1

Overview of Android

The Android operating system (OS) has come a long way since the announcement of the Open Handset Alliance in late 2007. The idea of an open source OS for embedded systems was not new, but Google aggressively backing it definitely has helped push Android to the forefront in just a few years.

Many wireless carriers in multiple countries across various communication protocols have one or more Android phones available. Other embedded devices, such as tablets, netbooks, televisions, set-top boxes, and even automobiles, have also adopted the Android OS.

This chapter discusses various general aspects of Android useful for a developer. It provides a foundation for the creation of Android applications and a context for the recipes in the rest of this book.

The Evolution of Android

Google, seeing a large growth of Internet use and search in mobile devices, acquired Android, Inc., in 2005 to focus its development on a mobile device platform. Apple introduced the iPhone in 2007 with some ground-breaking ideas including multitouch and an open market for applications. Android was quickly adapted to include these features and to offer definite distinctions, such as more control for developers and multitasking. In addition, Android incorporates enterprise requirements, such as exchange support, remote wipe, and Virtual Private Network (VPN) support, to go after the enterprise market that Research In Motion has developed and held so well with its Blackberry models.

Device diversity and quick adaptation have helped Android grow its user base, but it comes with potential challenges for developers. Applications need to support multiple screen sizes, resolution ratios, keyboards, hardware sensors, OS versions, wireless data rates, and system configurations. Each can lead to different and unpredictable behavior, but testing applications across all environments is an impossible task.

Android has therefore been constructed to ensure as uniform an experience across platforms as possible. By abstracting the hardware differences, Android OS tries to insulate applications from device-specific modifications while providing the flexibility to tune aspects as needed. Future-proofing of applications to the introduction of new hardware

platforms and OS updates is also a consideration. This mostly works as long as the developer is well aware of this systematic approach. The generic Application Programming Interfaces (API) that Android offers and how to ensure device and OS compatibility are main threads discussed throughout this book.

Still, as with any embedded platform, extensive testing of applications is required. Google provides assistance to third-party developers in many forms as Android Development Tool (ADT) plugins for Eclipse (also as standalone tools) including real-time logging capabilities, a realistic emulator that runs native ARM code, and in-field error reports from users to developers of Android Market applications.

The Dichotomy of Android

Android has some interesting dichotomies. Knowing about them upfront is useful not only in understanding what Android is, but what it is not.

Android is an embedded OS that relies on the Linux kernel for core system services, but it is not embedded Linux. For example, standard Linux utilities such as X-windows and GNU C libraries are not supported. Writing applications for Android utilizes the Java framework, but it is not Java. Standard Java libraries such as Swing are not supported. Other libraries such as Timer are not preferred; they have been replaced by Android's own libraries, which are optimized for usage in a resource-constrained, embedded environment.

The Android OS is open source, which means developers can view and use any of the system source code, including the radio stack. This source code is one of the first resources for seeing examples of Android code in action, and it helps clarify the usage when documentation is lacking. This also means developers can utilize the system in the same way as any core application and can swap out system components for their own components. However, Android devices do contain some proprietary software that is inaccessible to developers (such as Global Positioning System (GPS) navigation).

A final dichotomy of Android OS is that Google is also backing Chrome OS. Android OS is built for embedded platforms, and Chrome OS is built for cloud-based platforms. However, which is the best choice for embedded devices that live in the cloud? Netbooks, which fill the gap between smart phones and laptop computers, could presumably go either way (and they have). Android has started to utilize the cloud more. Does that mean Chrome OS's days are numbered? Google also backs a web-based market, so Chrome OS enjoys the same developer leverage that Android currently has. This points to a convergence that might have been in the cards all along.

Devices Running Android

There are more than 40 Android phones in the market from more than ten manufacturers. Other hardware also runs Android, such as tablets and televisions. Software can access information on the target device using the `android.os.Build` class, for example:

```
if(android.os.Build.MODEL.equals("Nexus+One")) { ... }
```

Android-supported hardware shares some common features due to the nature of the operating system. The Android OS is organized into the following images:

- Bootloader—Initiates loading of the boot image during startup
- Boot image—Kernel and RAMdisk
- System image—Android operating system platform and apps
- Data image—User data saved across power cycles
- Recovery image—Files used for rebuilding or updating the system
- Radio image—Files of the radio stack

These images are stored on nonvolatile flash memory, so they are protected when the device powers down. The flash memory is used like read-only memory (hence, some call it ROM), but can it be rewritten as necessary (for example, with over-the-air Android operating system updates).

On startup, the microprocessor executes the bootloader to load the kernel and RAMdisk to RAM for quick access. The microprocessor then executes instructions and pages portions of the system and data images into RAM as needed. The radio image resides on the baseband processor, which connects to the radio hardware.

