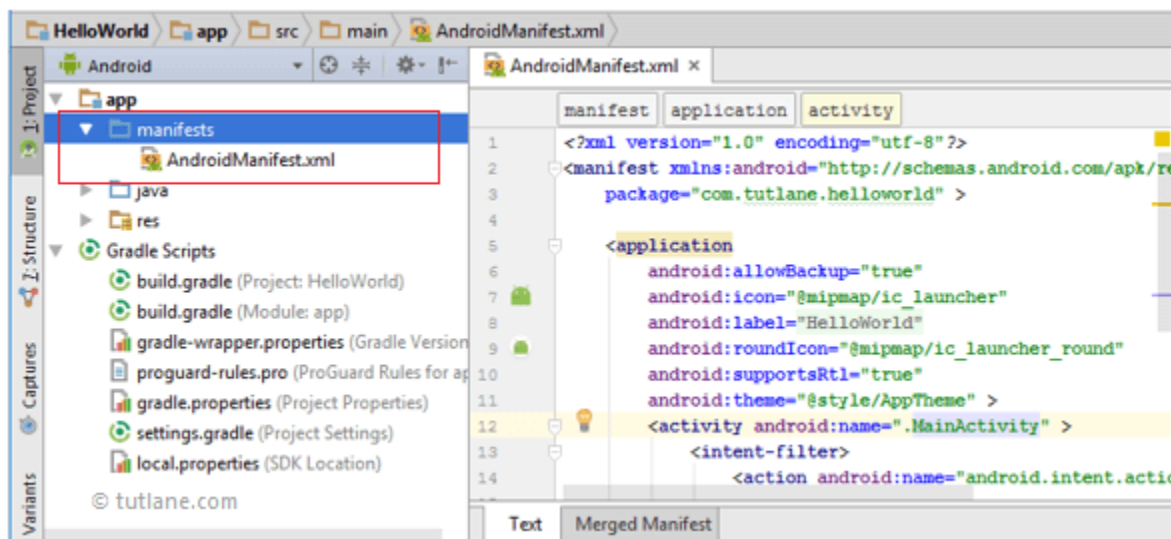
This folder will contain various XML files, such as strings, colors, style definitions and a static array of strings or integers. Following is the structure of the **values** folder in android application.



Manifests Folder

This folder will contain a manifest file (**AndroidManifest.xml**) for our android application. This manifest file will contain information about our application such as android version, access permissions, metadata, etc. of our application and its components. The manifest file will act as an intermediate between android OS and our application.

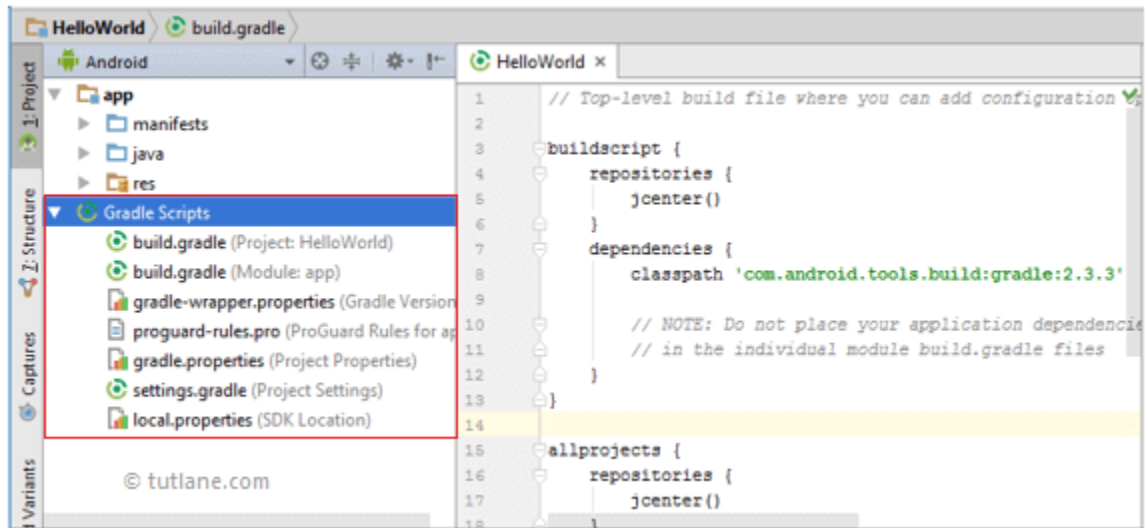
Following is the structure of the **manifests** folder in the android application.



