CHAPTER 6

Installing the Android SDK for Intel Application Development

How could this Y2K be a problem in a country where we have Intel and Microsoft?

—Al Gore

This chapter covers the information necessary to get started developing an Android application on an Intel Architecture processor. The first step is installing the software development kit (SDK) and setting up the appropriate environments for developing applications running on Intel Architecture-based Android devices. The SDK includes tools and platform components for developers to develop, build, test, debug, and optimize their Android applications, and manage the Android platform component installation. The SDK also provides easy ways to integrate with the build and development environments, for examples, with Eclipse or Apache Ant.

Preparing for the SDK Installation

This next section is dedicated to setting up a development environment available so that you can start producing Android applications on Intel platforms. If you already have an Android development environment setup, you can skip this section.

Supported Operating Systems

The following operating systems are supported:

- Windows XP (32-bit), Vista (32- or 64-bit), Windows 7 (32- or 64-bit), and Windows 8 (32- or 64-bit)
- Mac OS X (32- or 64-bit)
- Linux (Ubuntu, Fedora); GNU C library (**`glibc`**) 2.7 or later is required

- On Ubuntu Linux, version 8.04 or later is required
- On Fedora, target versions are F-12 and higher
- 64-bit distributions must be capable of running 32-bit applications

Hardware Requirements

The Android SDK requires disk storage for all of the components that you choose to install. Additional disk space is required to run the emulator, for example, to create SD cards for the Android Virtual Devices (AVDs).

Installing the JDK

At minimum, Java JDK 5 or JDK 6 is required by the SDK. JDK 7 is also supported. JRE (Java Runtime Environment) alone is not sufficient. If your system does not have JDK 5, 6, or 7 installed, you can download JDK SE 7 from http://www.oracle.com/technetwork/java/javase/downloads/index.html and install it on your system.

Installing Eclipse

Using the SDK with Eclipse to develop Android applications is highly recommended. You can go to http://www.eclipse.org/downloads/ to download or update Eclipse. We suggest using the following Eclipse setups to develop Android applications for Intel Architecture:

- Eclipse 3.5 (Galileo) or greater
- Eclipse Classic (versions 3.5.1 and higher)
- Android Development Tools plug-in (recommended)

Installing Apache Ant (Optional)

Developing Android applications using an integrated development environment such as Eclipse is highly recommended. But as an alternative, you can use Apache Ant to work with the SDK to build Android applications. You can visit http://ant.apache.org/ to download the binary distributions and install Ant. To work with the SDK, Ant 1.8 or later is required.

Downloading the SDK Starter Package and Adding SDK Components

You can download the SDK starter package at http://developer.android.com/sdk/index.html. The SDK starter package does not include the platform-specific components you need to develop Android applications. It only provides the core SDK tools for you to download the rest of the platform components.

After installing the SDK starter package, run the Android SDK and AVD Manager.

- On Windows, select Start ➤ All Programs ➤ Android SDK Tools ➤ SDK Manager
- On Linux, run **`your-android-sdk-directory/tools/android`**

In the left panel of the Android SDK and AVD Manager dialog box, select Available packages, and in the right panel, click and expand the Android Repository node and select the packages to install, as shown in Figure 6-1.

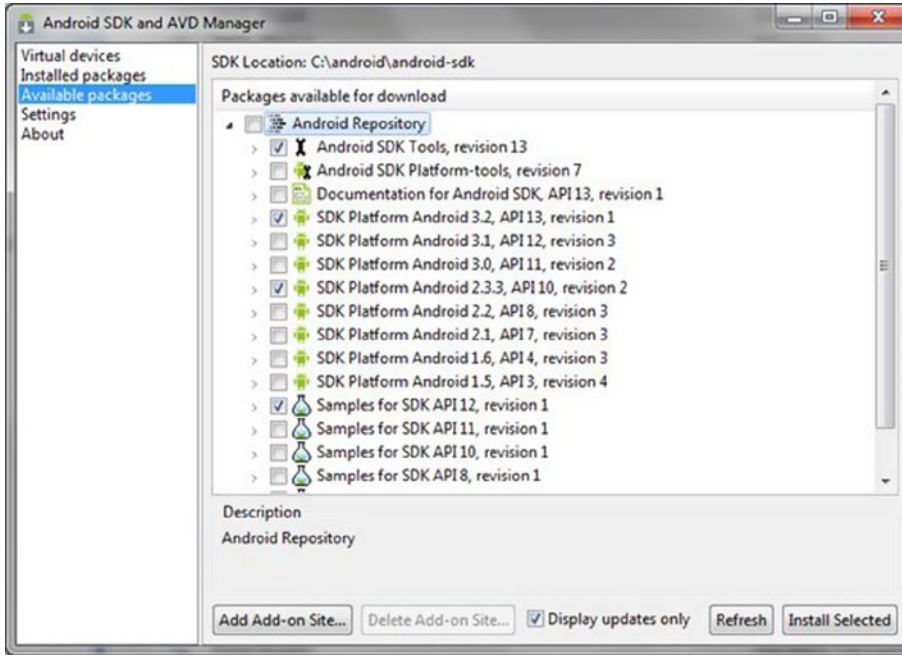

Figure 6-1. *Installing the Android SDK and AVD Manager in Linux*

■ **Note** You may see an error message displayed if you attempt to download from behind a firewall. If this occurs, try again from outside the firewall. If you still see an error message, close the SDK Manager, then right-click it in the Start menu and select Run as Administrator. These two steps will resolve most error messages you'll see when attempting to download.

Setting Up Eclipse to work with the SDK

If you use the Eclipse IDE to develop software, we highly recommend you install and set up the Android Development Tool (ADT) plug-in.

Installing the ADT Plug-in for Eclipse

To install the ADT plug-in for Eclipse, follow these steps:

1. Start Eclipse, select Help ➤ Install New Software in the Install dialog box, and click the Add button.
2. In the Add Repository dialog box, enter **`ADT Plugin`** in the Name field and enter `https://dl-ssl.google.com/android/eclipse/` in the Location fields, as shown in Figure 6-2. Then click OK.

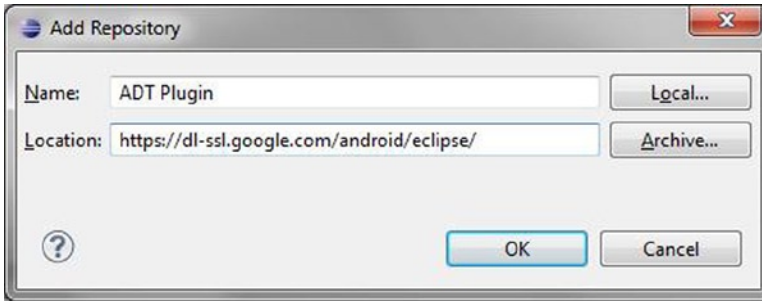

Figure 6-2. *Repository Dialog Box*

3. It will go back to the Install dialog box, connect to the Google repository server, and display the available ADT packages, as shown in Figure 6-3.