A comparison of some of the early and more recent smart phone models is shown in Table 1.1. It shows that the processing hardware architecture is similar across devices: a microprocessor unit (MPU), synchronous dynamic random access memory (SDRAM or RAM for short), and flash memory (called ROM for short). The screen size is given in pixels, but the dots per inch (dpi) vary depending on the physical screen size. For example, the HTC Magic has a 3.2-inch diagonal screen with 320x480 pixels. This equates to 180 pixels per inch, but is classified as a medium pixel density device by Android (which averages as 160 dpi). All smartphones also offer a CMOS image sensor camera, Bluetooth (BT), and Wi-Fi (802.11), although there are variations.

Table 1.1 **Comparison of Some Representative Android Smartphones. Data from http://en.wikipedia.org/wiki/List_of_Android_devices and http://pdadb.net/.**

Model	MPU	RAM/ ROM	Screen	Other Features
HTC Dream / G1 (October 2008)	528-MHz QCOM MSM7201A	192MB/ 256MB	TFT LCD 320x480 mdpi	GSM/UMTS slide out keyboard, trackball, AGPS BT2.0, 802.11b/g, 3.1-MP camera

Table 1.1 **Continued**

Model	MPU	RAM/ ROM	Screen	Other Features
Samsung Moment (November 2009)	800-MHz ARM1176 JZF-S	288MB/ 512MB	AMOLED 320x480 mdpi	CDMA/1xEV-DO slide out keyboard (backlit), DPAD BT2.0, 802.11b/g, 3.1-MP camera AGPS
Motorola Milestone / Droid (November 2009)	550-MHz TI OMAP3430	256MB/ 512MB	TFT LCD 480x854 hdpi	GSM/UMTS or CDMA/1xEV-DO slide out keyboard, DPAD BT2.1, 802.11b/g, 5-MP camera AGPS
Nexus One / HTC Passion (January 2010)	1-GHz QCOM Snapdragon	512MB/ 512MB	AMOLED 480x800 hdpi	GSM/UMTS Trackball, dual microphones BT2.0, 802.11a/b/g/n, 5-MP camera AGPS, geotagging
HTC Droid Incredible (April 2010)	1-GHz QCOM Snapdragon	512MB/ 512MB	AMOLED 480x800 hdpi	CDMA/1xEV-DO BT2.1, 802.11a/b/g/n, 8-MP camera AGPS, geotagging
HTC EVO 4G (June 2010)	1-GHz QCOM Snapdragon	512MB/ 1GB	TFT LCD 480x800 hdpi	CDMA/1xEV-DO/802.16e-2005 BT2.1, 802.11b/g, 8-MP camera 1.3MP front-facing camera, AGPS

Table 1.1 **Continued**

Model	MPU	RAM/ ROM	Screen	Other Features
Motorola Droid X (July 2010)	1-GHz TI OMAP3630	512MB/ 8GB	TFT LCD 480x854 hdpi	CDMA/1xEV-DO, FM radio BT2.1, 802.11b/g/n, 8-MP camera AGPS, geotagging
Sony-Ericsson Xperia X10a (June 2010)	1-GHz QCOM Snapdragon	256MB/ 1GB	TFT LCD 480x854 hdpi	GSM/UMTS, FM radio BT2.1, 802.11b/g, 8-MP camera AGPS, geotagging
Samsung Galaxy S Pro (August 2010)	1-GHz Samsung Hummingbird	512MB/ 2GB	AMOLED 480x800 hdpi	CDMA/1xEV-DO, 802.16, FM radio slide out keyboard BT3.0, 802.11b/g/n, 5-MP camera 0.3MP front-facing camera, AGPS
Acer Stream / Liquid (September 2010)	1-GHz QCOM Snapdragon	512MB/ 512MB	AMOLED 480x800 hdpi	GSM/UMTS, FM radio BT2.1, 802.11b/g/n, 5-MP camera AGPS, geotagging

Other than improved capacity and performance on newer models, another main differentiator is additional features. Some devices offer 4G, some have FM radio, some have slide-out keyboards, and some have a front-facing camera. Knowing the differentiators helps a developer create great applications. In addition to the built-in hardware, every Android device comes with a secure digital (SD) card slot. An SD card provides additional storage space for multimedia and extra application data. However, until Android 2.2, the apps themselves could be stored only on the internal ROM.

HTC Models

HTC is a Taiwanese company founded in 1997. The first commercially available hardware running Android was the HTC Dream (also known as the G1 with G standing for Google). It was released in October 2008. Since then, HTC has put out over ten phones running Android, including Google's Nexus One.

