Android 3 platform SDK techniques for developing smartphone and tablet apps

Pro Android 3

Satya Komatineni | Dave MacLean | Sayed Hashimi

Apress®

Pro Android 3

Satya Komatineni
Dave MacLean
Sayed Y. Hashimi

Apress®

Pro Android 3

ISBN-13 (pbk): 978-1-4302-3222-3

ISBN-13 (electronic): 978-1-4302-3223-0

President and Publisher: Paul Manning
Lead Editor: Matthew Moodie
Technical Reviewer: Dylan Phillips
Editorial Board: Steve Anglin, Mark Beckner, Ewan Buckingham, Gary Cornell, Jonathan Gennick, Jonathan Hassell, Michelle Lowman, Matthew Moodie, Jeffrey Pepper, Frank Pohlmann, Douglas Pundick, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh
Coordinating Editor: Corbin Collins
Copy Editors: Heather Lang, Tracy Brown, Mary Behr
Compositor: MacPS, LLC
Indexer: BIM Indexing & Proofreading Services
Artist: April Milne
Cover Designer: Anna Ishchenko

Distributed to the book trade worldwide by Springer Science+Business Media, LLC., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at www.apress.com/info/bulksales.

The source code for this book is available to readers at www.apress.com.

To my brother Hari, to whom life yielded few favors.

—Satya Komatineni

To my wife, Rosie, and my son, Mike, for their support; I couldn't have done this without you. And to Max, for spending so much time at my feet keeping me company.

—Dave MacLean

To my son, Sayed-Adieb.

—Sayed Y. Hashimi

Contents at a Glance

Contents

Foreword

All this has happened before, and all this will happen again. Emergence Theory is the way complex systems and patterns arise out of a set of environmental interactions.

And we have been here before.

When I started programming in 1985, there were a variety of personal computers available. While I cut my teeth on an Apple II C, my friends either had Commodore 128s, Tandy CoCo 3s, or Atari computers. Each of us grew within the constraints of our own environment, but we were rarely able to share our work. When affordable IBM clones running Microsoft's DOS began to emerge, developers started to see value of the marketplace created, and rapid evolution began to occur within the DOS ecosystem. Eventually Microsoft created the dominate position in the PC market that it still enjoys today.

In 2003, when I started mobile programming, the ecosystem looked much the same as it did back in 1985. You could implement your vision in everything from Microsoft .NET CF to Java Micro Edition to BREW. But like the games I coded with my friends, our applications were isolated within our chosen ecosystem.

As 2011 dawns, by spreading the Android OS across hardware vendors, Google looks to be the Microsoft of the Mobile Space. That is likely why you have picked up this book and are reading this foreword. Either you are a student of history or, like me, you were lucky enough to live it.

Well, good news! We have worked very hard in this edition of the book to ensure you have the tools to implement the ideas rattling around in your imagination. We take you from the basics of setting up your environment through deploying to the marketplace. Of course this is a vast journey, so we mainly take you down the road most travelled. But we will provide you plenty of resources to explore on your own.

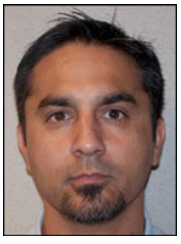
Good luck, and happy trails.

—Dylan Phillips

About the Authors

Satya Komatineni (www.satyakomatineni.com) has over 20 years of programming experience working with small and large corporations. Satya has published over 30 articles on web development using Java, .NET, and database technologies. He is a frequent speaker at industry conferences on innovative technologies and a regular contributor to the weblogs on Java.net. He is the author of AspireWeb (www.activeintellect.com/aspire), a simplified open source tool for Java web development, and the creator of Aspire Knowledge Central (www.knowledgefolders.com), an open source personal web operating system with a focus on individual productivity and publishing. Satya is also a contributing member to a number of Small Business Innovation Research Programs (SBIR). He received a bachelor's degree in electrical engineering from Andhra University, Visakhapatnam, and a master's degree in electrical engineering from the Indian Institute of Technology, New Delhi. You can contact him at satya.komatineni@gmail.com.