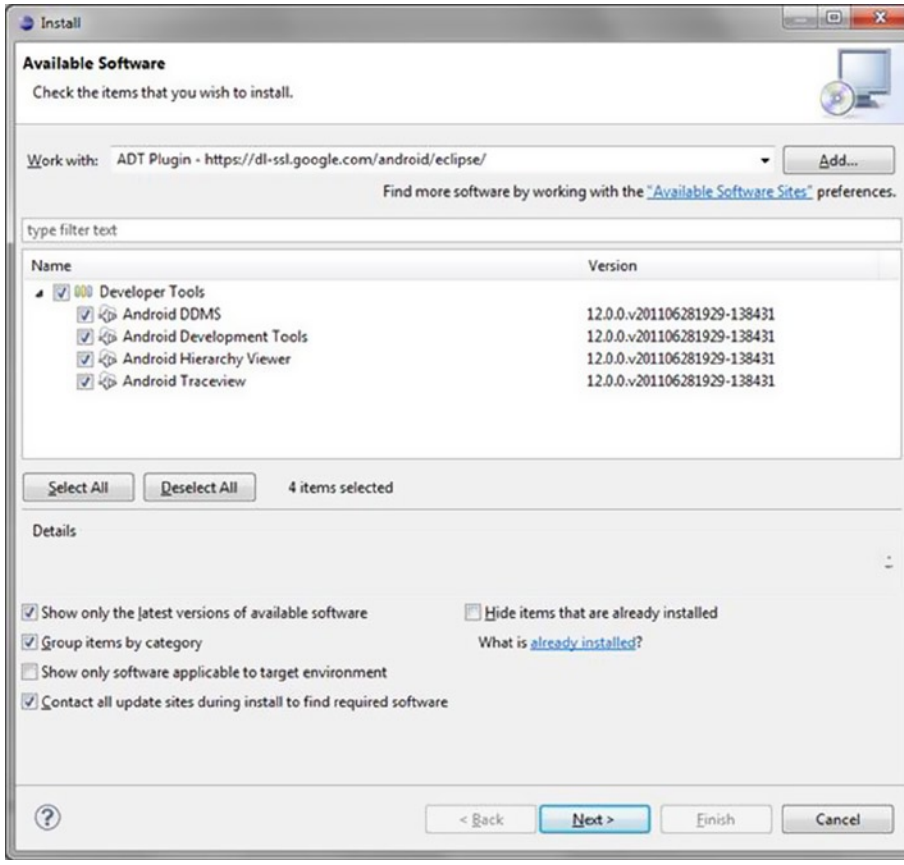

Figure 6-3. *ADT Package List*

4. Select Next, accept the license agreement, and then select Finish.
5. Restart Eclipse.

Configuring the ADT Plug-in

To configure the ADT plug-in, follow these steps:

1. Start Eclipse and select Windows ➤ Preferences.
2. In the Preferences dialog box, select Android from the left panel. On the right panel, use the Browse button to navigate to your Android SDK installation directory and click Apply. A list of SDK targets you have installed will show up, as shown in Figure 6-4. At this point, click OK.

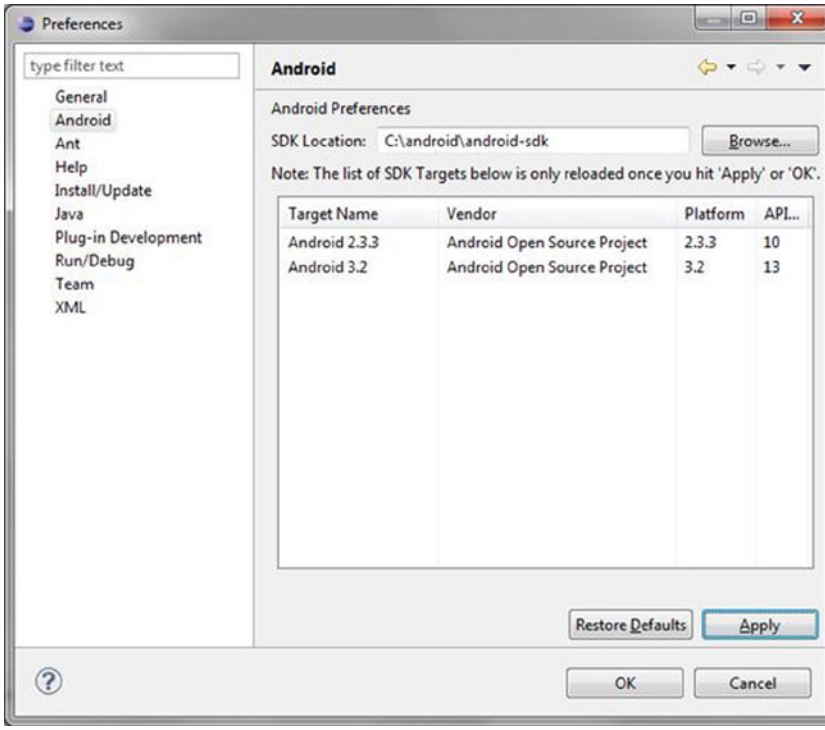

Figure 6-4. *ADT SDK Targets List*

After going through these steps, you will have the necessary tools available to start Android development. You now have everything you need to write your first app, but it would probably be best to also install the Intel Architecture (x86) emulator at this time, so that you can test your app as soon as it is ready. This next section takes you through the Intel Architecture (x86) emulator install. We discuss building an emulator image with AOSP sources, and emulating the resulting system images for x86.

The Android Developer Tools are updated regularly to include the latest APIs. When building an emulator image, the tools to build the most up-to-date Android version are at your fingertips. If you choose to download pre-built emulator system images, your choices will be a few months out of date. For this chapter, we use case studies emulating Android 2.3, codenamed Gingerbread and covering API levels 9 and 10, and Android 4.0, codenamed Ice Cream Sandwich and covering API levels 14 and 15. Gingerbread was the first Android release available on platforms other than ARM and its new features emerged directly from the Android open source project's efforts, which were discussed in **Chapter 1: History and Evolution of the Android OS**.

Overview of Android Virtual Device Emulation

Android runs on a variety of form factor devices with different screen sizes, hardware capabilities, and features. A typical device has a multitude of software (Android API) and hardware capabilities like sensors, GPS, camera, SD card, and a multitouch screen with specific dimensions.

The emulator is quite flexible and configurable with different software and hardware configuration options. Developers can customize the emulator using the emulator configuration called Android Virtual Device (AVD). AVD is basically a set of configuration files that specify different Android software and device hardware capabilities. The Android emulator uses these AVD configurations to configure and start the appropriate Android virtual image on the emulator.

As documented on the Android web site (see `http://developer.android.com/guide/developing/devices/index.html`), a typical AVD configuration has:

- A hardware profile that specifies all the device capabilities (such as cameras and sensors).
- A *system image*, which is used by the emulator for this AVD (designating which API level to target, such as 10 for Gingerbread or 19 for KitKat).
- A *data image* that acts as the dedicated storage space for user's data, settings, and SD card.
- Other options including emulator skin, the screen dimensions, and SD card size.

Developers are encouraged to target different API levels, screen sizes, and hardware capabilities (such as camera, sensors, and multitouch). The AVD configuration can be used to customize the emulator as needed. Developers can create as many AVDs as desired, each one targeting a different Intel architecture–based Android device. For example, a developer can create an Intel architecture–based Gingerbread AVD with a built-in skin like WVGA800, or a custom one that manually specifies the screen resolution.

The Android SDK has supported Intel architecture–based Android emulation since version r12. The SDK integrates this support into all developer tools, including the eclipse ADT plug-in. Figure 6-5 is a sample screenshot of the Android emulator for x86 running Gingerbread. The model number is highlighted, and shows Full Android on x86 emulator.

Figure 6-5. *Android Emulator*

For detailed instructions on how to use the emulator, refer to the following Android documentation: http://developer.android.com/tools/devices/emulator.html.

