Topic 1: Android Architecture:

Android is architected in the form of a software stack comprising applications, an operating system, run-time environment, middleware, services and libraries. This architecture can, perhaps, best be represented visually as outlined in Figure 6-1. Each layer of the stack, and the corresponding elements within each layer, are tightly integrated and carefully tuned to provide the optimal application development and execution environment for mobile devices.

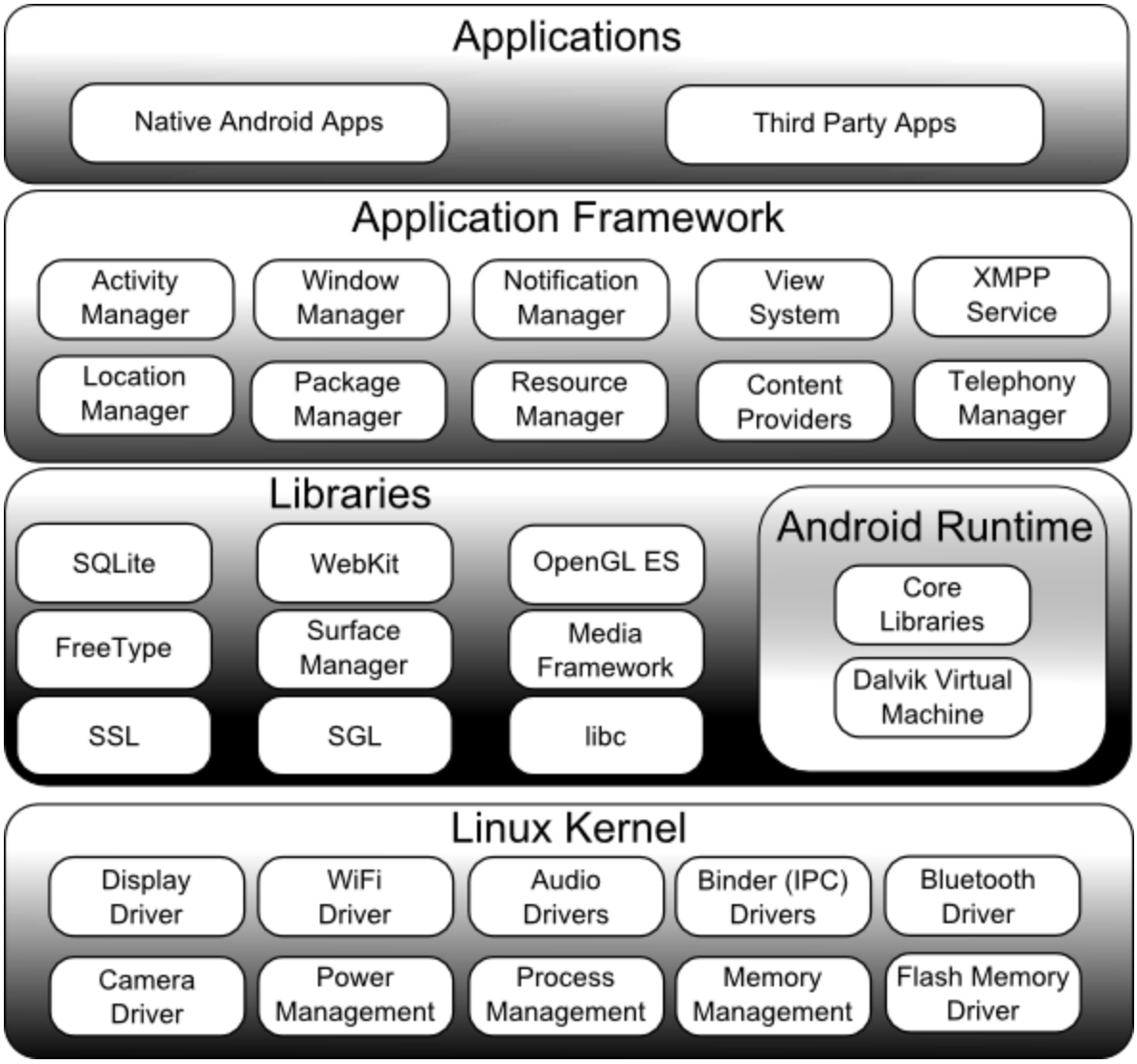

Figure 6-1

The remainder of this chapter will work through the different layers of the Android stack, starting at the bottom with the Linux Kernel.

