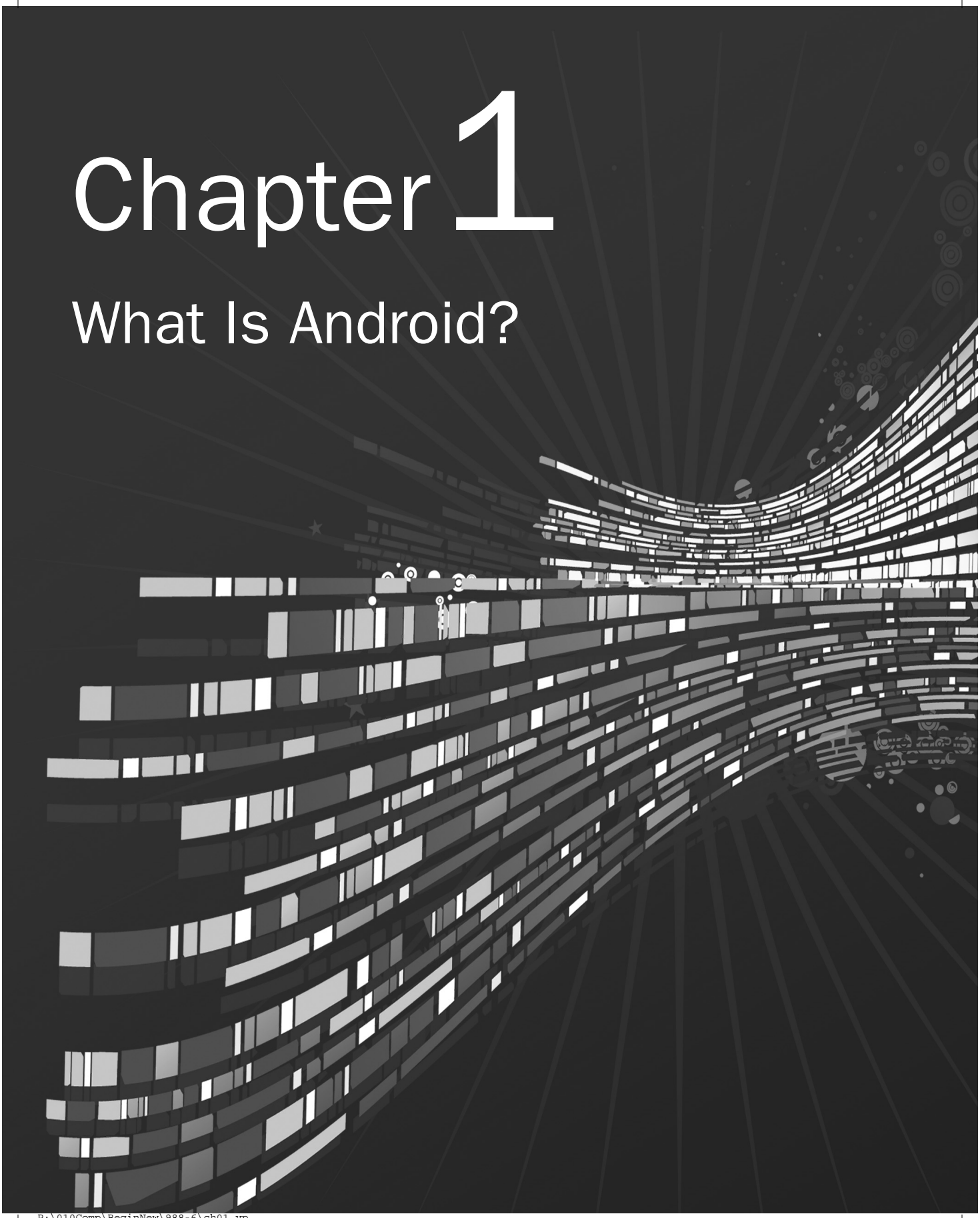
Chapter 1

What Is Android?



Key Skills & Concepts

- History of embedded device programming
- Explanation of Open Handset Alliance
- First look at the Android home screen

It can be said that, for a while, traditional desktop application developers have been spoiled. This is not to say that traditional desktop application development is easier than other forms of development. However, as traditional desktop application developers, we have had the ability to create almost any kind of application we can imagine. I am including myself in this grouping because I got my start in desktop programming.

One aspect that has made desktop programming more accessible is that we have had the ability to interact with the desktop operating system, and thus interact with any underlying hardware, pretty freely (or at least with minimal exceptions). This kind of freedom to program independently, however, has never really been available to the small group of programmers who dared to venture into the murky waters of cell phone development.