Which Emulator Should You Use

At the time of this writing, emulator images are available for Intel architecture (x86) for Android 2.3.7 (Gingerbread), Android 4.0.4 (Ice Cream Sandwich), and Android 4.3 (Jelly Bean). You can find the most recent image at http://software.intel.com/en-us/articles/android-43-jelly-bean-x86-emulator-system-image.

Although there are many advantages to developing for the latest Android operating system release, many developers prefer to target Android 2.x or Android 4.x, as a majority of Android phones run Android 4.x or higher. This percentage will change over time, so it is strongly suggested that you keep market conditions in mind when determining your target operating system.

For more Gingerbread-specific operating system information, the following article may prove useful: http://blogs.computerworld.com/17479/android_gingerbread_faq.

For Ice Cream Sandwich information, use this article: http://www.computerworld.com/s/article/9230152/Android_4.0_The_ultimate_guide_plus_cheat_sheet_.

Why Use the Emulator

First of all, it's free. The Android SDK and its third-party add-ons cost absolutely nothing and allow developers to emulate devices that they do not own and may not have access to. This is important, as not all phones acquire the latest Android OS versions via

over-the-air (OTA) updates, and it may not be feasible for developers to purchase every device that is expected to support their software package(s).

- *Development and testing.* Developers can use the SDK to create several Android Virtual Device (AVD) configurations for development and testing purposes. Each AVD can have varying screen dimensions, SD card sizes, or even versions of the Android SDK (which is useful for testing backward compatibility with prior Android builds).
- *Playing with a new build.* The emulator allows developers to just have fun with a new build of Android and learn more about it.

Let's now walk through the steps required to build and run an emulator image on an x86 build. For simplicity's sake, this section focuses on the Gingerbread emulator, although most of this content applies to the Ice Cream Sandwich emulator as well.

Building an Emulator Image

The first step is to follow the setup instructions listed here: http://source.android.com/source/initializing.html .

Google has made Gingerbread documentation available at http://source.android.com/source/downloading.html.

Currently, the default repo initialization instructions are only for getting the latest master branch or a specific branch, such as when using the command **android-2.3.7_r1**. The repo and branch instructions for building Gingerbread x86 emulator images were not given.

Use the branch labeled **gingerbread** to download the latest Android source branch, as shown in Listing 6-1. This branch has the required build files and modules to build the x86 SDK emulator images.

Listing 6-1. Set Up the Repository

```
$ repo init –u
https://android.googlesource.com/platform/manifest -b gingerbread
$ repo sync
```

Initialize the build environment, as shown in Listing 6-2.

Listing 6-2. Build Environment Initialization

```
$ source build/envsetup.sh
including device/htc/passion/vendorsetup.sh
including device/samsung/crespo4g/vendorsetup.sh
including device/samsung/crespo/vendorsetup.sh
```

Use the **lunch** command to configure the build for building the x86 SDK, as shown in Listing 6-3. This will build the emulator images for x86, along with other SDK modules.

Listing 6-3. Lunch Command

```
$ lunch sdk_x86-eng
==================================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=2.3.7
TARGET_PRODUCT=sdk_x86
TARGET_BUILD_VARIANT=eng
TARGET_SIMULATOR=false
TARGET_BUILD_TYPE=release
TARGET_BUILD_APPS=
TARGET_ARCH=x86
TARGET_ARCH_VARIANT=x86_atom
HOST_ARCH=x86
HOST_OS=linux
HOST_BUILD_TYPE=release
BUILD_ID=GINGERBREAD
==================================================
```

Finally, issue the **`make sdk`** command in order to kick off the build, as shown in Listing 6-4.

Listing 6-4. Make SDK

```
$ make sdk
==================================================
PLATFORM_VERSION_CODENAME=REL
PLATFORM_VERSION=2.3.7
TARGET_PRODUCT=sdk_x86
TARGET_BUILD_VARIANT=eng
TARGET_SIMULATOR=false
TARGET_BUILD_TYPE=release
TARGET_BUILD_APPS=
TARGET_ARCH=x86
TARGET_ARCH_VARIANT=x86_atom
HOST_ARCH=x86
HOST_OS=linux
HOST_BUILD_TYPE=release
BUILD_ID=GINGERBREAD
==================================================
```

Additionally, you can specify the number of parallel jobs for **`make`** using the **`-j`** parameter, to speed up the build process. (It is recommended that you use a value that is at least greater than or equal to the total number of CPUs in the system.)

The build will create three images, including the QEMU Android kernel **`image:t system.img`** (Android), **`userdata.img`**, and **`ramdisk.img`**.

When the build completes, the images will be in the **build** folder **$ANDROID_BUILD_TOP/out/target/product/generic_x86/**, as shown in Figure 6-6.

```
$ ls $ANDROID_BUILD_TOP/out/target/product/generic_x86/ -Altr
```

```
total 261476
-rw-r--r--  1 user user      4129 2011-10-31 20:16 clean_steps.mk
-rw-r--r--  1 user user       406 2011-10-31 20:16 previous_build_config.mk
drwxr-xr-x  2 user user      4096 2011-10-31 20:20 sdk
drwxr-xr-x  3 user user      4096 2011-10-31 20:20 data
drwxr-xr-x  2 user user      4096 2011-10-31 20:24 grub
drwxr-xr-x  4 user user      4096 2011-10-31 20:24 symbols
drwxr-xr-x  8 user user      4096 2011-10-31 20:24 root
-rw-r--r--  1 user user    212904 2011-10-31 20:24 ramdisk.img
drwxr-xr-x  3 user user      4096 2011-10-31 20:25 dex_bootjars
drwxr-xr-x 12 user user      4096 2011-10-31 20:27 system
-rw-------  1 user user   4587264 2011-10-31 20:27 userdata.img
-rw-------  1 user user 131539584 2011-10-31 20:28 system.img
drwxr-xr-x 13 user user      4096 2011-10-31 20:28 obj
-rw-r--r--  1 user user 131362140 2011-10-31 20:28 sdk_x86-symbols-eng.user.zip
```

Figure 6-6. *Image Location*

The Android kernel image for QEMU (**kernel-qemu**) comes with Android Sources. It is located under the prebuilt folder (**$ANDROID_BUILD_TOP/prebuilt/android-x86/kernel**), as shown in Figure 6-7.

```
$ ls $ANDROID_BUILD_TOP/prebuilt/android-x86/kernel -Altr
```

```
total 86332
-rw-r--r-- 1 user user    15872 2011-10-27 21:31 LINUX_KERNEL_COPYING
-rw-r--r-- 1 user user      589 2011-10-28 03:01 README
-rw-r--r-- 1 user user  2471824 2011-10-28 03:01 kernel-qemu
-rw-r--r-- 1 user user  2512912 2011-10-28 03:01 kernel-vbox
-rwxr-xr-x 1 user user  5930056 2011-10-28 03:01 vmlinux-qemu
-rwxr-xr-x 1 user user 77463163 2011-10-28 03:01 vmlinux-vbox
```

Figure 6-7. *Kernel Image*

You now have all the image files required for running an x86 Android Gingerbread image on the Android x86 emulator. You need to set up the image files with the SDK, which is covered in next section.

Setting Up the SDK to Use x86 Emulator Images

The Android SDK tools (Android and the AVD manager) expect the x86 emulator images to be present in the default SDK folders for platform images, which is **/platforms/android-10/images**.

The images that follow assume the **$ANDROID_SDK_TOP** environment variable is set to the location of the Android SDK installation folder.

