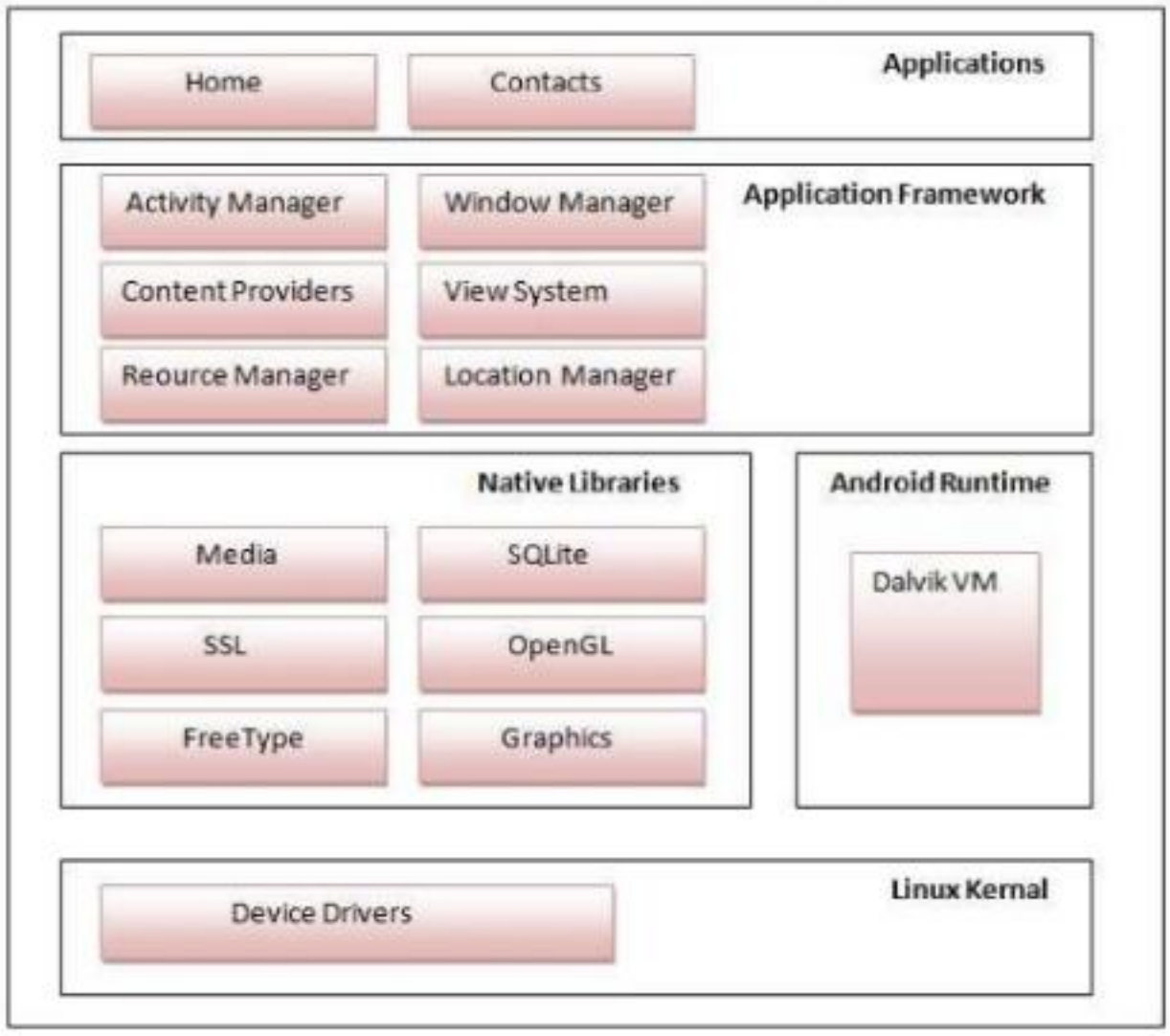
ANDROID SOFTWARE STACK



At the core of the Android platform is a Linux kernel responsible for device drivers, resource access, power management, and other OS duties. The supplied device drivers include Display, Camera, Keypad, Wi-Fi, Flash Memory, Audio, and inter-process communication (IPC). Sitting at the next level, on top of the kernel, are a number of C/C++ libraries such as OpenGL, WebKit, FreeType, Secure Sockets Layer (SSL), the C runtime library (libc), SQLite, and Media. The media libraries are based on PacketVideo's OpenCORE. These libraries are responsible for recording and playback of audio and video formats. A library called Surface Manager Controls access to the display system and supports 2D and 3D.