NOTE

I refer to two different kinds of developers in this discussion: *traditional desktop application developers*, who work in almost any language and whose end product, applications, are built to run on any "desktop" operating system; and *Android developers*, Java developers who develop for the Android platform. This is not for the purposes of saying one is by any means better or worse than the other. Rather, the distinction is made for purposes of comparing the development styles and tools of desktop operating system environments to the mobile operating system environment, Android.

Brief History of Embedded Device Programming

For a long time, cell phone developers comprised a small sect of a slightly larger group of developers known as embedded device developers. Seen as a less "glamorous" sibling to desktop—and later web—development, embedded device development typically got the

proverbial short end of the stick as far as hardware and operating system features, because embedded device manufacturers were notoriously stingy on feature support. Embedded device manufacturers typically needed to guard their hardware secrets closely, so they gave embedded device developers few libraries to call when trying to interact with a specific device.

Embedded devices differ from desktops in that an embedded device is typically a "computer on a chip." For example, consider your standard television remote control; it is not really seen as an overwhelming achievement of technological complexity. When any button is pressed, a chip interprets the signal in a way that has been programmed into the device. This allows the device to know what to expect from the input device (key pad), and how to respond to those commands (for example, turn on the television). This is a simple form of embedded device programming. However, believe it or not, simple devices such as these are definitely related to the roots of early cell phone devices and development.

Most embedded devices ran (and in some cases still run) proprietary operating systems. The reason for choosing to create a proprietary operating system rather than use any consumer system was really a product of necessity. Simple devices did not need very robust and optimized operating systems.

As a product of device evolution, many of the more complex embedded devices, such as early PDAs, household security systems, and GPSs, moved to somewhat standardized operating system platforms about five years ago. Small-footprint operating systems such as Linux, or even an embedded version of Microsoft Windows, have become more prevalent on many embedded devices. Around this time in device evolution, cell phones branched from other embedded devices onto their own path. This branching is evident when you examine their architecture.

Nearly since their inception, cell phones have been fringe devices insofar as they run on proprietary software—software that is owned and controlled by the manufacturer, and is almost always considered to be a "closed" system. The practice of manufacturers using proprietary operating systems began more out of necessity than any other reason. That is, cell phone manufacturers typically used hardware that was completely developed in-house, or at least hardware that was specifically developed for the purposes of running cell phone equipment. As a result, there were no openly available, off-the-shelf software packages or solutions that would reliably interact with their hardware. Since the manufacturers also wanted to guard very closely their hardware trade secrets, some of which could be revealed by allowing access to the software level of the device, the common practice

was, and in most cases still is, to use completely proprietary and closed software to run their devices. The downside to this is that anyone who wanted to develop applications for cell phones needed to have intimate knowledge of the proprietary environment within which it was to run. The solution was to purchase expensive development tools directly from the manufacturer. This isolated many of the "homebrew" developers.

NOTE

A growing culture of homebrew developers has embraced cell phone application development. The term "homebrew" refers to the fact that these developers typically do not work for a cell phone development company and generally produce small, one-off products on their own time.

Another, more compelling "necessity" that kept cell phone development out of the hands of the everyday developer was the hardware manufacturers' solution to the "memory versus need" dilemma. Until recently, cell phones did little more than execute and receive phone calls, track your contacts, and possibly send and receive short text messages; not really the "Swiss army knives" of technology they are today. Even as late as 2002, cell phones with cameras were not commonly found in the hands of consumers.

By 1997, small applications such as calculators and games (Tetris, for example) crept their way onto cell phones, but the overwhelming function was still that of a phone dialer itself. Cell phones had not yet become the multiuse, multifunction personal tools they are today. No one yet saw the need for Internet browsing, MP3 playing, or any of the multitudes of functions we are accustomed to using today. It is possible that the cell phone manufacturers of 1997 did not fully perceive the need consumers would have for an all-in-one device. However, even if the need was present, a lack of device memory and storage capacity was an even bigger obstacle to overcome. More people may have wanted their devices to be all-in-one tools, but manufacturers still had to climb the memory hurdle.

To put the problem simply, it takes memory to store and run applications on any device, cell phones included. Cell phones, as a device, until recently did not have the amount of memory available to them that would facilitate the inclusion of "extra" programs. Within the last two years, the price of memory has reached very low levels. Device manufacturers now have the ability to include more memory at lower prices. Many cell phones now have more standard memory than the average PC had in the mid-1990s. So, now that we have the need, and the memory, we can all jump in and develop cool applications for cell phones around the world, right? Not exactly.

