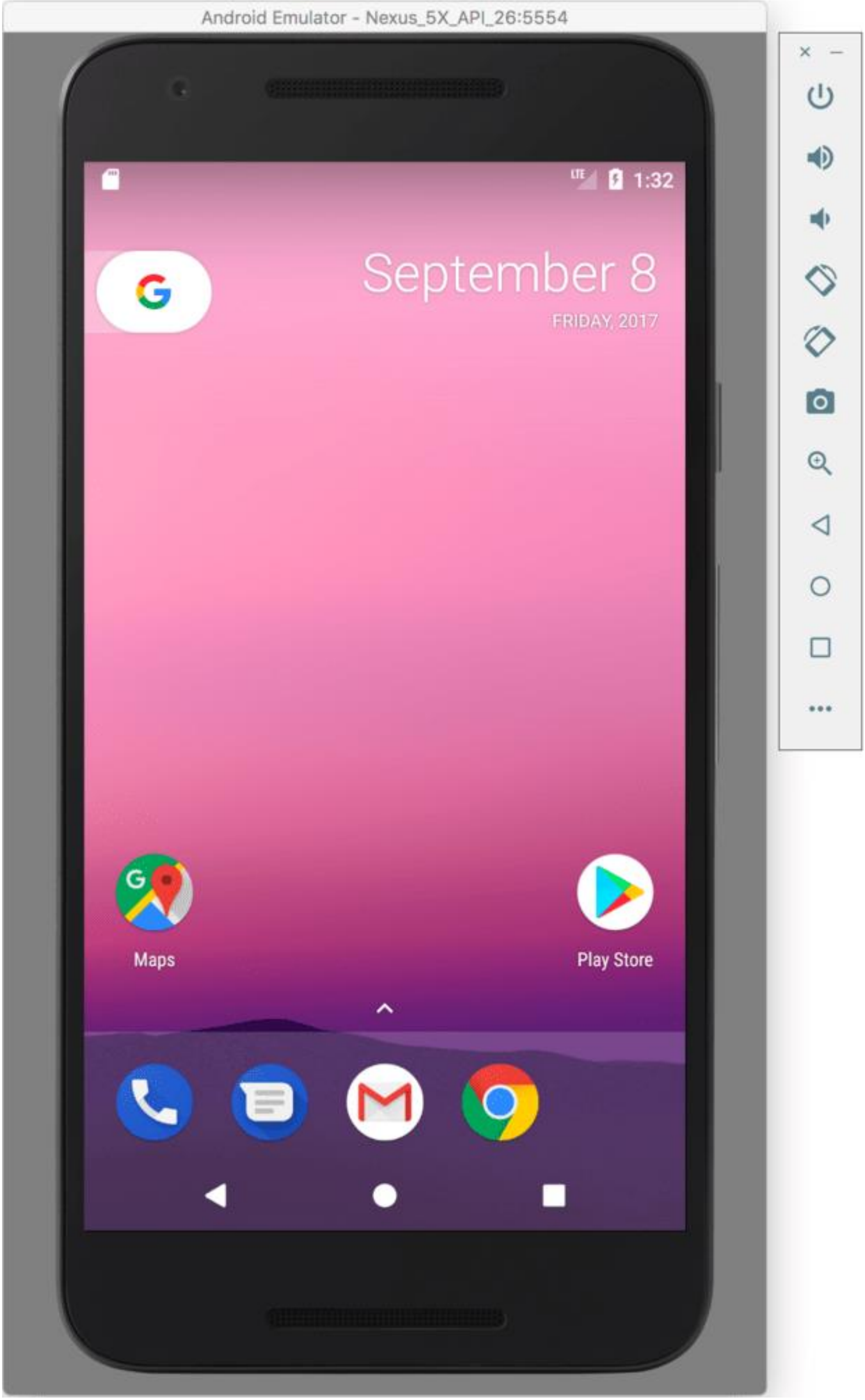
About Android Virtual Devices

AVDs are essentially emulators that allow Android applications to be tested without the necessity to install the application on a physical Android based device. An AVD may be configured to emulate a variety of hardware features including options such as screen size, memory capacity and the presence or otherwise of features such as a camera, GPS navigation support or an accelerometer. As part of the standard Android Studio installation, a number of emulator templates are installed allowing AVDs to be configured for a range of different devices. Additional templates may be loaded or custom configurations created to match any physical Android device by specifying properties such as processor type, memory capacity and the size and pixel density of the screen. Check the online developer documentation for your device to find out if emulator definitions are available for download and installation into the AVD environment.