The WebKit library is responsible for browser support; it is the same library that supports Google Chrome and Apple's Safari. The FreeType library is responsible for font support. SQLite is a relational database that is available on the device itself. SQLite is also an independent open source effort for relational databases and not directly tied to Android. Most of the application framework accesses these core libraries through the Dalvik VM, the gateway to the Android platform. As we indicated in the previous sections, Dalvik is optimized to run multiple instances of VMs. As Java applications access these core libraries, each application gets its own VM instance.

The Android Java API's main libraries include telephony, resources, locations, UI, content providers (data), and package managers (installation, security, and so on). From a media perspective, the Android platform supports the most common formats for audio, video, and images. From a wireless perspective, Android has APIs to support Bluetooth, EDGE, 3G, Wi-Fi, and Global System for Mobile Communication (GSM) telephony, depending on the hardware.

called the application's *manifest file* (AndroidManifest.xml).

ANDROID JAVA PACKAGES

One way to get a quick snapshot of the Android platform is to look at the structure of Java packages. Because Android deviates from the standard JDK distribution, it is important to know what is supported and what is not. Here's a brief description of the important packages that are included in the Android SDK:

android.app: Implements the Application model for Android. Primary classes include Application, representing the start and stop semantics, as well as a number of activity-related classes, fragments, controls, dialogs, alerts, and notifications. We work with most of these classes through out this book.
android.app.admin: Provides the ability to control the device by folks such as enterprise administrators.
android.accounts: Provides classes to manage accounts such as Google, Facebook, and so on. The primary classes are AccountManager and Account.

android.animation: Hosts all the new property animation classes.
android.app.backup: Provides hooks for applications to back up and restore their data when folks switch their devices.
android.appwidget: Provides functionality for home screen widgets.
android.bluetooth: Provides a number of classes to work with Bluetooth functionality. The main classes include BluetoothAdapter, BluetoothDevice, BluetoothSocket, BluetoothServerSocket, and BluetoothClass. We can use BluetoothAdapter to control the locally installed Bluetooth adapter.
android.content: Implements the concepts of content providers. Content providers abstract out data access from data stores.
android.content.res: Provides access to resource files, both structured and unstructured. The primary classes are AssetManager (for unstructured resources) and Resources.
android.database: Implements the idea of an abstract database. The primary interface is the Cursor interface.
android.database.sqlite: Implements the concepts from the android.database package using SQLite as the physical database. Primary classes are SQLiteCursor, SQLiteDatabase, SQLiteQuery, SQLiteQueryBuilder, and SQLiteStatement.
android.drm: Classes related to Digital Rights Management.
android.graphics: Contains the classes Bitmap, Canvas, Camera, Color, Matrix, Movie, Paint, Path, Rasterizer, Shader, SweepGradient, and TypeFace.
android.graphics.drawable: Implements drawing protocols and background images, and allows animation of drawable objects.
android.graphics.drawable.shapes: Implements shapes including ArcShape, OvalShape, PathShape, RectShape, and RoundRectShape.
android.hardware: Implements the physical Camera-related classes. The Camera represents the hardware camera, whereas
android.graphics.Camera: Rrepresents a graphical concept that's not related to a physical camera at all.
android.hardware.usb: Let us talk to USB devices from Android.
android.location: Contains the classes Address, GeoCoder, Location, LocationManager, and LocationProvider.
android.media: Contains the classes MediaPlayer, MediaRecorder, Ringtone, AudioManager, and FaceDetector. MediaPlayer, which supports streaming, is used to play audio and video. MediaRecorder is used to record audio and video. The Ringtone class is used to play short sound snippets that could serve as ringtones and notifications.