The Nexus One was one of the first Android devices to use a 1-GHz microprocessor, the Snapdragon platform from Qualcomm. The Snapdragon includes Qualcomm's own core as opposed to an ARM core, and it contains circuitry to decode high-definition video at 720p. Most smartphones that have followed also utilize a 1-GHz microprocessor. Other distinctions of the Nexus One are the use of two microphones to cancel background noise during phone conversations and a backlit trackball that lights up different colors based on the notification.

HTC also released the Droid Incredible in April 2010. As seen in Table 1.1, it is similar to the Nexus One but has a CDMA instead of a GSM radio hardware and a higher pixel density camera. The HTC EVO 4G released in June 2010 produced quite a sensation as the first commercially available phone that supports WiMAX (802.16e-2005).

Motorola Models

Motorola built the first cell phone in the 1980s and has had diverse success in the cell phone market since. More recently, the wireless division was wavering for a direction until it focused efforts on Android. The release of the Motorola Droid for CDMA (also known as the Milestone for the GSM worldwide version) in November 2009 is indeed considered by many as a major milestone for Android. The Droid's impact is apparent in that a significant fraction of Android phones accessing the Android Market are Droids.

In addition, Motorola has put out close to ten additional phone brands running Android. The Motorola Droid X has capabilities similar to the HTC Droid Incredible, including HD video capture.

Samsung Models

Samsung has been a strong force in the mobile market and is starting to come into its own with Android devices. The Samsung Moment was introduced in November 2009, but does not have hardware capability for multitouch. It will not be upgraded beyond Android 2.1. A custom version, including a Mobile TV antenna, is available in select markets for receiving Mobile ATSC signals.

The Samsung Galaxy S is Samsung's answer to the iPhone. It is well known that Samsung processors are used in the iPhone 3G and 3GS. With the Galaxy S, Samsung developed a 1-GHz Hummingbird processor with an ARM Cortex-8 core. It is also one of the first phones to offer Bluetooth 3.0 compatibility.

Tablets

With Apple's introduction of the iPad, Android manufacturers were expected to introduce tablet computers of their own. A tablet computer is loosely defined as having a screen of 4.8 inches or larger and Wi-Fi connectivity. Because many have 3G wireless service, they tend to be more like smartphones with large screens.

Archos was one of the first to market an Android tablet in late 2009. It has a diagonal screen size of 4.8 inches and is called the Archos 5. Archos has since introduced a 7-inch model called the Archos 7. These models come with an actual hard drive for more data storage. Dell has also introduced a 5-inch tablet called the Streak with plans for both a 7-inch and a 10-inch screen size model. Samsung offers the Galaxy Tab with a 7-inch screen. One downside is the inability for many of these tablets to access the Android Market, although that should soon change. A comparison of some tablet computer models is shown in Table 1.2.

Table 1.2 **Comparison of Representative Android Tablet Computers**

Model	MPU	RAM/ disk	Screen	Other Features
Archos 5 (September 2009)	800-MHz TI OMAP 3440	256MB/ 8GB	TFT LCD 4.8 inches 800x480	BT2.0, 802.11b/g/n, FM radio
Archos 7 (June 2010)	600-MHz Rockchip RK2808	128MB/ 8GB	TFT LCD 7 inches 800x480	802.11b/g
Dell Streak (June 2010)	1-GHz QCOM Snapdragon	256MB/ 512MB	TFT LCD 5 inches 800x480	GSM/UMTS, BT2.1, 802.11b/g, 5-MP camera, 0.3-MP front-facing camera AGPS, geotagging
Samsung Galaxy Tablet GT-P1000 (September 2010)	1-GHz Samsung Hummingbird	512MB/ 16GB	TFT LCD 7 inches 1024x600	GSM/UMTS BT3.0, 802.11b/g/n, 3.1-MP camera

Other Devices

Given Android is a generic embedded platform, it is expected to be utilized in many other industries beyond smartphones and tablet computers. The first Android-based automobile is the Roewe 350, which Shanghai Automotive Industry Corporation manufactures. Android is mainly used for GPS navigation but can also support web browsing.

The first Android-based television, Google TV, is a joint development between Google for software, Sony for televisions, Intel for processors, and Logitech for set-top boxes. It brings the Internet to televisions in a natural way, but it also provides access to the Android Market from the television.

Hardware Differences on Android Devices

The hardware available on each Android device varies, as seen in Table 1.1. In general, most of the differences are transparent to the developer and not covered further here. However, a few hardware differences are important to understand to assist in writing device-independent code. Screens, user input methods, and sensors are discussed here.

Screens

Two technologies used for displays are liquid crystal displays (LCD) and light-emitting diodes (LED). The two specific choices in Android phones are thin-film transistor (TFT) LCDs and active-matrix organic LED displays (AMOLED). A benefit of TFT displays is a longer lifetime. A benefit of AMOLED displays is no need for backlighting and therefore deeper blacks and lower power.