The Linux Kernel

Positioned at the bottom of the Android software stack, the Linux Kernel provides a level of abstraction between the device hardware and the upper layers of the Android software stack. Based on Linux version 2.6, the kernel provides preemptive multitasking, low-level core system services such as memory, process and power management in addition to providing a network stack and device drivers for hardware such as the device display, Wi-Fi and audio.

Android Runtime -

As previously noted, the Linux kernel provides a multitasking execution environment allowing multiple processes to execute concurrently. It would be easy to assume, therefore, that each Android application simply runs as a process directly on the Linux kernel. In fact, each application running on an Android device does so within its own instance of the Dalvik virtual machine (VM).

Running applications in virtual machines provides a number of advantages. Firstly, applications are essentially sandboxed, in that they cannot detrimentally interfere (intentionally or otherwise) with the operating system or other applications, nor can they directly access the device hardware. Secondly, this enforced level of abstraction makes applications platform neutral in that they are never tied to any specific hardware.

Android Runtime – Core Libraries

The Android Core Libraries (also referred to as the Dalvik Libraries) fall into three main categories, each of which merits an individual description:

Java Interoperability Libraries

Android applications are predominantly developed using the Java programming language.

Android Libraries

This category encompasses those Java-based libraries that are specific to Android development. Examples of libraries in this category include the application framework libraries in addition to those that facilitate user interface building, graphics drawing and database access.

A summary of few of the key core Android libraries available to the Android developer is as follows:

- **android.app** – Provides access to the application model and is the cornerstone of all Android applications.

- **android.content** – Facilitates content access, publishing and messaging between applications and application components.
- **android.database** – Used to access data published by content providers and includes SQLite database management classes.
- **android.graphics** – A low-level 2D graphics drawing API including colors, points, filters, rectangles and canvases.
- **android.hardware** – Presents an API providing access to hardware such as the accelerometer and light sensor.
- **android.opengl** – A Java interface to the OpenGL ES 3D graphics rendering API.
- **android.os** – Provides applications with access to standard operating system services including messages, system services and inter-process communication.
- **android.media** – Provides classes to enable playback of audio and video.
- **android.net** – A set of APIs providing access to the network stack. Includes android.net.wifi, which provides access to the device's wireless stack.

Having covered the Java-based core libraries in the Android runtime, it is now time to turn our attention to the C/C++ based libraries contained in this layer of the Android software stack.

C/C++ Libraries

It is important to note, however, that the core libraries do not actually perform much of the actual work and are, in fact, essentially Java "wrappers" around a set of C/C++ based libraries. When making calls, for example, to the android.opengl library to draw 3D graphics on the device display, the library actually ultimately makes calls to the OpenGL ES C++ library which, in turn, works with the underlying Linux kernel to perform the drawing tasks.

C/C++ libraries are included to fulfill a wide and diverse range of functions including 2D and 3D graphics drawing, Secure Sockets Layer (SSL) communication, SQLite database management, audio and video playback, bitmap and vector font rendering, display subsystem and graphic layer management and an implementation of the standard C system library (libc).

Application Framework

The Application Framework is a set of services that collectively form the environment in which Android applications run and are managed. This framework implements the concept that Android applications are constructed from reusable, interchangeable and replaceable components. This concept is taken a step further in that an application is also able to publish its capabilities along with any corresponding data so that they can be found and reused by other applications.

The Android framework includes the following key services:

- **Activity Manager** – Controls all aspects of the application lifecycle and activity stack.
- **Content Providers – Allows applications to publish and share data with other applications.**
- **Resource Manager** – Provides access to non-code embedded resources such as strings, color settings and user interface layouts.
- **Notifications Manager** – Allows applications to display alerts and notifications to the user.
- **View System** – An extensible set of views used to create application user interfaces.
- **Package Manager** – The system by which applications are able to find out information about other applications currently installed on the device.
- **Telephony Manager** – Provides information to the application about the telephony services available on the device such as status and subscriber information.
- **Location Manager** – Provides access to the location services allowing an application to receive updates about location changes.