As in Figure 6-8, Android-10 comes with emulator images for ARM by default. In order to set up the x86 emulator images in the SDK, you need to create an **x86** folder and copy the images you build into that folder. You can also move the ARM images to their own folder, as shown in Listing 6-5.

```
$ cd $ANDROID_SDK_TOP/platforms/android-10/images/
$ls -l
```

```
total 97012
-rw-r--r-- 1 user user  4112064 2011-11-01 09:56 userdata.img
-rw-r--r-- 1 user user   146641 2011-11-01 09:56 ramdisk.img
-rwxr-xr-x 1 user user  1466272 2011-11-01 09:56 kernel-qemu
-rw-r--r-- 1 user user 93282816 2011-11-01 09:56 system.img
-rw-r--r-- 1 user user   322846 2011-11-01 09:56 NOTICE.txt
```

Figure 6-8. *Image Location*

Listing 6-5. ARM Folder

```
$ mkdir arm
$ mv *.img kernel-qemu arm/
```

Listing 6-6 shows instructions for the x86 folder.

Listing 6-6. x86 Instructions

```
$ mkdir x86

$ cp $ANDROID_BUILD_TOP/out/target/product/generic_x86/*img x86/
$ cp $ANDROID_BUILD_TOP/prebuilt/android-x86/kernel/kernel-qemu x86/

$ cp NOTICE.txt arm/
$ cp NOTICE.txt x86/
```

The final images folder of the Android-10 platform is shown in Figure 6-9.

```
$ ls -l *
```

```
arm:
total 97012
-rwxr-xr-x 1 user user  1466272 2011-11-01 09:56 kernel-qemu
-rw-r--r-- 1 user user   322846 2011-11-01 10:03 NOTICE.txt
-rw-r--r-- 1 user user   146641 2011-11-01 09:56 ramdisk.img
-rw-r--r-- 1 user user 93282816 2011-11-01 09:56 system.img
-rw-r--r-- 1 user user  4112064 2011-11-01 09:56 userdata.img

x86:
total 135880
-rwxr-xr-x 1 user user   2471824 2011-11-01 10:11 kernel-qemu
-rw-r--r-- 1 user user    322846 2011-11-01 10:03 NOTICE.txt
-rwxr-xr-x 1 user user    212904 2011-11-01 10:10 ramdisk.img
-rwxr-xr-x 1 user user 131539584 2011-11-01 10:10 system.img
-rwxr-xr-x 1 user user   4587264 2011-11-01 10:10 userdata.img
```

Figure 6-9. *Final Images Folder*

Before using the Gingerbread Intel architecture images with the x86 emulator, you must create an AVD configuration that specifies the required software and hardware customizations. More detailed information about AVDs can be found next, and in the Intel Software Network article titled "Android Virtual Device Emulation for Intel Architecture" (http://software.intel.com/en-us/articles/android-virtual-device-emulation-for-intel-architecture).

At this point, your emulator and x86 image are ready to use. Note that the emulator performance will see drastic improvements if you use Intel Hardware Acceleration Execution Manager (Intel HAXM) with this system image—without it, performance may vary. (Intel HAXM requires an Intel processor with Intel VT-x support. For more information about Intel HAXM, see **Chapter 11: Using Intel® Hardware Accelerated Execution Manager to Speed-up Android on x86 Emulation**.

It is now time to use the Gingerbread x86 emulator to emulate an x86 image.

Open the Android tool or bring up the AVD creation tool directly from Eclipse.

Figures 6-10 and 6-11 show the creation of an AVD for Gingerbread with an Intel Atom (x86) CPU.

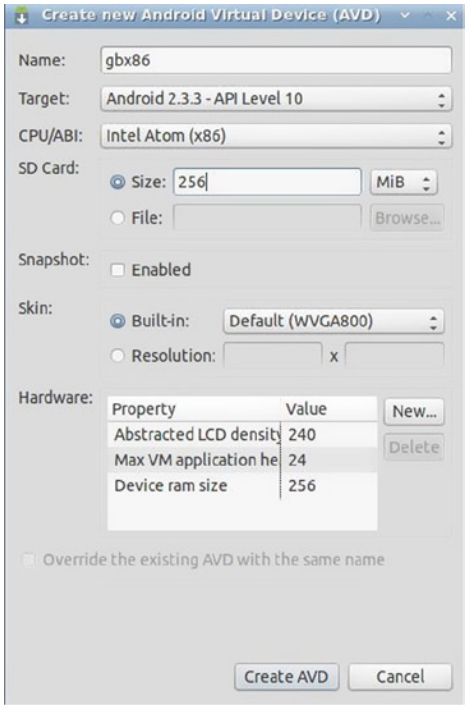

Figure 6-10. *New AVD Creation*

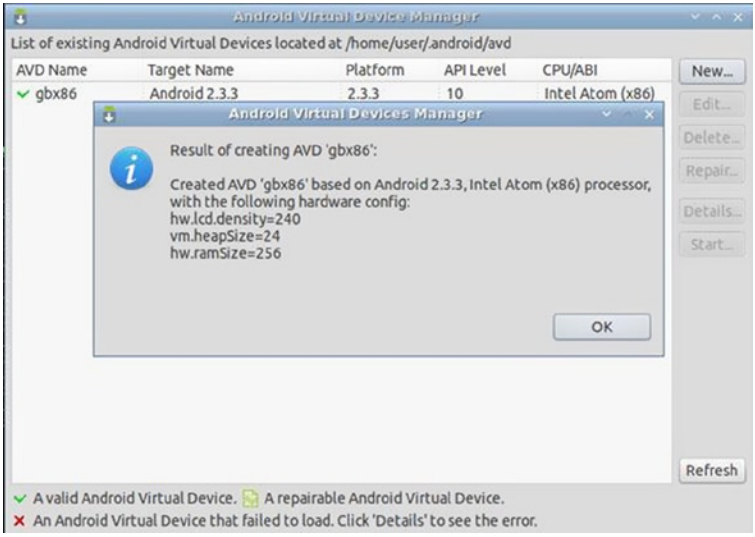

Figure 6-11. *Success Dialog Box*

Test the x86 Gingerbread AVD by selecting the AVD and clicking on Start, as shown in Figure 6-12.

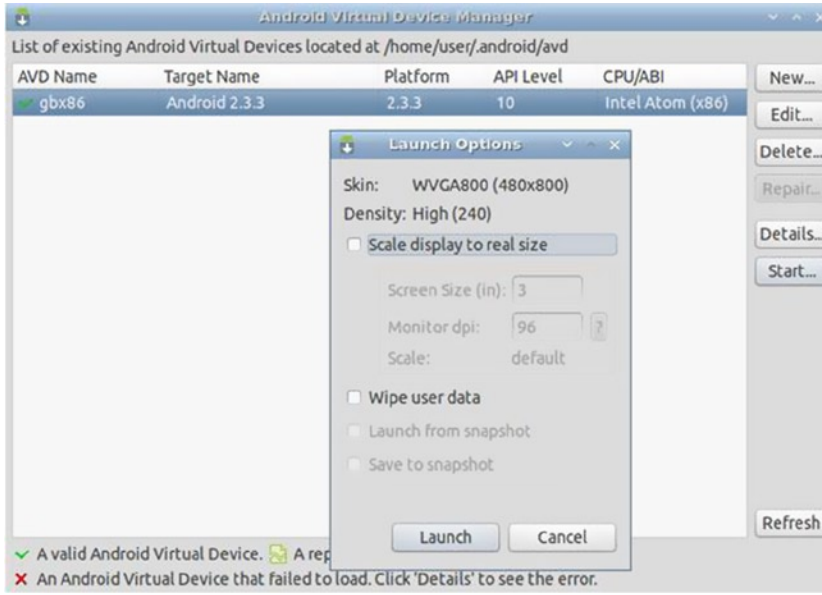

Figure 6-12. *Launch Options*

Figure 6-13 shows the home screen of the Gingerbread for Intel Atom (x86) on emulator x86.