Overall, Android devices are categorized into small, normal, and large screens and low-, medium-, and high-pixel density. Note that the actual pixel density might vary but will be chosen as one of these. A summary of currently available device screens is shown in Table 1.3. Note that Table 1.1 provides the screen density classification for each device listed.

Table 1.3 **Summary of Device Screens Supported by Android**

Screen Type	Low-Density (~120ppi), ldpi	Medium-Density (~160ppi), mdpi	High-Density (~240ppi), hdpi
Small screen	**QVGA** (240x320), 2.6-inch to 3.0-inch diagonal		
Normal screen	**WQVGA** (240x400), 3.2-inch to 3.5-inch diagonal **FWQVGA** (240x432), 3.5-inch to 3.8-inch diagonal	**HVGA** (320x480), 3.0-inch to 3.5-inch diagonal	**WVGA** (480x800), 3.3-inch to 4.0-inch diagonal **FWVGA** (480x854), 3.5-inch to 4.0-inch diagonal
Large screen		**WVGA** (480x800), 4.8-inch to 5.5-inch diagonal **FWVGA** (480x854), 5.0-inch to 5.8-inch diagonal	

User Input Methods

Touchscreens enable users to interact with the visual display. There are three types of touchscreen technology:

- Resistive—Two resistive material layers sit on top of a glass screen. When a finger, stylus, or any object applies pressure, the two layers touch together and the location of the touch can be determined. Resistive touchscreens are cost-effective, but only 75 percent of the light shows through, and until recently, multitouch was not possible.
- Capacitive—A charged material layer is overlaid on a glass screen. When a finger or any conductive object touches the layer, some charge is drawn off, changing the capacitance, which is measured to determine the location of the touch. Capacitive touchscreens allow as much as 90 percent of the light through, although accuracy can be less than resistive.
- Surface Acoustic Wave—This uses a more advanced method that sends and receives ultrasonic waves. When a finger or any object touches the screen, the waves are absorbed. The waves are measured to determine the location of the touch. It is the most durable solution, but more suitable for large-scale screens such as automatic bank tellers.

All Android devices use either resistive or capacitive touchscreen technology, and with a few early exceptions, all support multitouch.

In addition, each Android device needs an alternative method to access the screen. This is through one of the following methods:

- D-pad (directional pad)—An up-down-right-left type of joystick
- Trackball—A rolling ball acting as a pointing device that is similar to a mouse
- Trackpad—A special rectangular surface acting as a pointing device

Sensors

Smartphones are becoming sensor hubs in a way, opening a rich experience for users. Other than the microphone that every phone has, the first additional sensor introduced on phones was the camera. Different phone cameras have varying capabilities, and this is an important factor for people in selecting a device. The same type of diversity is now seen with the additional sensors.

Most smartphones have at least three basic sensors: a three-axis accelerometer to measure gravity, a three-axis magnetometer to measure the ambient magnetic field, and a temperature sensor to measure the ambient temperature. For example, the HTC Dream (G1) contains the following sensors (which can be displayed using `getSensorList()` as described further in Chapter 7, "Hardware Interface"):

```
AK8976A 3-axis Accelerometer
AK8976A 3-axis Magnetic field sensor
AK8976A Orientation sensor
AK8976A Temperature sensor
```

The AK8976A is a single package from Asahi Kasei Microsystems (AKM) that combines a piezoresistive accelerometer, Hall-effect magnetometer, and temperature sensor. All provide 8-bit precision data. The orientation sensor is a virtual sensor that uses the accelerometer and magnetometer to determine the orientation.

For comparison, the Motorola Droid contains the following sensors:

```
LIS331DLH 3-axis Accelerometer
AK8973 3-axis Magnetic field sensor
AK8973 Temperature sensor
SFH7743 Proximity sensor
Orientation sensor type
LM3530 Light sensor
```

The LIS331DLH is a 12-bit capacitive accelerometer from ST Microelectronics. It provides much more accurate data and can sample up to 1kHz. The AK8973 is an AKM package with an 8-bit Hall-effect magnetometer and temperature sensor.

In addition, the Droid contains two more sensors. The SFH7743 is an Opto Semiconductor's short-range proximity detector that turns the screen off when an object (such as the ear) is within about 40mm distance. The LM3530 is an LED driver with a programmable light sensor from National Semiconductor that detects ambient light and adjusts the screen backlight and LED flash appropriately.

One other example of sensors available on an Android device is the HTC EVO 4G, which has the following sensors:

```
BMA150 3-axis Accelerometer
AK8973 3-axis Magnetic field sensor
AK8973 Orientation sensor
CM3602 Proximity sensor
CM3602 Light sensor
```

The BMA150 is a Bosch Sensortec 10-bit accelerometer which can sample up to 1.5kHz. The CM3602 is a Capella Microsystems, Inc., short distance proximity sensor and ambient light sensor combined into one.