Topic 2: Android Application Development:

Dalvik Virtual Machine:

Android based systems utilize their own virtual machine (VM), which is known as the Dalvik Virtual Machine (DVM). The DVM uses special byte-code, hence native Java bytecode cannot directly be executed on Android systems. The Android community provides a tool (dx) that allows converting Java class files into Dalvik executables (dex). The DVM implementation is highly optimized in order to perform as efficiently and as effectively as possible on mobile devices that are normally equipped with a rather slow (single) CPU, limited memory resources, no OS swap space, and limited battery capacity. The DVM has been implemented in a way that allows a device to execute multiple VM's in a rather efficient manner.

It also has to be pointed out that the DVM relies on the modified Linux kernel for any potential threading and low-level memory management functionalities. With Android 2.2, some major changes to the JVM infrastructure were implemented. Up to version 2.2, the JVM was an actual interpreter, similar to the original JVM solution deployed with Java 1.0. While the Android solution always reflected a very efficient interpreter, it was still an interpreter and hence, no native code was generated. With the release of Android 2.2, a justin-time (JIT) compiler has been incorporated into the solution stack, which translates the Dalvik byte-code into much more efficient machine code (similar to a C compiler). Down the road, additional JIT and

garbage collection (GC) features will be deployed with Android, further busting (potential) aggregate systems performance.