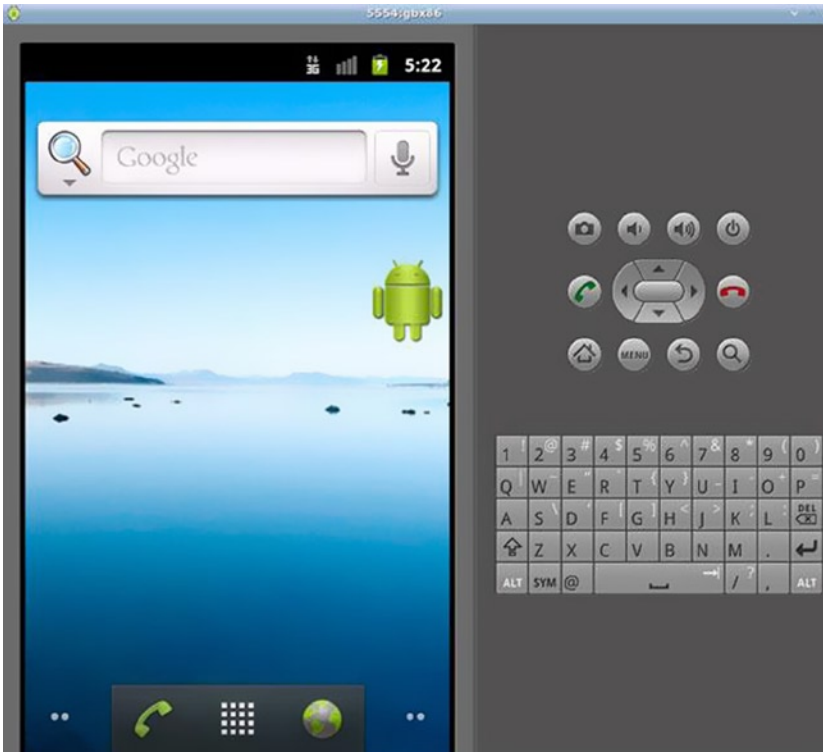

Figure 6-13. *Home Screen*

It is recommended that you use the x86 emulator with Intel VT hardware acceleration. On Linux, you can do this using the Linux KVM. Ubuntu has further documentation on how to configure and use it at https://help.ubuntu.com/community/KVM.

Listing 6-7. KVM

```
$ emulator-x86 -avd gbx86 -qemu -m 512 -enable-kvm
```

With KVM, shown in Listing 6-7 (**`-enable-kvm`**), users will likely notice performance boosts during the Android boot, along with quicker emulator responsiveness.

Key Gingerbread Features

This next section highlights a few of the key features that Gingerbread supports. These features are also supported in the latest release of Android, with full plans to have continued support.

Battery Usage Stats

In About Phone, as seen in Figure 6-14 there is a Battery Use section. Battery stats will vary based on the emulation device and build. In the case where the developer can **telnet** into the emulated device, some simple commands can be used to mock battery drain. Take a look at one such example at http://android-er.blogspot.com/2010/09/how-to-set-battery-status-of-android.html.

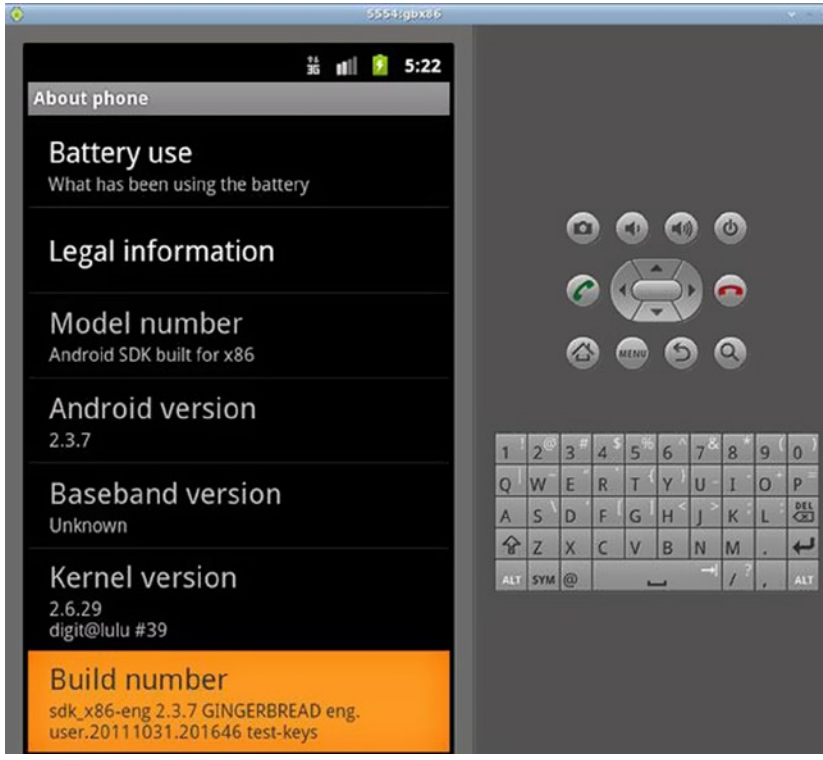

Figure 6-14. *About Phone View*

Task Manager

From Settings ➤ Applications ➤ Running Services, choose the Running tab. It shows what's currently running, as you can see in Figure 6-15.

Figure 6-15. *The Task Manager's Running Tab*

For example, the developer can close the Settings process by clicking on the item and selecting Stop.

Cut and Paste Text

Opening the Messaging application and choosing New Message allows you to type a message as if you're sending an SMS. By clicking in the text field and typing on the host keyboard, characters appear seamlessly on the Android screen. After typing **`Hello 2.3.5!`**, the screen looks like Figure 6-16.

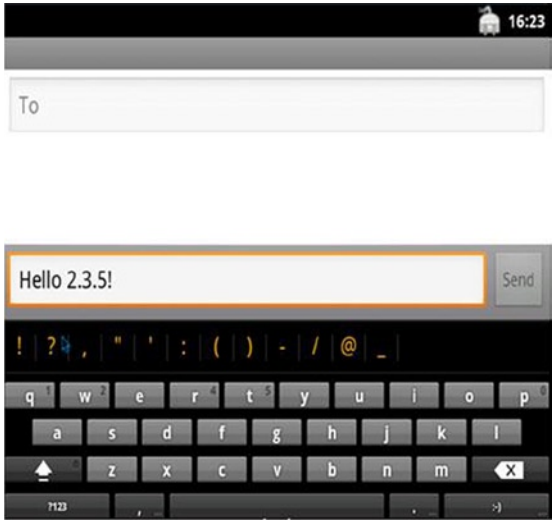

Figure 6-16. *Messaging Fun*

If you dragged the mouse into the text field where **`Hello 2.3.5!`** is found and then held down the mouse button (or touchpad button) for about two seconds, a tooltip menu appears to Edit Text. If you choose Select All and then repeat the mouse operation, you can cut the text. After cutting the text, you can repeat the mouse operation again to paste the text elsewhere.

Ice Cream Sandwich Emulation

The x86 Android 4.0.4 emulator system image enables you to run an emulation of Android Ice Cream Sandwich on your development machine. In combination with the Android SDK, you can test your Android applications on a virtual Android device based on Intel architecture (x86).