Gradle Scripts

In android, Gradle means automated build system and by using this we can define a build configuration that applies to all modules in our application. In Gradle **build.gradle (Project)**, and **build.gradle (Module)** files are useful to build configurations that apply to all our app modules or specific to one app module.

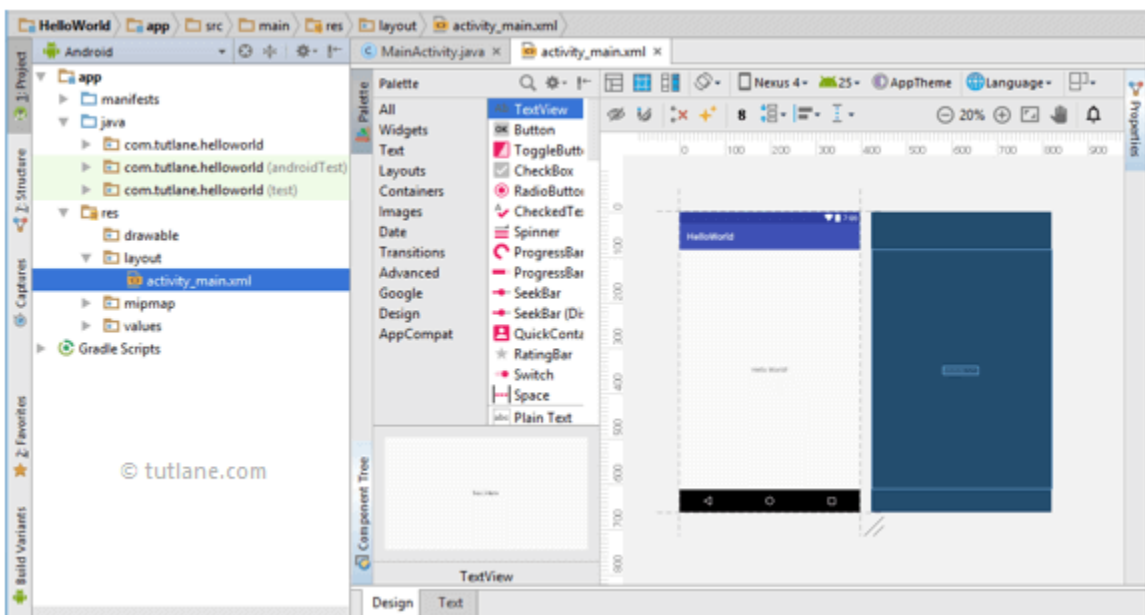
Following is the structure of **Gradle Scripts** in the android application.



Following are the important files which we need to implement an app in android studio.

Android Layout File (activity_main.xml)

The UI of our application will be designed in this file and it will contain **Design** and **Text** modes. It will exist in the **layouts** folder and the structure of **activity_main.xml** file in **Design** mode like as shown below.



We can make required design modifications in **activity_main.xml** file either using **Design** or **Text** modes. If we switch to **Text** mode **activity_main.xml** file will contain a code like as shown below.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout 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:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.tutlane.helloworld.MainActivity">
    <TextView
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
```

```
</android.support.constraint.ConstraintLayout>
```

Android Main Activity File (MainActivity.java)

The main activity file in the android application is **MainActivity.java** and it will exist in the **java** folder. The **MainActivity.java** file will contain the java code to handle all the activities related to our app.

Following is the default code of **MainActivity.java** file which is generated by our HelloWorld application (/tutorial/android/android-hello-world-app-example).

```
package com.tutlane.helloworld;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Android Manifest File (AndroidManifest.xml)

Generally, our application will contain multiple activities (/tutorial/android/android-activity-lifecycle) and we need to define all those activities (/tutorial/android/android-activity-lifecycle) in the **AndroidManifest.xml** file. In our manifest file, we need to mention the main activity for our app using the **MAIN** action and **LAUNCHER** category attributes in **intent filters** (<intent-filter>). In case if we didn't mention MAIN action or LAUNCHER category for the main activity, our app icon will not appear in the home screen's list of apps.