Dave MacLean is a software engineer and architect currently living and working in Jacksonville, Florida. Since 1980, he has programmed in many languages, developing solutions ranging from robot automation systems to data warehousing, from web self-service applications to EDI transaction processors. Dave has worked for Sun Microsystems, IBM, Trimble Navigation, General Motors, and several small companies. He graduated from the University of Waterloo in Canada with a degree in systems design engineering. Visit his blog at http://davemac327.blogspot.com or contact him at davemac327@gmail.com.



Sayed Y. Hashimi was born in Afghanistan and now resides in Jacksonville, Florida. His expertise spans the fields of health care, financials, logistics, and service-oriented architecture. In his professional career, Sayed has developed large-scale distributed applications with a variety of programming languages and platforms, including C/C++, MFC, J2EE, and .NET. He has published articles in major software journals and has written several other popular Apress titles. Sayed holds a master's degree in engineering from the University of Florida. You can reach him by visiting www.sayedhashimi.com.

Please visit the authors at their web site: www.androidbook.com.

About the Technical Reviewer

Dylan Phillips is a software engineer and architect who has been working in the mobile space for the last 10 years. With a broad range of experience ranging from J2ME to .NET Compact Framework to Android, he is incredibly excited about the opportunity presented by the broad consumer adoption of an array of Android devices. He can be reached at `mykoan@hotmail.com`, `@mykoan` on Twitter, or at lunch, in various Pho Houses around the country.

Acknowledgments

Writing this book took effort not only on the part of the authors, but also from some of the very talented staff at Apress, as well as the technical reviewer. Therefore, we would like to thank Steve Anglin, Matthew Moodie, Corbin Collins, Heather Lang, Tracy Brown, Mary Behr, and Brigid Duffy from Apress. We would also like to extend our appreciation to the technical reviewer, Dylan Phillips, for the work he did on this book. His commentary and corrections were invaluable. When searching for answers on the android developers forum, we were often helped by Dianne Hackborn, Nick Pelly, Brad Fitzpatrick, and other members of the Android Team, at all hours of the day and weekends, and to them we would like to say, "Thank you." They truly are the hardest working team in mobile. The Android community is very much alive and well and was also very helpful in answering questions and offering advice. We hope this book in some way is able to give back to the community. Finally, the authors are deeply grateful to their families for accommodating prolonged irresponsibility.

Preface

Have you ever wanted to be a Rodin? Sitting with a chisel and eroding away a block of rock to mold it to your vision? Well, mainstream programmers have kept away from the severely constrained mobile devices for fear of being unable to chisel out at a workable application. Those times have passed.

Android OS places the incredible reach of a programmable device at your door step. In this book we want to positively confirm your suspicion that Android is a great OS to program with. If you are a Java programmer, you have a great opportunity to profit from this exciting, capable, general-purpose computing platform. We are excited about Android because it is an advanced platform that introduces a number of new paradigms in framework design (even with the limitations of a mobile platform).

This is our third edition on the subject of Android, and it's our best edition yet. *Pro Android 3* is an extensive programming guide. In this edition we've refined, rewritten, and enhanced everything from *Pro Android 2* to create a thoroughly updated guide for both beginners and professionals—the result of our three years of research. We cover over 100 topics in 31 chapters. This edition covers versions 2.3 and 3.0 of Android, the optimized versions of Android for phones and tablets, respectively.

In this edition we have beefed up Android internals by covering threads, processes, long running services, broadcast receivers, and alarm managers. We cover many more UI controls in this edition. We have over 150 pages of dedicated material on 3.0, covering fragments, fragment dialogs, ActionBar, and drag and drop. We have significantly enhanced the services and sensor chapters. OpenGL has been revised to include OpenGL ES 2.0.

Concepts, Code, and Tutorials are the essence of this book. Every chapter in the book reflects this philosophy. The self-contained tutorials in each chapter are annotated with expert advice. All projects in the book are available for download for easy importing into Eclipse. We have worked hard so that the code can also be compiled right out of the book. The list of files that goes into each project are explicitly catalogued and listed in each chapter for easy reference.