In order to install the emulator system image, you can use the Android SDK Manager (recommended method), or you can download the binary ZIP file and unzip and copy the included directory into the **`add-ons`** directory of your Android SDK installation. (Note that this method does not allow for automatic updates of the add-on.)

The following section provides a guide for ICS image installation.

Prerequisites

The Android x86 emulator image requires the Android SDK to be installed. For instructions on installing and configuring the Android SDK, refer to the Android developer website (see `http://developer.android.com/sdk/`).

■ **Note** The x86 emulator image for Android can be accelerated using Intel Hardware Accelerated Execution Manager (Intel HAXM). For more information, refer to **Chapter 11: Using Intel® Hardware Accelerated Execution Manager to Speed-up Android on x86 Emulation**.

Downloading Through the Android SDK Manager

1. Start the Android SDK Manager.
2. Under Android 4.0.4 (some screenshots may refer to older versions), select Intel x86 Atom System Image, as shown in Figure 6-17.

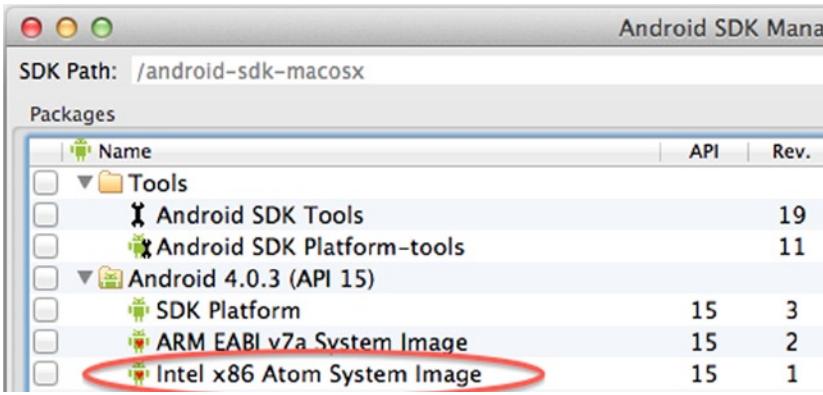

Figure 6-17. *Intel x86 Atom System Image*

3. Once it's selected, click the Install Package button.
4. Review the Intel Corporation license agreement. If you accept the terms, select Accept and click Install.
5. The SDK Manager will download and extract the system image to the appropriate location within the Android SDK directory.

Using the System Image

1. Start the Android AVD Manager and create a new AVD, setting Target to Android 4.0.X, and CPU/ABI to Intel Atom (x86), as shown in Figure 6-18.

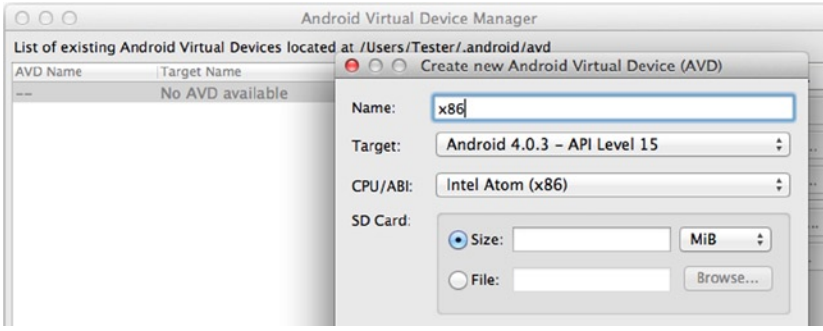

Figure 6-18. *Setting the Target*

■ **Note** If the Intel Atom (x86) CPU/ABI option is not available, make sure that the system image is installed correctly.

2. Click the Create AVD button.
3. The AVD has been successfully created and is now ready to use, as shown in Figure 6-19.

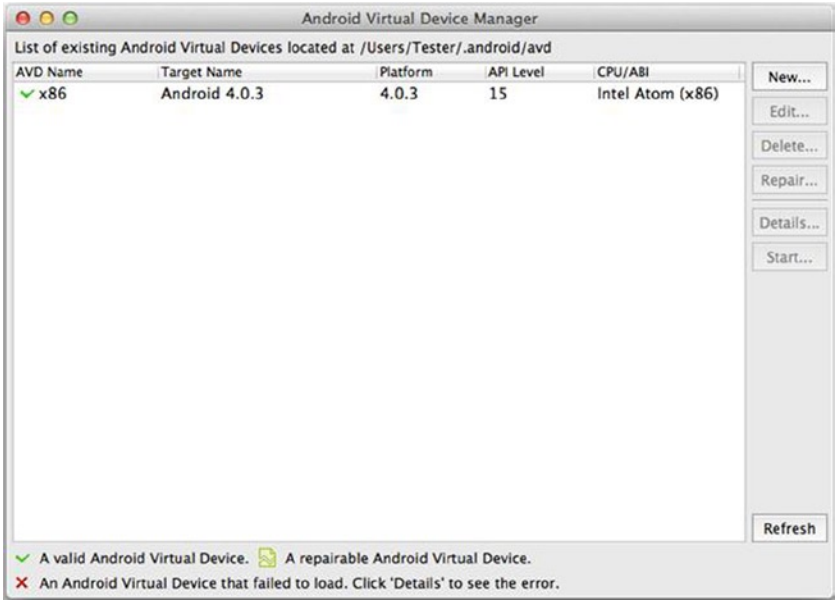

Figure 6-19. *Image Ready*

Downloading Manually

1. Go to `http://www.intel.com/software/android`.
2. Download the Intel x86 Atom System Image (found under the Tools and Downloads tab).
3. Navigate to the directory containing the Android SDK, as shown in Figure 6-20.

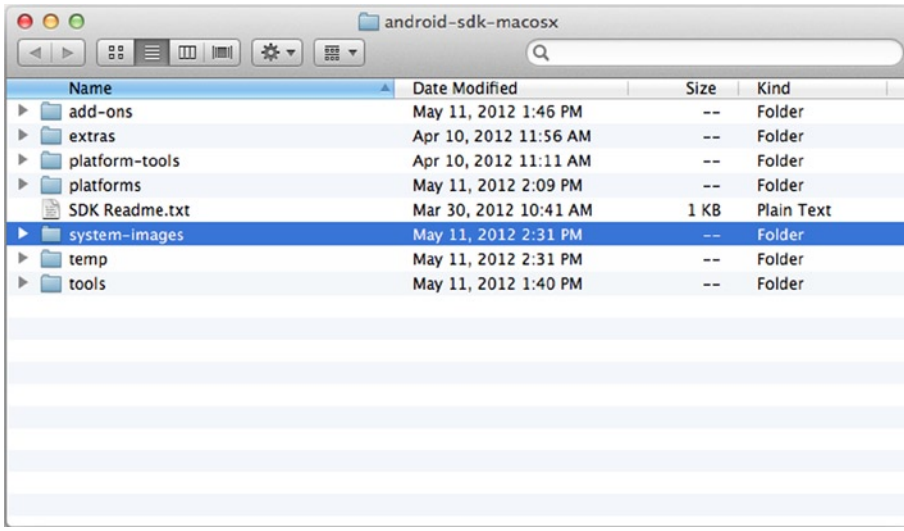

Figure 6-20. *Android SDK Directory*

4. The **`system-images`** directory contains Android's system images, separated by architecture, as shown in Figure 6-21.