Overall, it is important to understand each Android model has different underlying hardware. These differences can lead to varying performance and accuracy of the sensors.

Features of Android

The detailed features of Android and how to take advantage of them provide a main theme throughout this book. On a broader level, some key features of Android are major selling points and differentiators. It is good to be aware of these strong points of Android and utilize them as much as possible.

Multiprocess and App Widgets

The Android OS does not restrict the processor to a single application at a time. The system manages priorities of applications and threads within a single application. This has the benefit that background tasks can be run while a user engages the device in a foreground process. For example, while a user plays a game, a background process can check stock prices and trigger an alert as necessary.

App Widgets are mini applications that can be embedded in other applications (such as the Home screen). They can process events, such as start a music stream or update the outside temperature, while other applications are running.

Multiprocessing has the benefit of a rich user experience. However, care must be taken to avoid power-hungry applications that drain the battery. These multiprocess features are discussed further in Chapter 3, "Threads, Services, Receivers, and Alerts."

Touch, Gestures, and Multitouch

The touchscreen is an intuitive user interface for a hand-held device. If utilized well, it can transcend a need for detailed instructions. After a finger touches the screen, drags and flings are natural ways to interact with graphics. Multitouch provides a way to track more than one finger down at the same time. This is often used to zoom or rotate a view.

Some touch events are available transparently to the developer without the need to implement their detailed behaviors. Custom gestures can be defined as needed. It is important to try to maintain a consistent usage of touch events as compared to other applications. These touch events are discussed further in Chapter 5, "User Interface Events."

Hard and Soft Keyboards

One feature on a pocket device that galvanizes users is whether it should have a physical (also called hard) keyboard or software (also called soft) keyboard. The tactile feedback and definite placement of keys provided by a hard keyboard tends to make typing much faster for some, whereas others prefer the sleek design and convenience offered by a software-only input device. With the large variety of Android devices available, either type can be found. A side effect for developers is the need to support both. One downside of a soft keyboard is a portion of the screen needs to be dedicated to the input. This needs to be considered and tested for any user interface (UI) layout.

Android Development

This book is focused on writing Android code, the main aspect of Android development. However, dedicating a few words to the other aspects of development, including design and distribution, is appropriate.

How to Use the Recipes in This Book

In general, the code recipes in this cookbook are self-contained and include all the information necessary to run a working application on an Android device. As discussed in detail in Chapter 2, "Application Basics: Activities and Intents," there are multiple user-generated files needed to get an application working. When even one is omitted from an example, its absence impedes those unfamiliar with the Android setup. Therefore, every recipe contains the necessary files to get code working. Each file is shown as a code listing with the full filename as the title. This helps to convey where the file lives in an Android project.

At the same time, when too many files are shown, it clouds functionality. Therefore, two coding styles are slightly different than would be expected in a normal application:

- The code has limited comments. The text explains the functionality clearer than in-line comments could, and **bolded code** shows the main lines needed to get the particular technique working. In practice, actual code should have more comments than presented in the recipes.
- Strings are explicit and do not point to a global resource. The method of using a global resource for strings is encouraged and discussed in detail in Chapter 4, "User Interface Layout," with multiple examples. In this book, however, when only a few strings are needed for a recipe, the strings are made explicit rather than including a whole additional file just to define them.

People just starting with Android are served well to use Eclipse for the development environment with the Android plugin. As discussed more in Chapter 2, this ensures proper Android project setup and context, and Eclipse even adds a placeholder icon figure. It also helps with more advanced tasks, such as signing an application for distribution.

The emulator provided with the Android Software Development Kit (SDK) is useful, but nothing beats seeing the application run on a true Android device. It leads to faster development and more realistic testing. All code examples in this book have been tested on an actual device running Android 2.1, and as needed, Android 1.5 or Android 2.2. Some functionality (for example, Bluetooth pairing or sensor changes) is difficult and opaque when using the emulator. Therefore, it is recommended that initial testing be done with an action Android device.

Designing Applications Well

Three elements are needed for an excellent application: a good idea, good coding, and good design. Often, the last element is paid the least attention because most developers work alone and are not graphic designers. Google must realize this because it has created a set of design guidelines: icon design, App Widget design, activity and task design, and menu design. These can be found at http://developer.android.com/guide/practices/ui_guidelines/.

Good design cannot be stressed enough. It sets an application apart, improves user adoption, and builds user appreciation. Some of the most successful apps on the Market

are a result of the collaboration between a developer and graphic designer. A significant portion of an app's development time should be dedicated to considering the best design for an app.