When launched, an AVD will appear as a window containing an emulated Android device environment. Figure 4-1, for example, shows an AVD session configured to emulate the Google Nexus 5X model.

New AVDs are created and managed using the Android Virtual Device Manager, which may be used either in command-line mode or with a more user-friendly graphical user interface.

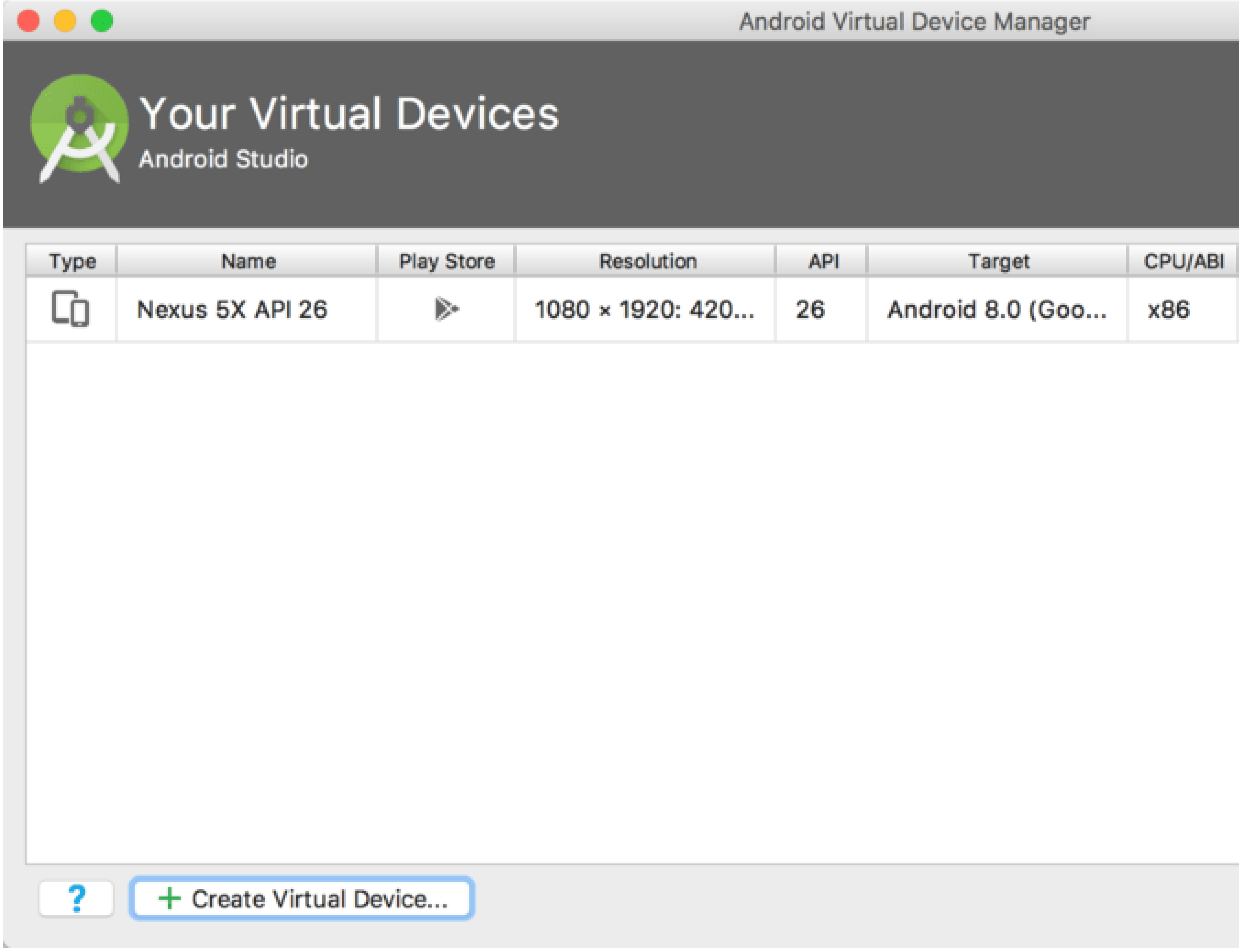

Android Emulator - Nexus_5X_API_26:5554
1:32
September 8