The areas we cover in the book include key concepts such as resources, intents, content providers, processes, threads, UI controls, broadcast receivers, services, and long running services. We have a lot of coverage on OpenGL ES 1.0 and 2.0 for OpenGL beginners. We have a lot of coverage on text to speech, sensors, and multi-touch. We are also able to incorporate a lot of coverage on 3.0 topics that include fragments, fragment dialogs, ActionBar, and drag and drop.

Finally, in this book we went beyond basics, asked tough questions on every topic, and documented the results (see the table of contents for the extensive list of what we cover in the book). We are also actively updating the supplemental website (`www.androidbook.com`) with current and future research material on the Android SDK. As you walk through the book, if you have any questions we are only an email away for a quick response.

Chapter 1

Introducing the Android Computing Platform

Computing continues to become more and more personalized and accessible. Handheld devices have largely transformed into computing platforms. Mobile phones are no longer just for talking—they have been capable of carrying data and video for some time. Be it a phone or a tablet, the mobile device is now so capable of general-purpose computing that it's becoming more like a PC. A number of traditional PC manufacturers such as ASUS, HP, and Dell are producing devices of various form factors based on the Android OS. The battles between operating systems, computing platforms, programming languages, and development frameworks are being shifted and reapplied to mobile devices.

We are also seeing a surge in mobile programming as more and more IT applications start to offer mobile counterparts. In this book, we'll show you how to take advantage of your Java skills to write programs for devices that run on Google's Android platform (`http://developer.android.com/index.html`), an open-source platform for mobile and tablet development.

> **NOTE:** We are excited about Android because it is an advanced platform that introduces a number of new paradigms in framework design (even with the limitations of a mobile platform).

In this chapter, we'll provide an overview of Android and its SDK, give a brief overview of key packages, introduce what we are going to cover in each chapter briefly, show you how to take advantage of Android source code, and highlight the benefits of programming for the Android platform.

A New Platform for a New Personal Computer

The fact that dedicated devices such as mobile phones can now count themselves among general-computing platforms is good news for programmers (see Figure 1–1).

Starting with Android 3.0, we can officially add tablets to this list. This trend makes programming for mobile devices possible with general-purpose computing languages, which increases the range and market share for mobile applications.

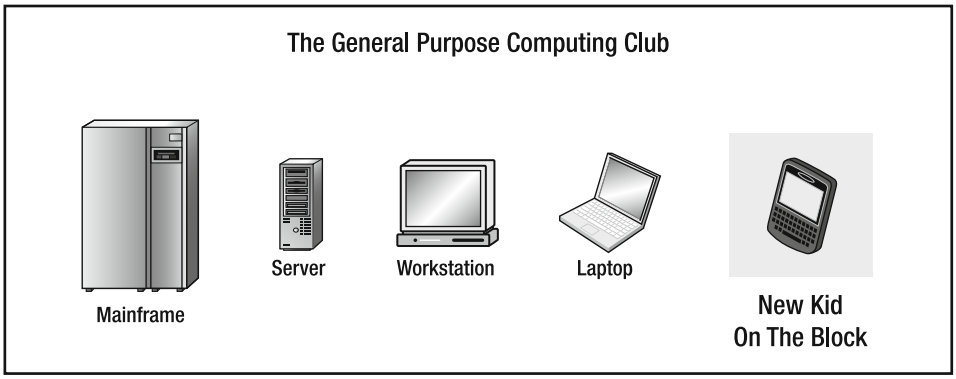


Figure 1–1. *Handheld is the new PC.*

The Android platform embraces this idea of general-purpose computing for handheld devices. It is a comprehensive platform that features a Linux-based operating system stack for managing devices, memory, and processes. Android's Java libraries cover telephony, video, speech, graphics, connectivity, UI programming, and a number of other aspects of the device.

> **NOTE:** Although built for mobile- and tablet-based devices, the Android platform exhibits the characteristics of a full-featured desktop framework. Google makes this framework available to Java programmers through a Software Development Kit (SDK) called the Android SDK. When you are working with the Android SDK, you rarely feel that you are writing to a mobile device because you have access to most of the class libraries that you use on a desktop or a server—including a relational database.