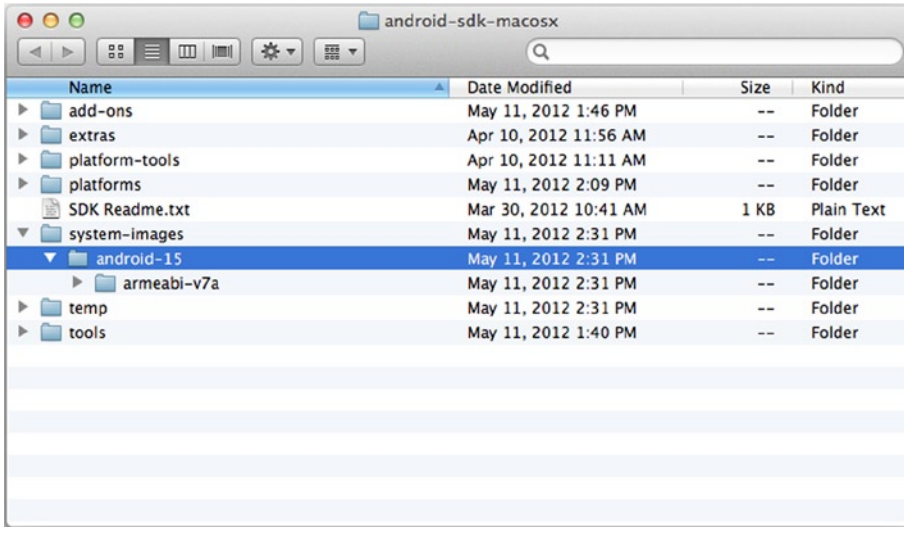

Figure 6-21. *Separated Images*

5. Expand **`android-15`** (this directory contains API level 15 system images), as shown in Figure 6-22.

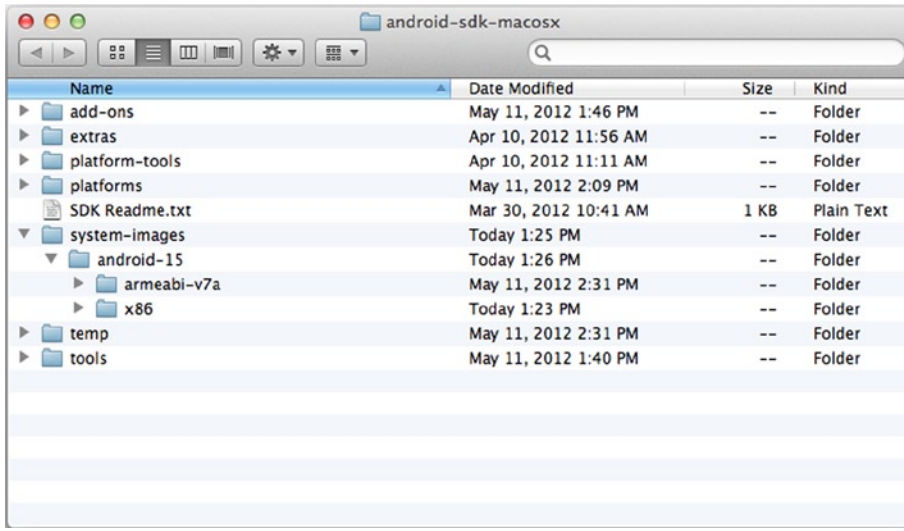

Figure 6-22. *API Level 15*

6. Extract the **x86** directory contained in the downloaded system image archive directly into the **android-15** directory.

7. The directory structure should now look like Figure 6-23.

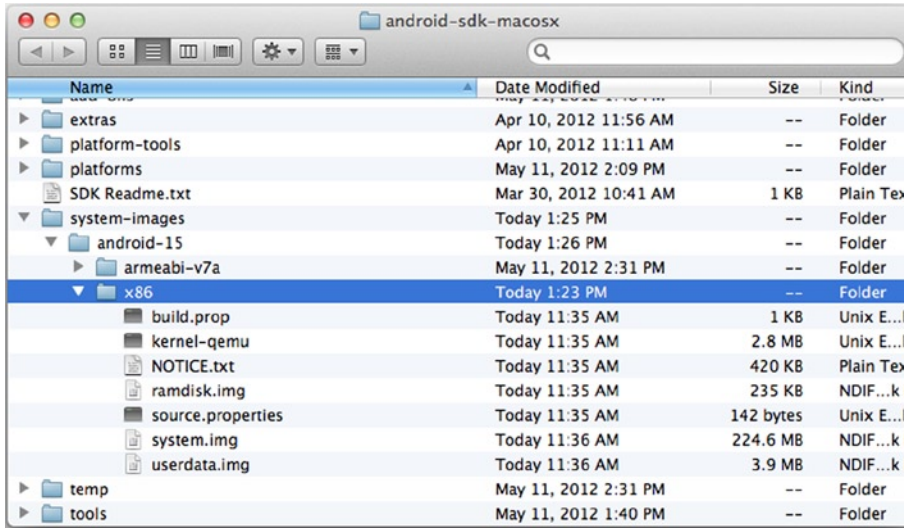

Figure 6-23. *Expected Directory Structure*

8. The system image is now installed and ready to be used.

CPU Acceleration

You can improve the performance of Intel Atom x86 Image for Android Ice Cream Sandwich using hardware-based virtualization with Intel VT-x technology. If your computer has an Intel processor with VT-x support, it is recommended that you use Intel HAXM with this system image. For more information about Intel HAXM, visit http://int-software.intel.com/en-us/android.

■ **Note** Intel HAXM is for Windows and OS X operating systems only. For Linux hosts, you can use Kernel-based Virtual Machine (KVM) to accelerate emulation performance. For information on installing and configuring KVM on Ubuntu, refer to the following guide at https://help.ubuntu.com/community/KVM/Installation.

GPU Acceleration

The Intel Atom x86 image for Android Ice Cream Sandwich can make use of hardware GPU features to increase the performance of games, graphics-intensive programs, and user interface elements.

■ **Note** The functionality and performance of GPU acceleration is highly dependent on your computer's graphics card and graphics drivers.

To use hardware GPU acceleration, perform the following steps:

1. Open the Android AVD Manager.
2. Select the AVD and click Edit.
3. The AVD editor window will appear. In the Hardware section, click New, as shown in Figure 6-24.