Following is the default code of **AndroidManifest.xml** file which is generated by our **HelloWorld** application.

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

    <application
        android:allowBackup="true"
```

```
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme" >
    <activity android:name=".MainActivity" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>
</manifest>
```

These are the main folders and files required to implement an application in android studio. If you want to see the actual file structure of the project, select **Project** from the **Project** dropdown instead of **Android**.



Android Manifest File in Android



dewangNautiyal



Read

Discuss

Every project in Android includes a Manifest XML file, which is **AndroidManifest.xml**, located in the root directory of its project hierarchy. The manifest file is an important part of our app because it defines the structure and metadata of our application, its components, and its requirements. This file includes nodes for each of the [Activities](#), [Services](#), [Content Providers](#), and [Broadcast Receivers](#) that make the application, and using [Intent Filters](#) and Permissions determines how they coordinate with each other and other applications.

The manifest file also specifies the application metadata, which includes its icon, version number, themes, etc., and additional top-level nodes can specify any required permissions, and unit tests, and define hardware, screen, or platform requirements. The manifest comprises a root manifest tag with a package attribute set to the project's package. It should also include an `xmlns:android` attribute that will supply several system attributes used within the file. We use the `versionCode` attribute is used to define the current application version in the form of an integer that increments itself with the iteration of the version due to update. Also, the `versionName` attribute is used to specify a public version that will be displayed to the users.

We can also specify whether our app should install on an SD card of the internal memory using the `installLocation` attribute. A typical manifest file looks as:

XML

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

```

package="com.example.geeksforgeeks"
android:versionCode="1"
android:versionName="1.0"
android:installLocation="preferExternal">

<uses-sdk
    android:minSdkVersion="18"
    android:targetSdkVersion="27" />

<application
    android:allowBackup="true"
    android:dataExtractionRules="@xml/data_extraction_rules"
    android:fullBackupContent="@xml/backup_rules"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/Theme.MyApplication"
    tools:targetApi="31">
    <activity
        android:name=".MainActivity"
        android:exported="true">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>

</manifest>

```

A manifest file includes the nodes that define the application components, security settings, test classes, and requirements that make up the application. Some of the manifest sub-node tags that are mainly used are:

AD



Data Structures and Algorithms - Self Paced

1. manifest

The main component of the AndroidManifest.xml file is known as manifest. Additionally, the packaging field describes the activity class's package name. It must contain an <application> element with the xmlns:android and package attribute specified.

XML

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.example.geeksforgeeks">

    <!-- manifest nodes -->

    <application>

    </application>

</manifest>
```

2. uses-sdk

It is used to define a minimum and maximum SDK version by means of an API Level integer that must be available on a device so that our application functions properly, and the target SDK for which it has been designed using a combination of minSdkVersion, maxSdkVersion, and targetSdkVersion attributes, respectively. It is contained within the <manifest> element.

XML

```
<uses-sdk
    android:minSdkVersion="18"
    android:targetSdkVersion="27" />
```

3. uses-permission

It outlines a system permission that must be granted by the user for the app to function properly and is contained within the <manifest> element. When an application is installed (on Android 5.1 and lower devices or Android 6.0 and higher), the user must grant the application permissions.

XML

```
<uses-permission
    android:name="android.permission.CAMERA"
    android:maxSdkVersion="18" />
```

4. application

A manifest can contain only one application node. It uses attributes to specify the metadata for your application (including its title, icon, and theme). During development, we should include a debuggable attribute set to true to enable debugging, then be sure to disable it for your release builds. The application node also acts as a container for the Activity, Service, Content Provider, and Broadcast Receiver nodes that specify the application components. The name of our custom application class can be specified using the android:name attribute.

XML

```
<application
    android:name=".GeeksForGeeks"
    android:allowBackup="true"
    android:dataExtractionRules="@xml/data_extraction_rules"
    android:fullBackupContent="@xml/backup_rules"
    android:icon="@drawable/gfgIcon"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportRtl="true"
    android:theme="@android:style/Theme.Light"
    android:debuggable="true"
    tools:targetApi="31">

    <!-- application nodes -->

</application>
```

5. uses-library

It defines a shared library against which the application must be linked. This element instructs the system to add the library's code to the package's class loader. It is contained within the <application> element.