Maintaining Forward Compatibility

New Android versions are generally additive and forward compatible at the API level. In fact, a device can be called an Android device only if it passes compatibly tests with the Android APIs. However, if an application makes changes to the underlying system, compatibility is not guaranteed. To ensure forward compatibility of an application when future Android updates are installed on devices, follow these rules suggested by Google:

- Do not use internal or unsupported APIs.
- Do not directly manipulate settings without asking the user. A future release might constrain settings for security reasons. For instance, it used to be possible for an app to turn on GPS or data roaming by itself, but this is no longer allowed.
- Do not go overboard with layouts. This is rare, but complicated layouts (more than 10 deep or 30 total) can cause crashes.
- Do not make bad hardware assumptions. Not all Android devices have all possible supported hardware. Be sure to check for the hardware needed, and if it does not exist, handle the exception.
- Ensure device orientations do not disrupt the application or result in unpredictable behavior. Screen orientation can be locked, as described in Chapter 2.

Note that backward compatibility is not guaranteed with Android. It is best to declare the minimum SDK version as described in Chapter 2, so the device can load the proper compatibility settings. Utilizing other new features on older targets is also discussed at various places throughout the book.

Robustness

In the same vein as compatibility support, applications should be designed and tested for robustness. Following are a few tips to help ensure robustness:

- Use the Android libraries before Java libraries. Android libraries are constructed specifically for embedded devices and cover many of the requirements needed in an application. For the other cases, Java libraries are included. However, for cases where either can be used, the Android library is best.
- Take care of memory allocation. Initialize variables. Try to reuse objects rather than reallocate. This speeds up application execution and avoids excessive use of garbage collection. Memory allocations can be tracked using the Dalvik Debug Monitor Server (DDMS) tool as discussed in Chapter 12, "Debugging."

- Utilize the LogCat tool for debugging and check for warnings or errors as also discussed in Chapter 12.
- Test thoroughly, including different environments and devices if possible.

Software Development Kit

The Android SDK is comprised of the platform, tools, sample code, and documentation needed to develop Android applications. It is built as an add-on to the Java Development Kit and has an integrated plugin for the Eclipse Integrated Development Environment.

Installing and Upgrading

There are many places on the Internet that discuss detailed step-by-step instructions on how to install the Android SDK. For example, all the necessary links can be found on the Google website http://developer.android.com/sdk/. Therefore, the general procedure outlined here serves to emphasize the most common installation steps for reference. These steps should be done on a host computer used as the development environment.

1. Install the Java Development Kit (for example, install JDK 6.0 for use with Android 2.1 or above; JDK 5.0 is the minimum version needed for any earlier version of Android).
2. Install Eclipse Classic (for example, version 3.5.2). In the case of Windows, this just needs to be unzipped in place and is ready to use.
3. Install the Android SDK starter package (for example, version r06). In the case of Windows, this just needs to be unzipped in place and is ready to use.
4. Start Eclipse and select **Help → Install New Software**..., and then type https://dl-ssl.google.com/android/eclipse/ and install the Android DDMS and Android Development Tools.
5. In Eclipse, select **Window → Preferences**... (on a Mac, select **Eclipse → Preferences**) and select Android. Browse to the location where the SDK was unzipped and apply.
6. In Eclipse, select **Window → Android SDK and AVD Manager → Available Packages**, and then choose the necessary APIs to install (for example, Documentation for Android SDK, API 8; SDK Platform Android 2.2, API 8; Google APIs by Google Inc.; and Android API 8).
7. From the same Android SDK and AVD Manager menu, create an Android virtual device to run the emulator or install USB drivers to run applications on a plugged-in phone.
8. In Eclipse, select **Run → Run Configurations**... and create a new run configuration to be used with each Android application (or similar for a Debug Configuration). Android JUnit tests can be configured here, too.

Now, the environment should be configured to easily develop any Android application and run on the emulator or an actual Android device. To upgrade to a new version of the SDK, it is simply a matter of selecting **Help → Software Updates...** in Eclipse and choosing the appropriate version.

Software Features and API Level

The Android OS periodically rolls out new features, enhancements such as improved efficiency, and bug fixes. A main driver in OS improvement is the increased capability of hardware on new devices. In fact, major releases of the OS are generally coordinated with new hardware roll-outs (such as Eclair's release with Droid).

Some legacy Android devices cannot support the new version requirements and are not updated with new OS releases. This leads to a user base with a variety of different possible experiences. The developer is left with the task of checking for device capability or at least warning devices of required features. This can be done through a check of a single number: the API level.

The following summarizes the different OS releases and main features from a developer's perspective:

Cupcake: Android OS 1.5, API level 3, Released April 30, 2009

- Linux kernel 2.6.27.
- Smart virtual (soft) keyboard, support for third-party keyboards.
- AppWidget framework.
- Live Folders.
- Raw audio recording and playback.
- Interactive MIDI playback engine.
- Video recording APIs.
- Stereo Bluetooth support.
- Removed end-user root access (unless tethered to computer and using SDK).
- Speech recognition via RecognizerIntent (cloud service).
- Faster GPS location gathering (using AGPS).