Topic 3: **Android Application structure:**

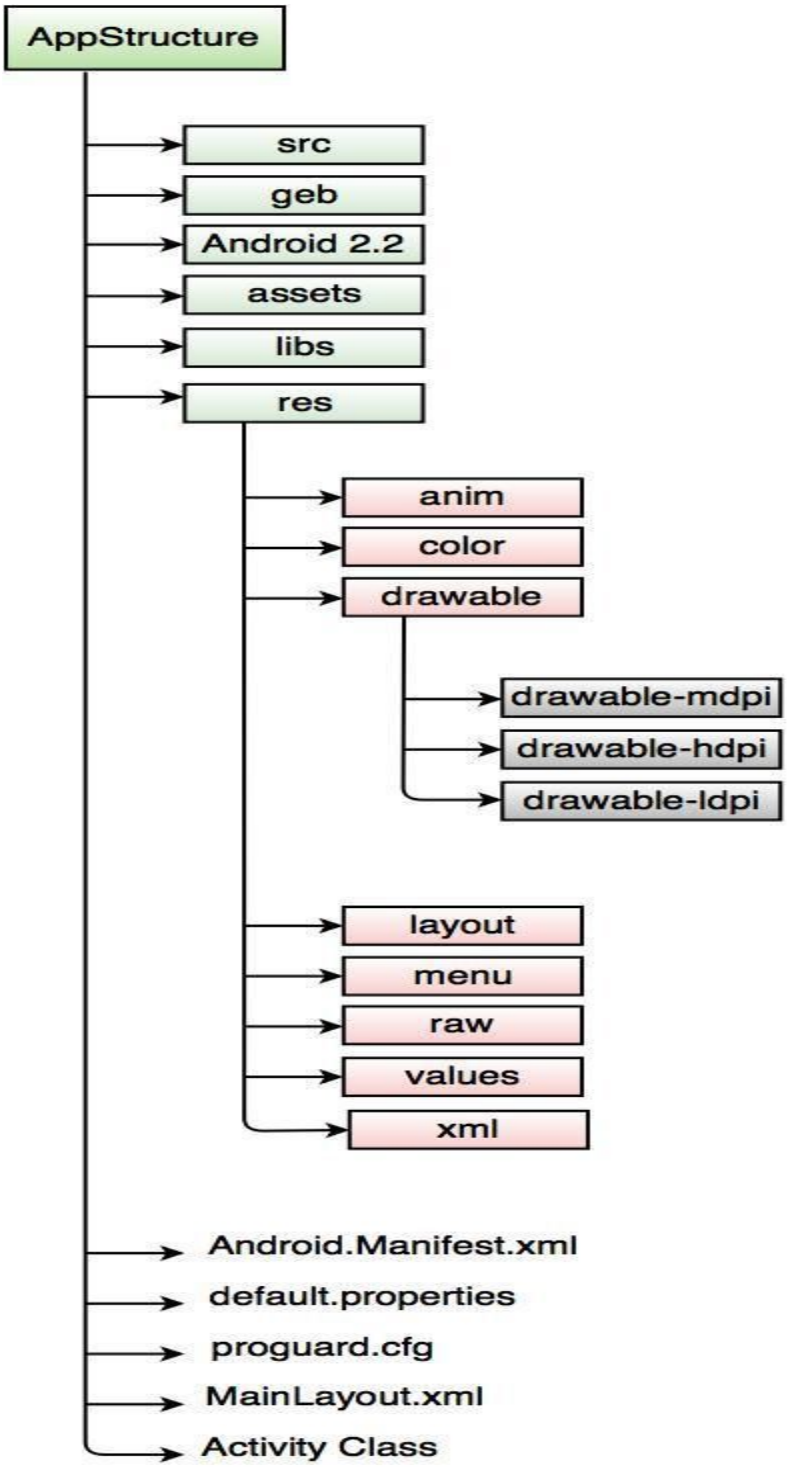

Fig. Structure of Android Application

Every Android project contains several folders, like:

Folder Name	Description
src	The 'src' stands for **Source Code.** It contains the Java Source files.
gen	The 'gen' stands for **Generated Java Library.** This library is for Android internal use only.
Android 2.2	The Android Framework Library is stored here.
assets	It is used to store raw asset files.
libs	It contains private libraries.
res	The 'res' stands for **Resource file.** It can store resource files such as pictures, XML files, etc. It contains some additional folders such as Drawable, Layout and Values. **anim:** It is used for XML files that are compiled into animation objects. **color:** It is used for XML files that describe colors. **drawable:** It is used to store various graphics files. In Android project structure, **there are three types of drawable folders,** 1.drawable-mdpi 2.drawable-hdpi 3.drawable-ldpi The above drawable folders are required in order to adapt to different screen resolutions. **layout:** It is used for placing the XML layout files, which defines how various Android objects such as textbox, buttons, etc. are organized on the screen. **menu:** It is used for defining the XML files in the application menu. **raw:** The 'raw' stands for **Raw Asset Files.** These files are referenced from the application using a resource identifier in the R class. **For example,** good place for media is MP3 or Ogg files. **values:** It is used for XML files which stores various string values, such as

	titles, labels, etc. **xml:** It is used for configuring the application components.
AndroidManifest.xml	This file indicates the Android definition file. This file contains the information about the Android application such as minimum Android version, permission to access Android device capabilities such as Internet access permission, phone permission etc.
default.properties	This file contains the project settings, such as build the target. Do not edit this file manually. It should be maintained in a Source Revision Control System.
Proguard.cfg	This file defines how ProGuard optimizes and makes your code unclear.
MainLayout.xml	This file describes the layout of the page. So all the components such as textboxes, labels, radio buttons, etc. are displaying on the application screen.
Activity class	The application occupies the entire device screen which needs at least one class inherits from the Activity class. OnCreate() method initiates the application and loads the layout page.

Topic 4: Application Process Management:

A process on Android can be in one of five different states at any given time, from most important to least important:

- **1. Foreground process**: The app you're using is considered the foreground process. Other processes can also be considered foreground processes — for example, if they're interacting with the process that's currently in the foreground. There are only a few foreground processes at any given time.
- **2. Visible process**: A visible process isn't in the foreground, but is still affecting what you see on your screen. For example, the foreground process may be a dialog that allows you to see an app behind it — the app visible in the background would be a visible process.
- **3. Service process**: A service process isn't tied to any app that's visible on your screen. However, it's doing something in the background, such as playing music or downloading data in the background. For example, if you start playing music and switch to another app, the music-playing is in the background is being handled by a service process.
- **4. Background process**: Background processes are not currently visible to the user. They have no impact on the experience of using the phone. At any given time, many background processes are currently running. You can think of these background processes as "paused" apps. They're kept in memory so you can quickly resume using them when you go back to them, but they aren't using valuable CPU time or other non-memory resources.
- **5. Empty process**: An empty process doesn't contain any app data anymore. It may be kept around for caching purposes to speed up app launches later, or the system may kill it as necessary.

For example, let's say you turn on your phone and open a music app. While you use it, the music app will be a foreground process. When you start playing music and leave the music app, the music will continue playing as a service process.