XML

```
<uses-library
    android:name="android.test.runner"
    android:required="true" />
```

6. activity

The Activity sub-element of an application refers to an activity that needs to be specified in the AndroidManifest.xml file. It has various characteristics, like label, name, theme, launchMode, and others. In the manifest file, all elements must be represented by <activity>. Any activity that is not declared there won't run and won't be visible to the system. It is contained within the <application> element.

XML

```
<activity
    android:name=".MainActivity"
    android:exported="true">
</activity>
```

7. intent-filter

It is the sub-element of activity that specifies the type of intent to which the activity, service, or broadcast receiver can send a response. It allows the component to receive intents of a certain type while filtering out those that are not useful for the component. The intent filter must contain at least one <action> element.

XML

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

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

8. action

It adds an action for the intent-filter. It is contained within the <intent-filter> element.

XML

```
<action android:name="android.intent.action.MAIN" />
```

9. category

It adds a category name to an intent-filter. It is contained within the <intent-filter> element.

XML

```
<category android:name="android.intent.category.LAUNCHER" />
```

10. uses-configuration

The uses-configuration components are used to specify the combination of input mechanisms that are supported by our application. It is useful for games that require particular input controls.

XML

```
<uses-configuration
    android:reqTouchScreen="finger"
    android:reqNavigation="trackball"
    android:reqHardKeyboard="true"
    android:reqKeyboardType="qwerty"/>

<uses-configuration
    android:reqTouchScreen="finger"
    android:reqNavigation="trackball"
    android:reqHardKeyboard="true"
    android:reqKeyboardType="twelvekey"/>
```

11. uses-features

It is used to specify which hardware features your application requires. This will prevent our application from being installed on a device that does not include a required piece of hardware such as NFC hardware, as follows:

XML

```
<uses-feature android:name="android.hardware.nfc" />
```

12. permission

It is used to create permissions to restrict access to shared application components. We can also use the existing platform permissions for this purpose or define your own permissions in the manifest.

XML


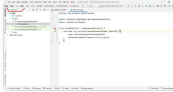
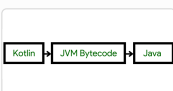
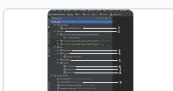
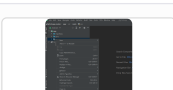



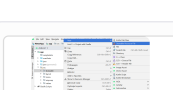
```
<permission
    android:name="com.paad.DETONATE_DEVICE"
    android:protectionLevel="dangerous"
    android:label="Self Destruct"
    android:description="@string/detonate_description">
</permission>
```

Last Updated : 07 Mar, 2023

👍 19



Similar Reads

	Manifest Merger Fails with Multiple Errors in Android Studio		How to Add Manifest Permission to an Android Application?
	How to Convert a Kotlin Source File to a Java Source File in Android?		Android Android Application File Structure
	How to Add Resource File in Existing Android Project in Android Studio?		How to Fetch Audio file From Storage in Android?
	Insert and Fetch Text from a Text File in Android		Extract Data From PDF File using Android Jetpack Compose
Similar read thumbnail	How to Convert SVG, PSD Images to Vector Drawable File in Android Studio?		How to Create Drawable Resource XML File in Android Studio?

1.9 Android SDK vs Android NDK

The Android SDK (Software Development Kit) and NDK (Native Development Kit) are two distinct components of the Android development ecosystem, each serving a specific purpose. Here's a differentiation between the two:

1. Android SDK (Software Development Kit)

- **Purpose:**
 - The Android SDK provides a comprehensive set of tools, libraries, and resources to develop Android applications using Java or Kotlin programming languages.
 - It's primarily used for creating the user interface, managing app logic, accessing device features, and interacting with the Android platform.
- **Key Components:**
 - **Build tools:** Compile code, package resources, and generate APK files.
 - **Emulator:** Test apps on virtual Android devices with different configurations.
 - **Android platforms:** Provide APIs to access various system functionalities.
 - **Support libraries:** Offer backward-compatible versions of features.
 - **Android Debug Bridge (ADB):** Facilitate communication with devices for app installation, debugging, and more.
 - **Android Studio IDE:** A development environment that integrates various SDK components.
- **Primary Use Case:**
 - Android SDK is used for developing apps that primarily rely on Java or Kotlin code and interact with the Android framework's features and resources.