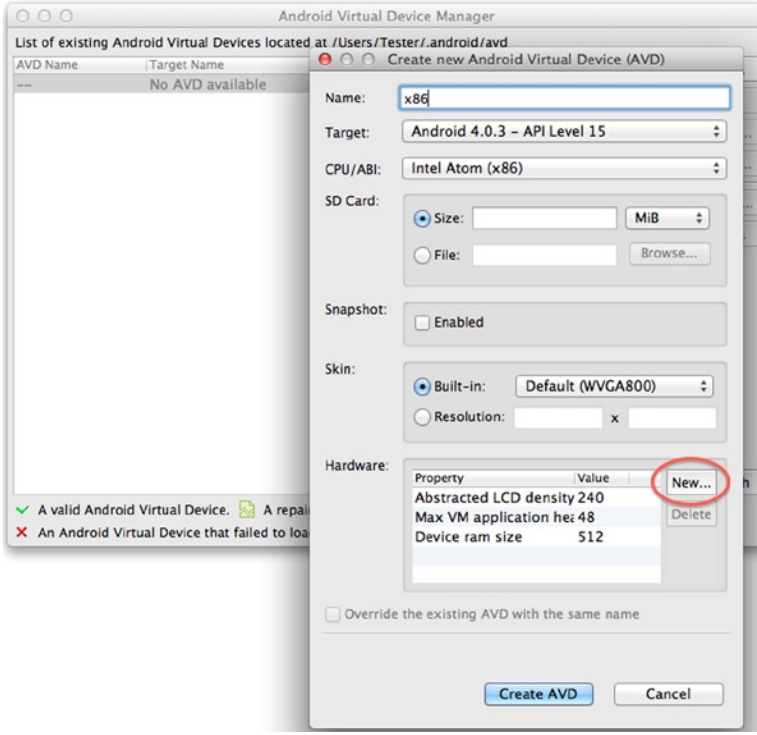

Figure 6-24. *Hardware Section*

4. In the Property drop-down box, select GPU Emulation, as shown in Figure 6-25.

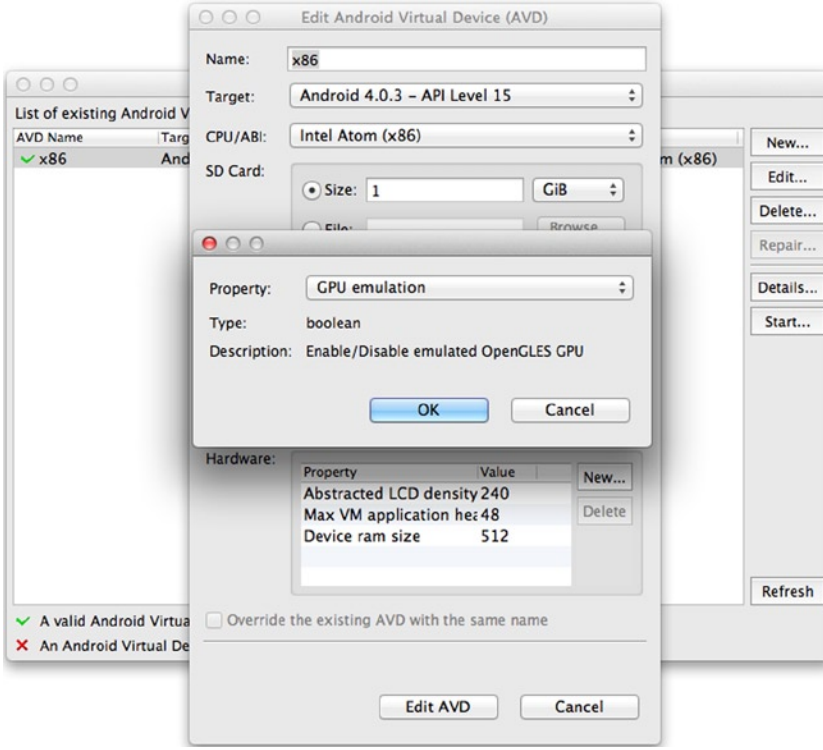

Figure 6-25. *GPU Emulation Option*

5. Click OK.
6. After the GPU Emulation property has been added, change the Value to Yes, as shown in Figure 6-26.

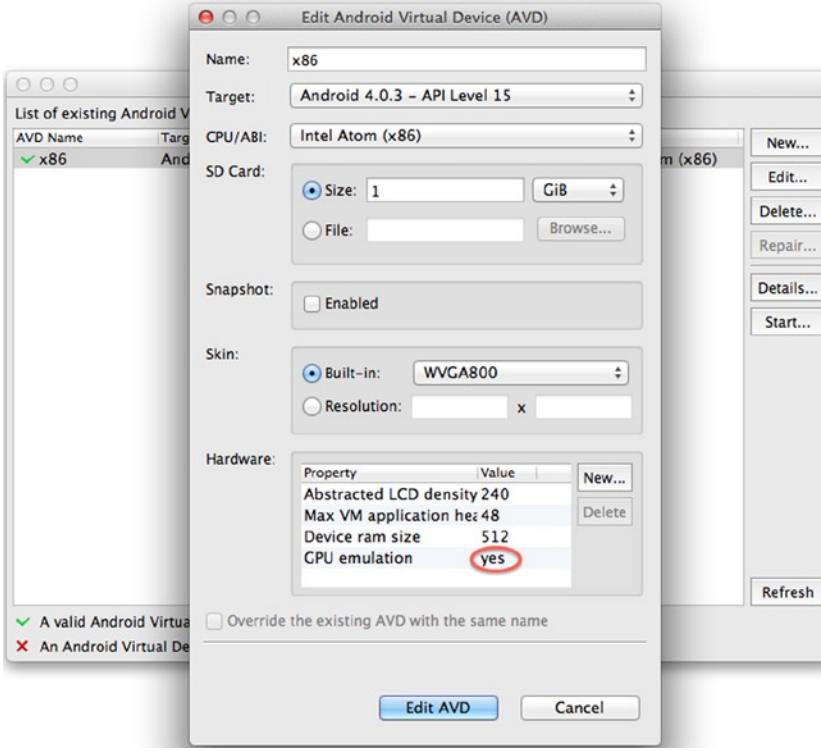

Figure 6-26. *Value Changed to Yes*

7. Click Edit AVD to save the AVD.
8. After the AVD has been modified, a dialog box will appear confirming the AVD settings, shown in Figure 6-27.

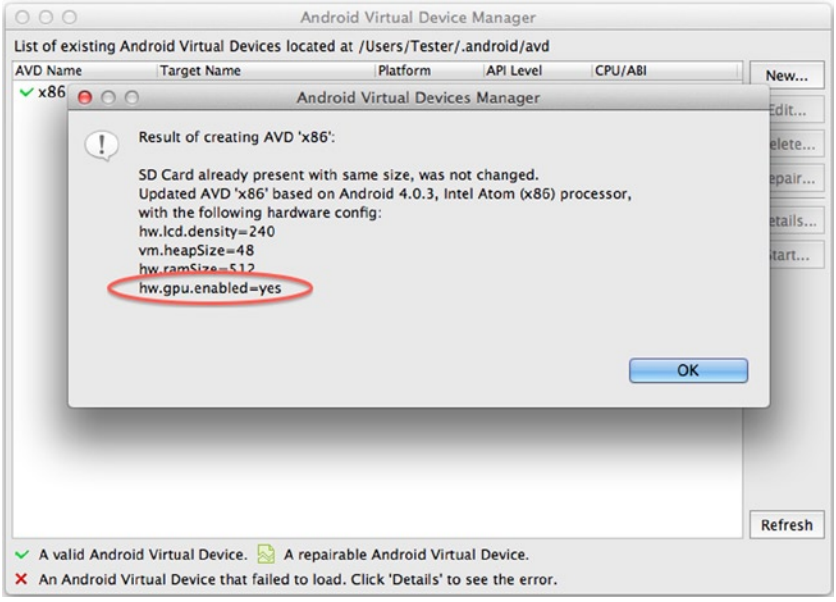

Figure 6-27. *Confirmation Dialog Box*

In the confirmation dialog box, the **`hw.gpu.enabled=yes`** line indicates that GPU acceleration is enabled for that particular AVD.

■ **Note** The GPU acceleration must be enabled on a per-AVD basis.

Overview

In this chapter, you set up a fully functioning Android development environment. You also installed the prerequisites for Android and the SDK. The chapter discussed the Android emulator in detail, and you created an x86 emulator for easy testing. You even created a fully functioning Android 4.0.4 (Ice Cream Sandwich) emulator on a virtual x86 platform, for testing the latest features of Android. In the next chapter, you will learn how to install and use the Android Native Development Kit in order to create and port applications for the Intel platform.