Device manufacturers still closely guard the operating systems that run on their devices. While a few have opened up to the point where they will allow some Java-based applications to run within a small environment on the phone, many do not allow this. Even the systems that do allow some Java apps to run do not allow the kind of access to the "core" system that standard desktop developers are accustomed to having.

Open Handset Alliance and Android

This barrier to application development began to crumble in November of 2007 when Google, under the Open Handset Alliance, released Android. The Open Handset Alliance is a group of hardware and software developers, including Google, NTT DoCoMo, Sprint Nextel, and HTC, whose goal is to create a more open cell phone environment. The first product to be released under the alliance is the mobile device operating system, Android. (For more information about the Open Handset Alliance, see www.openhandsetalliance.com.)

With the release of Android, Google made available a host of development tools and tutorials to aid would-be developers onto the new system. Help files, the platform software development kit (SDK), and even a developers' community can be found at Google's Android website, http://code.google.com/android. This site should be your starting point, and I highly encourage you to visit the site.

NOTE

Google, in promoting the new Android operating system, even went as far as to create a $10 million contest looking for new and exciting Android applications.

While cell phones running Linux, Windows, and even PalmOS are easy to find, as of this writing, no hardware platforms have been announced for Android to run on. HTC, LG Electronics, Motorola, and Samsung are members of the Open Handset Alliance, under which Android has been released, so we can only hope that they have plans for a few Android-based devices in the near future. With its release in November 2007, the system itself is still in a software-only beta. This is good news for developers because it gives us a rare advance look at a future system and a chance to begin developing applications that will run as soon as the hardware is released.

NOTE

This strategy clearly gives the Open Handset Alliance a big advantage over other cell phone operating system developers, because there could be an uncountable number of applications available immediately for the first devices released to run Android.

Introduction to Android

Android, as a system, is a Java-based operating system that runs on the Linux 2.6 kernel. The system is very lightweight and full featured. Figure 1-1 shows the unmodified Android home screen.

Figure 1-1 The current Android home screen as seen on the Android Emulator.

Android applications are developed using Java and can be ported rather easily to the new platform. If you have not yet downloaded Java or are unsure about which version you need, I detail the installation of the development environment in Chapter 2. Other features of Android include an accelerated 3-D graphics engine (based on hardware support), database support powered by SQLite, and an integrated web browser.

If you are familiar with Java programming or are an OOP developer of any sort, you are likely used to programmatic user interface (UI) development—that is, UI placement which is handled directly within the program code. Android, while recognizing and allowing for programmatic UI development, also supports the newer, XML-based UI layout. XML UI layout is a fairly new concept to the average desktop developer. I will cover both the XML UI layout and the programmatic UI development in the supporting chapters of this book.

One of the more exciting and compelling features of Android is that, because of its architecture, third-party applications—including those that are "home grown"—are executed with the same system priority as those that are bundled with the core system. This is a major departure from most systems, which give embedded system apps a greater execution priority than the thread priority available to apps created by third-party developers. Also, each application is executed within its own thread using a very lightweight virtual machine.

Aside from the very generous SDK and the well-formed libraries that are available to us to develop with, the most exciting feature for Android developers is that we now have access to anything the operating system has access to. In other words, if you want to create an application that dials the phone, you have access to the phone's dialer; if you want to create an application that utilizes the phone's internal GPS (if equipped), you have access to it. The potential for developers to create dynamic and intriguing applications is now wide open.

On top of all the features that are available from the Android side of the equation, Google has thrown in some very tantalizing features of its own. Developers of Android applications will be able to tie their applications into existing Google offerings such as Google Maps and the omnipresent Google Search. Suppose you want to write an application that pulls up a Google map of where an incoming call is emanating from, or you want to be able to store common search results with your contacts; the doors of possibility have been flung wide open with Android.

Chapter 2 begins your journey to Android development. You will learn the hows and whys of using specific development environments or integrated development environments (IDE), and you will download and install the Java IDE Eclipse.

Ask the Expert

Q: **What is the difference between Google and the Open Handset Alliance?**

A: Google is a member of the Open Handset Alliance. Google, after purchasing the original developer of Android, released the operating system under the Open Handset Alliance.

Q: **Is Android capable of running any Linux software?**

A: Not necessarily. While I am sure that there will be ways to get around most any open source system, applications need to be compiled using the Android SDK to run on Android. The main reason for this is that Android applications execute files in a specific format; this will be discussed in later chapters.