2. Android NDK (Native Development Kit)

- **Purpose:**
 - The Android NDK is a toolset that allows developers to use C and C++ languages to write performance-critical parts of their apps.
 - It's used for situations where native code is preferred due to factors like performance optimization or existing codebases written in C/C++.
- **Key Components:**
 - **Native libraries:** Develop and include native code libraries alongside Java/Kotlin code.
 - **JNI (Java Native Interface):** Provides a bridge for communication between Java/Kotlin code and native code.
- **Primary Use Case:**
 - Android NDK is used for scenarios where apps require low-level access, computational efficiency, or integration with existing C/C++ libraries.
 - Examples include graphics-intensive games, audio processing, and some system-level components.

3. Comparison Summary:

- The Android SDK is used for creating complete Android applications using Java or Kotlin, focusing on user interface, app logic, and system integration.
- The Android NDK is used when performance optimization or existing C/C++ codebases are essential, allowing developers to write native code alongside Java/Kotlin components.

In summary, while the Android SDK is the primary tool for most Android app development, the Android NDK provides a way to integrate native code into apps where performance or existing code considerations are significant.

1.10 Java vs Kotlin

Here are some key differences between Java and Kotlin for Android app development:

- **Syntax** - Java has a verbose syntax with lots of boilerplate code while Kotlin has a more concise and readable syntax.
- **Null Safety** - Kotlin handles null references in a safer way compared to Java. It has nullable and non-nullable types built-in.
- **Interoperability** - Kotlin is interoperable with Java code. Kotlin classes can extend Java classes.
- **Functional Programming** - Kotlin has support for functional programming with lambdas, higher order functions etc. Java has limited functional capabilities.
- **Immutability** - Kotlin has better support for immutable programming with read-only properties. Java requires more boilerplate code to achieve immutability.
- **Concurrency** - Kotlin has better concurrency constructs like coroutines. Java relies on threads and synchronize.
- **Reflection** - Kotlin has better reflection support for things like annotations, lambdas etc. Java reflection is more limited.
- **Type inference** - Kotlin can infer types in many cases while Java requires explicit types.
- **Boot time** - Kotlin has slightly slower boot time compared to Java due to runtime overhead.
- **Community** - Java has an older and more established community while Kotlin adoption is growing rapidly.

1.11 Sample Questions

Marks	Question
2	Explain briefly how Android came to be and how it has changed over time.
2	Compare and contrast Android's functionality with that of Apple's iOS.
2	Explain the benefits and strengths of using Android as a mobile operating system.
3	Find out more about the Open Handset Alliance and how they've helped shape the Android ecosystem.
3	Analyse how Android's open-source nature has affected its market share.
3	Make a diagram explaining how the Android OS is structured, and provide some examples of its components.
2	Create a well-reasoned case for or against Android's status as an open-source OS. Explain why you think the way you do.
2	Investigate whether or not a specific programming language is required for Android applications. Provide an explanation to back up your answer.
6	Investigate the role that Android Dev Tools play in the bigger picture of app creation.
4	List the main functions and features that Android's SDK provides to programmers.

1.12 Related Videos

- Set up Android Studio on Windows
<https://www.youtube.com/watch?v=PmiSj2QGZVM>
- Complete Overview of Android Studio
<https://www.youtube.com/watch?v=-nZONAGIdaI>
- How to choose Java in new project in Flamingo version in Android Studio
<https://www.youtube.com/watch?v=wxCKgjRBHlo>
- First Project
<https://www.youtube.com/watch?v=poDgvBrEZGo>
- Connect Device to Android Studio to test and run your App on Android Smartphone (Android Debug Bridge)
<https://www.youtube.com/watch?v=ZgkoXdI-2-s>
<https://www.youtube.com/watch?v=L6xP0SoKgvM>
- Create Android Virtual Device (AVD) Emulator for Android Studio
<https://www.youtube.com/watch?v=0RMrUK9TRec>
- Run Android App on Virtual Device (AVD)
<https://www.youtube.com/watch?v=L6xP0SoKgvM>