android.hardware: Implements the physical Camera-related classes. The Camera represents the hardware camera, whereas
android.graphics.Camera: Rrepresents a graphical concept that's not related to a physical camera at all.
android.hardware.usb: Let us talk to USB devices from Android.
android.location: Contains the classes Address, GeoCoder, Location, LocationManager, and LocationProvider.
android.media: Contains the classes MediaPlayer, MediaRecorder, Ringtone, AudioManager, and FaceDetector. MediaPlayer, which supports streaming, is used to play audio and video. MediaRecorder is used to record audio and video. The Ringtone class is used to play short sound snippets that could serve as ringtones and notifications.
android.media.audiofx: Provides audio effects.
android.media.effect: Provides video effects.
android.mtp: Provides the ability to interact with cameras and music devices.
android.net: Implements the basic socket-level network APIs. Primary classes include Uri, ConnectivityManager, LocalSocket, and LocalServerSocket. It is also

worth noting here that Android supports HTTPS at the browser level and also at the network level. Android also supports JavaScript in its browser.
android.net.rtp: Supports streaming protocols.
android.net.sip: Provides support for VOIP.
android.net.wifi: Manages Wi-Fi connectivity. Primary classes include WifiManager and WifiConfiguration. WifiManager is responsible for listing the configured networks and the currently active Wi-Fi network.
android.net.wifi.p2p: Supports P2P networks with Wi-Fi Direct.
android.telephony: Contains the classes CellLocation, PhoneNumberUtils, and TelephonyManager. TelephonyManager lets we determine cell location, phone number, network operator name, network type, phone type, and Subscriber Identity Module (SIM) serial number.
android.telephony.gsm: Allows we to gather cell location based on cell towers and also hosts classes responsible for SMS messaging. This package is called GSM because Global System for Mobile Communication is the technology that originally defined the SMS data messaging standard.
android.telephony.cdma: Provides support for CDMA telephony. *android.test, android.test.mock, android.test.suitebuilder:* Packages to support writing unit tests for Android applications.
android.text: Contains text-processing classes.
android.text.style: Provides a number of styling mechanisms for a span of text.
android.utils: Contains the classes Log, DebugUtils, TimeUtils, and Xml.
android.view: Contains the classes Menu, View, and ViewGroup, and a series of listeners and callbacks.
android.view.animation: Provides support for tweening animation. The main classes include Animation, a series of interpolators for animation, and a set of specific animator classes that include AlphaAnimation, ScaleAnimation, TranslationAnimation, and RotationAnimation.Some of the classes.
android.webkit: Contains classes representing the web browser. The primary classes include WebView, CacheManager, and CookieManager. *android.widget:* Contains all of the UI controls usually derived from the View class. Primary widgets include Button, Checkbox, Chronometer, AnalogClock, DatePicker, DigitalClock, EditText, ListView, FrameLayout, GridView, ImageButton, MediaController, ProgressBar, RadioButton, RadioGroup, RatingButton, Scroller, ScrollView, Spinner, TabWidget, TextView, TimePicker, VideoView, and ZoomButton.
com.google.android.maps: Contains the classes MapView, MapController, and MapActivity, essentially classes required to work with Google maps.

Mobile Communication (GSM) telephony, depending on the hardware.

DELVING INTO DALVIK VM

Google optimizing designs for low-powered handheld devices The key figure in Google's implementation of this JVM is Dan Bornstein, who wrote the Dalvik VM—Dalvik is the name of a town in Iceland. Dalvik VM takes the generated Java class files and combines them into one or more Dalvik Executable (.dex) files. The goal of the Dalvik VM is to find every possible way to optimize the JVM for space, performance, and battery life. The final executable code in Android, as a result of the Dalvik VM, is based not on Java byte code but on .dex files instead. This means we cannot directly execute Java byte code; we have to start with Java class files and then convert them to linkable .dex files.

DEVELOPING END USER APPLICATION USING ANDROID SDK

packaging and indexing occur only when truly needed.