Donut: Android OS 1.6, API Level 4, Released September 15, 2009

- Linux kernel 2.6.29.
- Support for multiple screen sizes.
- Gesture APIs.
- Text-to-speech engine.
- Integrate with the Quick Search Box using the SearchManager.
- Virtual Private Network (VPN) support.

Eclair: Android OS 2.0, API Level 5, Released October 26, 2009
Android OS 2.0.1, API Level 6, Released December 3, 2009
Android OS 2.1, API Level 7, Released January 12, 2010

- Sync adapter APIs to connect to any backend.
- Embed Quick Contact accessible in applications.
- Applications can control the Bluetooth connection to devices.
- HTML5 support.
- Microsoft Exchange support.
- Multitouch is accessible through the MotionEvent class.
- Animated wallpaper support.

FroYo: Android OS 2.2, API Level 8, Released May 20, 2010

- Linux kernel 2.6.32.
- Just-In-Time compilation (JIT) enabled, leading to faster code execution.
- Voice dialing using Bluetooth.
- Car and desk dock themes.
- Better definition of multitouch events.
- Cloud-to-device APIs.
- Applications can request to be installed on the SD memory card.
- Wi-Fi tether support on select devices.
- Thumbnail utility for videos and images.
- Multiple language support on keyboard input.
- Application error reporting for Market apps.

Android is starting to mature in that releases are less frequent. Although possible, the over-the-air updates are logistically tricky and carriers prefer to avoid them. Hardware manufacturers also appreciate a level of stability, which does not mean the first flashed devices in stores need an immediate update. However, when a release is made, the level of additional features for developers remains high and worthwhile to utilize.

Emulator and Android Device Debug

The emulator launches a window on the development computer that looks like an Android phone and runs actual ARM instructions. Note the initial startup is slow, even on high-end computers. Although there are ways to configure the emulator to try to emulate many aspects of a real Android device such as incoming phone calls, limited data rate, and screen orientation change, some features (such as sensors and audio/video) are not the same. The emulator should be considered a useful way to validate basic functionality for

devices not available to the user. For example, the tablet screen size can be tried without purchasing a tablet.

Note that a target virtual device must be created before the emulator can properly run. Eclipse provides a nice method to manage Android Virtual Devices (AVD). A handy list of keyboard shortcuts for emulator functions is shown in Table 1.4.

Table 1.4 **Android OS Emulator Controls**

Key	Emulated Function
Escape	Back button
Home	Home button
F2, PageUp	Menu button
Shift-F2, PageDown	Start button
F3	Call/Dial button
F4	Hangup/EndCall button
F5	Search button
F7	Power button
Ctrl-F3, Ctrl-KEYPAD_5	Camera button
Ctrl-F5, KEYPAD_PLUS	Volume up button
Ctrl-F6, KEYPAD_MINUS	Volume down button
KEYPAD_5	DPAD center
KEYPAD_4, KEYPAD_6	DPAD left, DPAD right
KEYPAD_8, KEYPAD_2	DPAD up, DPAD down
F8	Toggle cell network on/off
F9	Toggle code profiling (when **-trace** set)
Alt-ENTER	Toggle fullscreen mode
Ctrl-T	Toggle trackball mode
Ctrl-F11, KEYPAD_7 Ctrl-F12, KEYPAD_9	Rotate screen orientation to previous or next layout

In general, the first testing is best done with an Android phone. This ensures full functionality and real-time issues that cannot be fully recreated with the emulator. For an Android device to be used as a developer platform, just hook it to the USB using the USB cable that came with the phone and ensure the USB driver is detected (this is automatic with a MAC; the drivers are included with the SDK for Windows; and see Google's web page for Linux).

Some settings on the Android device need to be changed to enable developer usage. From the home screen, select **MENU→Settings→Applications→Unknown sources**

and **MENU→Settings→Applications→Development→USB debugging** to enable installation of applications through the USB cable. More details about Android debugging are provided in Chapter 12.

Using the Android Debug Bridge

It is often convenient to use the command line to access the Android device. This is possible when it is connected to a computer using the USB cable. The Android Debug Bridge, which comes with the SDK, can be used to access the Android device. For example, to log into the Android device as if it were a Linux computer, type the following:

```
> adb shell
```

Then, many UNIX commands are usable on the device. Use `exit` to exit the shell. A single command can be appended to this to be executed without needing to enter and exit the shell:

```
> adb shell mkdir /sdcard/app_bkup/
```

To copy files off the device, use `pull` and rename it as needed:

```
> adb pull /system/app/VoiceSearchWithKeyboard.apk VSwithKeyboard.apk
```

To copy a file onto the device, use `push`:

```
> adb push VSwithKeyboard.apk /sdcard/app_bkup/
```

To delete an application, for example `com.dummy.game`, from the device, type the following:

```
> adb uninstall com.dummy.game
```

These commands are the most commonly used, but more are available. Some additional commands are introduced in Chapter 12.