The Android SDK supports most of the Java Platform, Standard Edition (Java SE), except for the Abstract Window Toolkit (AWT) and Swing. In place of AWT and Swing, Android SDK has its own *extensive modern UI framework*. Because you're programming your applications in Java, you could expect that you need a Java Virtual Machine (JVM) that is responsible for interpreting the runtime Java byte code. A JVM typically provides the necessary optimization to help Java reach performance levels comparable to compiled languages such as C and C++. Android offers its own optimized JVM to run the compiled Java class files in order to counter the handheld device limitations such as memory, processor speed, and power. This virtual machine is called the Dalvik VM, which we'll explore in a later section "Delving into the Dalvik VM."

NOTE: The familiarity and simplicity of the Java programming language, coupled with Android's extensive class library, makes Android a compelling platform to write programs for.

Figure 1–2 provides an overview of the Android software stack. (We'll provide further details in the section "Understanding the Android Software Stack.")

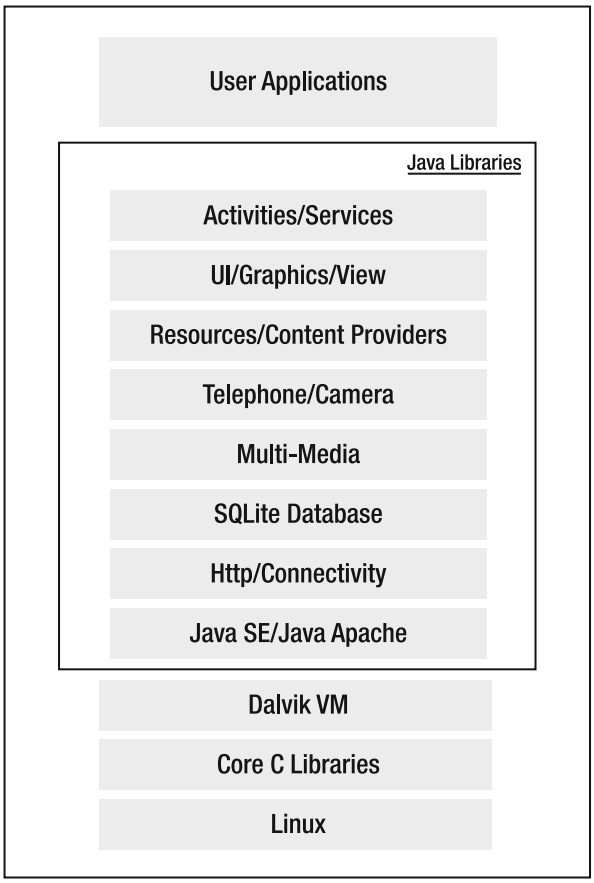


Figure 1–2. *High-level view of the Android software stack*

Early History of Android

Mobile phones use a variety of operating systems, such as Symbian OS, Microsoft's Windows Mobile, Mobile Linux, iPhone OS (based on Mac OS X), Moblin (from Intel), and many other proprietary OSes. So far, no single OS has become the de facto standard. The available APIs and environments for developing mobile applications are too restrictive and seem to fall behind when compared to desktop frameworks. In contrast, the Android platform promised openness, affordability, open-source code, and, more important, a high-end, all-in-one-place, consistent development framework.

Google acquired the startup company Android Inc. in 2005 to start the development of the Android platform (see Figure 1–3). The key players at Android Inc. included Andy Rubin, Rich Miner, Nick Sears, and Chris White.

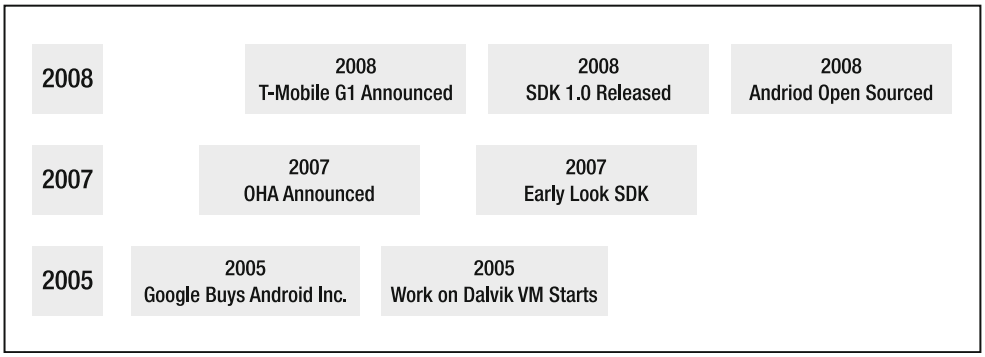


Figure 1–3. *Android early timeline*

In late 2007, a group of industry leaders came together around the Android platform to form the Open Handset Alliance (`www.openhandsetalliance.com`). Some of the alliance's prominent members as of 2009 were as follows:

Sprint Nextel

T-Mobile

Motorola

Samsung

Sony Ericsson

Toshiba

Vodafone

Google

Intel

Texas Instruments

As of 2011, this list has grown by multifold (over 80 in number), as you can see at the Open Handset Alliance web site.

According to the site, part of the alliance's goal is to innovate rapidly and respond better to consumer needs in the mobile space and its first key outcome was the Android platform. Android was designed to serve the needs of mobile operators, handset manufacturers, and application developers. The members have committed to release significant intellectual property through the open source Apache License, Version 2.0.

The Android SDK was first issued as an "early look" release in November 2007. In September 2008, T-Mobile announced the availability of T-Mobile G1, the first smartphone based on the Android platform. A few days after that, Google announced

the availability of Android SDK Release Candidate 1.0. In October 2008, Google made the source code of the Android platform available under Apache's open source license. In late 2010, Google released Android SDK 2.3 for smartphones, code named Gingerbread, which was upgraded to 2.3.3 by March 2011. In early 2011 an optimized version of Android for tablets, Android 3.0 code named Honeycomb, was released. Motorola XOOM is one of the early tablets to carry this OS release.

When Android was released, one of its key architectural goals was to allow applications to interact with one another and reuse components from one another. This reuse not only applies to services, but also to data and the user interface (UI). As a result, the Android platform has a number of architectural features that keep this openness a reality.

Android has attracted an early following and sustained the developer momentum because of its fully developed features to exploit the cloud-computing model offered by Web resources and to enhance that experience with local data stores on the handset itself. Android's support for a relational database on the handset also played a part in early adoption.

In releases 1.0 and 1.1 (2008) Android did not support soft keyboards, requiring the devices to carry physical keys. Android fixed this issue by releasing the 1.5 SDK in April 2009, along with a number of other features, such as advanced media-recording capabilities, widgets, and live folders.

In September 2009 came release 1.6 of the Android OS and, within a month, Android 2.0 followed, facilitating a flood of Android devices in time for the 2009 Christmas season. This release introduced advanced search capabilities and text to speech.

With support for HTML 5, Android 2.0 introduces interesting possibilities for using HTML. The contact API is significantly overhauled. Support for Flash is added. More and more Android-based applications are introduced every day, as well as new types of independent online application stores. Much anticipated tablet computers based on Android can now be purchased.

In Android 2.3 the significant features include remote wiping of secure data by administrators, the ability to use camera and video in low-light conditions, WiFi hotspot, significant performance improvements, improved Bluetooth functionality, installation of applications on the SD card optionally, OpenGL ES 2.0 support, improvements in backup, improvements in search usability, Near Field Communications support for credit card processing, much improved motion and sensor support (similar to Wii), video chat, and improved Market.

The latest incarnation of Android, 3.0 is focused on tablet-based devices and much more powerful dual core processors such as Nvidia Tegra2. The main features of this release include support to use larger screen. A significantly new concept called Fragments has been introduced. This permeates the 3.0 experience. More desktop-like capabilities, such as ActionBar and Drag and Drop, have been introduced. Home screen widgets have been significantly enhanced. More UI controls are now available. In the 3D space, OpenGL has been enhanced with Renderscript to further supplement ES 2.0. It is an exciting introduction for tablets.

Delving Into the Dalvik VM

As part of Android, Google has spent a lot of time thinking about optimizing designs for low-powered handheld devices. Handheld devices lag behind their desktop counterparts in memory and speed by eight to ten years. They also have limited power for computation. The performance requirements on handsets are severe as a result, requiring handset designers to optimize everything. If you look at the list of packages in Android, you'll see that they are fully featured and extensive.