FUNDAMENTAL COMPONENTS

When we build applications for Android we need to understand JavaServer Pages (JSP) and servlets in order to write Java 2 Platform, Enterprise Edition (J2EE) applications. Similarly, we need to understand views, activities, fragments, intents, content providers, services, and the AndroidManifest.xml file.

1. View

Views are user interface (UI) elements that form the basic building blocks of a user interface. A view can be a button, a label, a text field, or many other UI elements. Views are also used as containers for views, which mean there's usually a hierarchy of views in the UI.

2. Activity

An *activity* is a UI concept that usually represents a single screen in our application. It generally contains one or more views, but it doesn't have to. An activity is pretty much like it sounds—something that helps the user do one thing, which could be viewing data, creating data, or editing data. Most Android applications have several activities within them.

3. Fragment

When a screen is large, it becomes difficult to manage all of its functionality in a single activity. *Fragments* are like sub-activities, and an activity can display one or more fragments on the screen at the same time. When a screen is small, an activity is more likely to contain just one fragment, and that fragment can be the same one used within larger screens.

4. Intent

An *intent* generically defines an "intention" to do some work. Intents encapsulate several concepts, so the best approach to understanding them is to see examples of their use. We can use intents to perform the following tasks:

Broadcast a message.
Start a service.
Launch an activity.
Display a web page or a list of contacts.
Dial a phone number or answer a phone call.

Intents are not always initiated by our application—they're also used by the system to notify our application of specific events (such as the arrival of a text message). Intents can be explicit or implicit. If we simply say that we want to display a URL, the system decides what component will fulfill the intention. We can also provide specific information about what should handle the intention.

5. Content Provider

Data sharing among mobile applications on a device is common. Therefore, Android defines a standard mechanism for applications to share data (such as a list of contacts) without exposing the underlying storage, structure, and implementation. Through content providers, we can expose our data and have our applications use data from other applications.

6. Service

Services in Android resemble services we see in Windows or other platforms—they're background processes that can potentially run for a long time. Android defines two types of services: local services and remote services. Local services are components that are only accessible by the application that is hosting the service. Conversely, remote services are services that are meant to be accessed remotely by other applications running on the device.

7. AndroidManifest.xml

AndroidManifest.xml, which is similar to the web.xml file in the J2EE world, defines the contents and behavior of our application. For example, it lists our application's activities and services, along with the permissions and features the application needs to run.

ANDROID VIRTUAL DEVICES

An Android Virtual Device (AVD) represents a device configuration. It allows developers to test their applications without hooking up an actual Android device (typically a phone or a tablet). AVDs can be created in various configurations to emulate different types of real devices.

into. If not, try editing our Run Configuration to manually select the target.

STRUCTURE OF ANDROID APPLICATION

An Android application is primarily made up of three pieces: the application descriptor, a collection of various resources, and the application's source code. Android applications have some artifacts that are required and some that are optional. The following table summarizes the elements of an Android application.

Artifact	Description	Required?
AndroidManifest.xml	The Android application descriptor file. This file defines the activities, content providers, services, and intent receivers of the application. We can also use this file to declaratively define permissions required by the application, as well as grant specific permissions to other applications using the services of the application. Moreover, the file can contain instrumentation detail that we can use to test the application or another application.	Yes
src	A folder containing all of the source code of the application	Yes
assets	An arbitrary collection of folders and files	No
Res	A folder containing the resources of the application. This is the parent folder of drawable, anim, layout, menu, values, xml, and raw.	Yes
drawable	A folder containing the images or image-descriptor files used by the application.	No
Animator	A folder containing the XML-descriptor files that describe the animations used by the application. On older Android versions, this is called anim.	No
Layout	A folder containing views of the application.	No

	We should create our application's views by using XML descriptors rather than coding them.	
Menu	A folder containing XML-descriptor files for menus in the application.	No
Values	A folder containing other resources used by the application. Examples of resources found in this folder include strings, arrays, styles, and colors.	No
Xml	A folder containing additional XML files used by the application.	No
raw	A folder containing additional data—possibly non-XML data— that is required by the application.	No