Signing and Publishing

For an application to be accepted on the Android Market, it needs to be signed. To do this, a private key needs to be generated and kept in a secure place. Then, the app needs to be packaged in release mode and signed with the private key. When an application is upgraded, the same key needs to sign it to ensure a transparent update for the user.

Eclipse automatically does all of this. Just right-click on the project to be signed and select **Export... → Export Android Application** to initiate packaging. A password can be used to create a private key, which is saved for future applications and upgrades. Then, continue through the menu to the creation of an APK file. This is a packaged version of the Android project in release mode and signed with the private key. It is ready for upload to the Android Market.

Android Market

After an application is designed, developed, tested, and signed, it is ready to be deployed into the Android Market. To use Google's Android Market, a Google Checkout account needs to be created. It is used not only to pay for the initial developer fee of $25, but is also used for payment back to the developer for any charged apps. Public exposure to a developer's creation is often exciting. Within hours of upload, the application can get hundreds of views, downloads, ratings, and reviews from around the world. A few considerations for publication of an app are provided here for reference.

End-User License Agreement

Any original content distributed in a tangible form is automatically copyrighted in most of the world under the Berne Convention. Still, it is common practice to add a copyright with a date of publication to the content, such as © 2010. The method for adding this symbol to an Android app is discussed in Chapter 4.

This can be taken one step further in an End User License Agreement (EULA), which is a contract between the developer (or company) and the customer (or end user) providing the developer a form of protection for publicly distributed software. Most EULAs contain sections such as "Grant of License," "Copyright," and "No Warranties." It is common practice to add a EULA to an application, especially if it is offered for sale. The method for adding a EULA to an Android app is discussed in Chapter 9, "Data Storage Methods."

Improving App Visibility

Users find applications in three different ways. Catering to these methods helps to increase visibility for an application.

The first way users see an app is by choosing to list the "Just in" apps. Choose a good descriptive name for the application and place it in an appropriate category, such as **Games** or **Communication**. Keep the description simple and to the point to get more views. The **Games** category is over laden with apps, so there are sub-categories. If the app is fun but has no score or goal, consider the **Entertainment** category. Even so, with over 10,000 applications uploaded to the Android Market each month, an uploaded application is pushed off the "Just in" list within a day or two.

The second way users see an app is by keyword search. Determine the essential keywords users might use and include those in either the title or description of the app. Some users might speak a different language, so including appropriate international keywords can help.

The third way users see an app is by choosing the "Top" apps. This is a combination of the highest rating and the most downloads. To get in this category takes time and effort with possible updates to fix bugs. This points to the last consideration for app visibility:

robustness. Ensure the app does not contain major bugs, does not waste excessive battery, and has a foolproof way to exit the application. Nothing turns off a potential customer more than seeing reviews that say, "This app uses all of my battery," or, "I can't uninstall this app."

One side note to mention: Almost all interactions between the developer and users are done through the Android Market. Providing developer contact information or a supporting website is often superfluous, as people browsing the mobile market rarely use it.

Differentiating an App

Sometimes, the developer creates an application only to find a similar variant already in the Android Market. This should be treated as an opportunity rather than a discouragement. Differentiating the app simply through a better design, interface, or execution can quickly win over a user base. Basically, originality is nice, but it is not required. That being said, one must be careful to avoid using copyrighted material.

Charging for an App

Every time a new application or its update is uploaded to the Android Market, the developer must choose whether to provide it for free or charge for it. Following are the main options:

- Provide the app for free. Everyone who can access the Android market can see and install the app.
- Provide a free app, but include advertisements. In some cases, the developer negotiates sponsorship for an app. More often, the developer works with a third-party aggregator. Payouts are provided for clicked ads and less often for impressions (ad views). Figure 1.1 shows an example banner ad from AdMob. Such ads require the application have permission to access the Internet and the location of the device. Consider using coarse location instead of fine location to avoid deterring some potential customers from installing the app.
- Provide the app for a charge. Google handles its charges, but takes 30 percent of the proceeds. Countries that are not set up for charges through Google Checkout cannot see or cannot install an app for charge. For these reasons, some developers turn to third-party app stores for distribution.
- Post a free, limited version, but charge for a full version. This gives users the opportunity to try the app and if they like it, they will have less resistance to purchasing the full version. For some apps, this is a natural model (such as a game with ten free levels), but not all apps can be partitioned this way.
- Sell virtual goods inside the app. This is an important way Facebook apps work, and it is catching on in the mobile world.