FRIDAY, 2017
Maps
Play Store

Creating a New AVD

In order to test the behavior of an application in the absence of a physical device, it will be necessary to create an AVD for a specific Android device configuration.

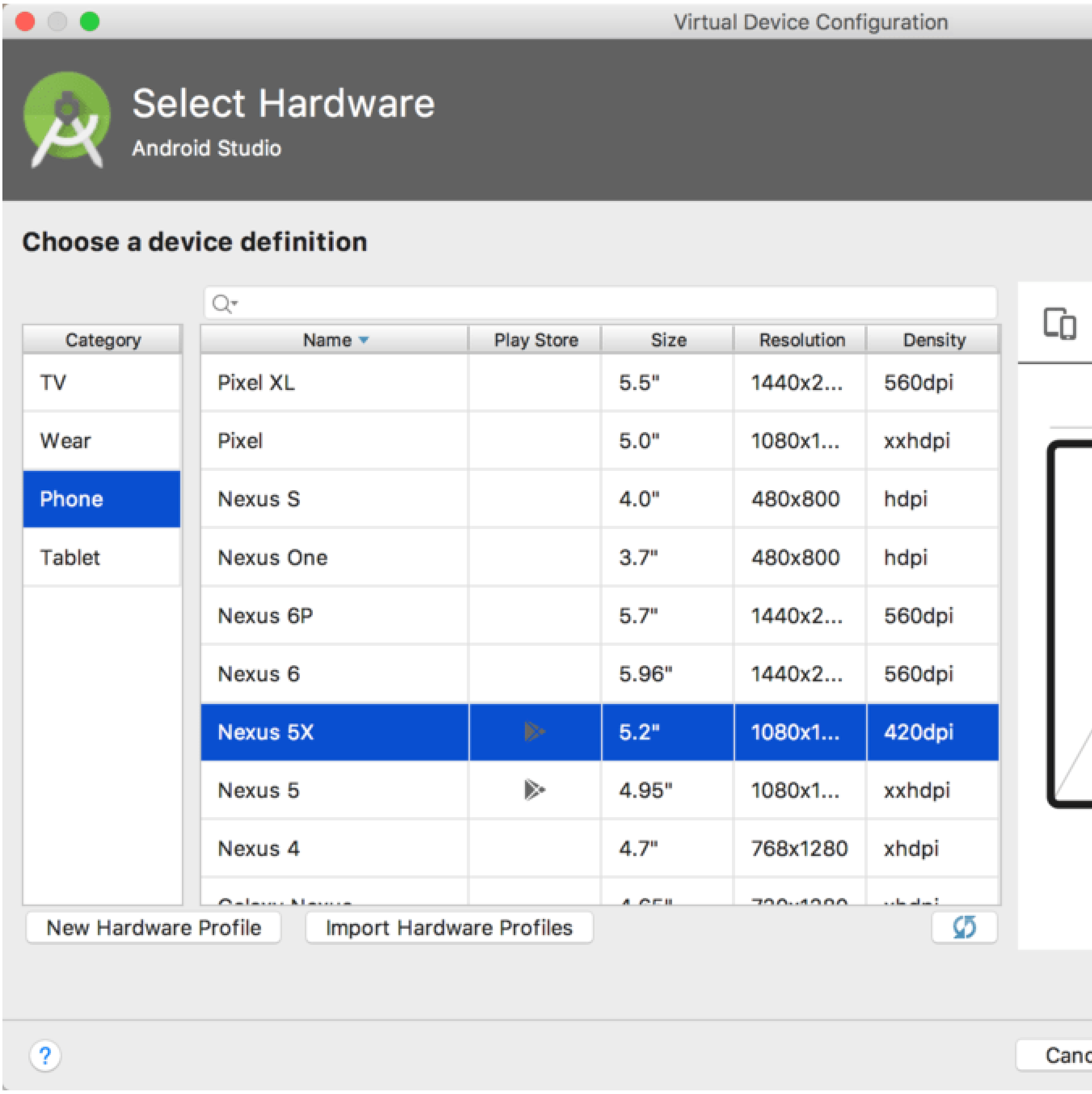
To create a new AVD, the first step is to launch the AVD Manager. This can be achieved from within the Android Studio environment by selecting the Tools -> Android -> AVD Manager menu option from within the main window.