These issues led Google to revisit the standard JVM implementation in many respects. 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. It reuses duplicate information from multiple class files, effectively reducing the space requirement (uncompressed) by half from a traditional .jar file

Google has also fine-tuned the garbage collection in the Dalvik VM, but it has chosen to omit a just-in-time (JIT) compiler, in early releases. Android 2.3 has added JIT. The reports are that this can give two to five times faster raw performance at places and 10 to 20% for general-purpose applications.

Dalvik VM uses a different kind of assembly-code generation, in which it uses registers as the primary units of data storage instead of the stack. Google is hoping to accomplish 30% fewer instructions as a result. We should point out that 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 you cannot directly execute Java byte code; you have to start with Java class files and then convert them to linkable .dex files.

This performance paranoia extends into the rest of the Android SDK. For example, the Android SDK uses XML extensively to define UI layouts. However, all of this XML is compiled to binary files before these binary files become resident on the devices. Android provides special mechanisms to use this XML data.

Understanding the Android Software Stack

So far we've covered Android's history and its optimization features including the Dalvik VM, and we've hinted at the Java programming stack available. In this section, we will cover the development aspect of Android. Figure 1–4 is a good place to start this discussion.

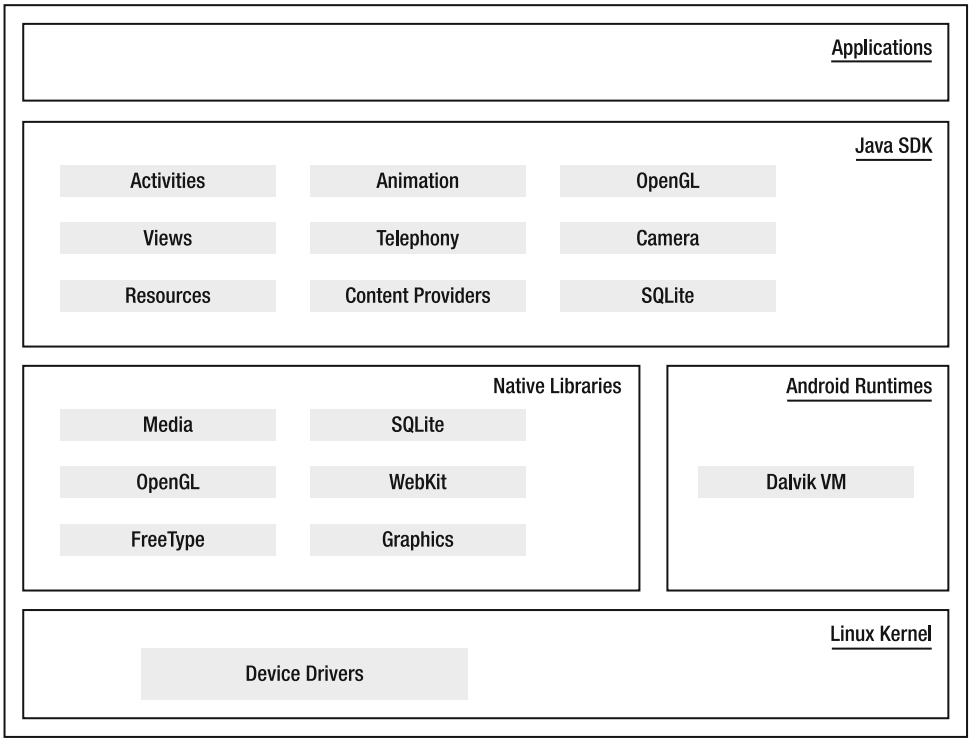


Figure 1–4. *Detailed Android SDK software stack*

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, WiFi, Flash Memory, Audio, and IPC (inter-process communication). Although the core is Linux, the majority—if not all—of the applications on an Android device such as a Motorola Droid are developed in Java and run through the Dalvik VM.

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 system C library based on Berkeley Software Distribution (BSD) is tuned (to roughly half its original size) for embedded Linux-based devices. The media libraries are based on PacketVideo's (`www.packetvideo.com/`) 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. More of these native libraries are likely to be added with new releases.

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 (`www.sqlite.org/`) 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. You can acquire and use tools meant for SQLite for Android databases as well.