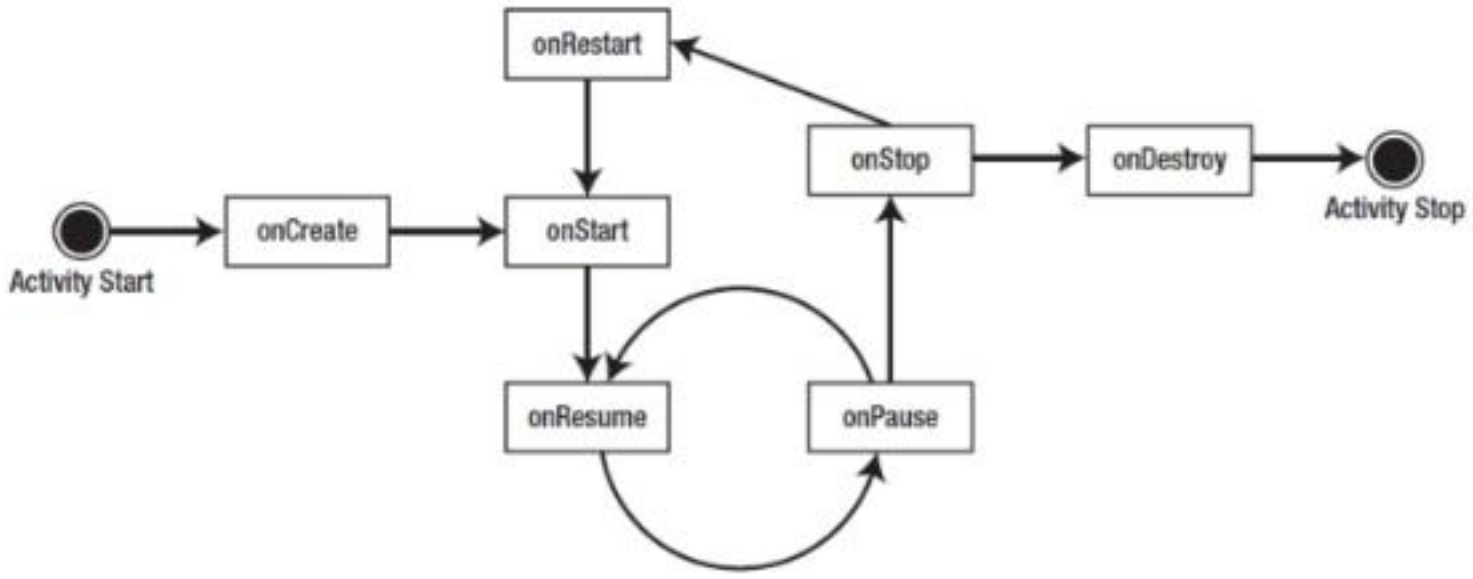
	application.	

APPLICATION LIFE CYCLE.

The life cycle of an Android application is strictly managed by the system, based on the user's needs, available resources, and so on. Android is sensitive to the life cycle of an application and its components.

Life-Cycle Methods of an Activity

protected void onCreate(Bundle savedInstanceState);
protected void onStart();
protected void onRestart();
protected void onResume();
protected void onPause();
protected void onStop();
protected void onDestroy();



The system can start and stop our activities based on what else is happening. Android calls the onCreate() method when the activity is freshly created. onCreate() is always followed by a call to onStart(), but onStart() is not always preceded by a call to onCreate() because onStart() can be called if our application was stopped. When onStart() is called, our activity is not visible to the user, but it's about to be. onResume() is called after onStart(), just when the activity is in the foreground and accessible to the user. At this point, the user can interact with our activity. When the user decides to move to another activity, the system calls our activity's onPause() method. From onPause(), we can expect either onResume() or onStop() to be called. onResume() is called, for example, if the user brings our activity back to the foreground. onStop() is called if our activity becomes invisible to the user. If our activity is brought back to the foreground after a call to onStop(), then onRestart() is called. If our activity sits on the activity stack but is not visible to the user, and the system decides to kill our activity, onDestroy() is called.