Once launched, the tool will appear as outlined in Figure 4-2 if existing AVD instances have been created:



Type	Name	Play Store	Resolution	API	Target	CPU/ABI
	Nexus 5X API 26	▶	1080 × 1920: 420...	26	Android 8.0 (Goo...	x86

To add an additional AVD, begin by clicking on the Create Virtual Device button in order to invoke the Virtual Device Configuration dialog:



Virtual Device Configuration

Select Hardware

Android Studio

Choose a device definition

Category
TV
Wear
Phone
Tablet

Name ▾	Play Store	Size	Resolution	Density
Pixel XL		5.5"	1440x2...	560dpi
Pixel		5.0"	1080x1...	xxhdpi
Nexus S		4.0"	480x800	hdpi
Nexus One		3.7"	480x800	hdpi
Nexus 6P		5.7"	1440x2...	560dpi
Nexus 6		5.96"	1440x2...	560dpi
Nexus 5X	▶	5.2"	1080x1...	420dpi
Nexus 5	▶	4.95"	1080x1...	xxhdpi
Nexus 4		4.7"	768x1280	xhdpi
Galaxy Nexus		4.65"	720x1280	xhdpi

Within the dialog, perform the following steps to create a Nexus 5X compatible emulator:

From the Category panel, select the Phone option to display the list of available Android tablet AVD templates.

Select the Nexus 5X device option and click Next.

On the System Image screen, select the latest version of Android (at time of writing this is API level 28, Android 9.0 with Google Play) for the x86 ABI. Note that if the system image has not yet been installed a Download link will be provided next to the Release Name. Click this link to download and install the system image before selecting it. If the image you need is not listed, click on the x86 images and Other images tabs to view alternative lists.

Click Next to proceed and enter a descriptive name (for example Nexus 5X API 28) into the name field or simply accept the default name.

Click Finish to create the AVD.

With the AVD created, the AVD Manager may now be closed. If future modifications to the AVD are necessary, simply re-open the AVD Manager, select the AVD from the list and click on the pencil icon in the Actions column of the device row in the AVD Manager.

Starting the Emulator

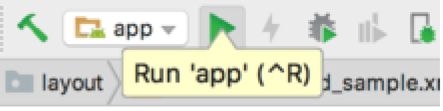
To perform a test run of the newly created AVD emulator, simply select the emulator from the AVD Manager and click on the launch button (the green triangle in the Actions column). The emulator will appear in a new window and begin the startup process. The amount of time it takes for the emulator to start will depend on the configuration of both the AVD and the system on which it is running.

Although the emulator probably defaulted to appearing in portrait orientation, this and other default options can be changed. Within the AVD Manager, select the new Nexus 5X entry and click on the pencil icon in the Actions column of the device row. In the configuration screen locate the Startup and orientation section and change the orientation setting. Exit and restart the emulator session to see this change take effect. More details on the emulator are covered in the next chapter ("Using and Configuring the Android Studio AVD Emulator").

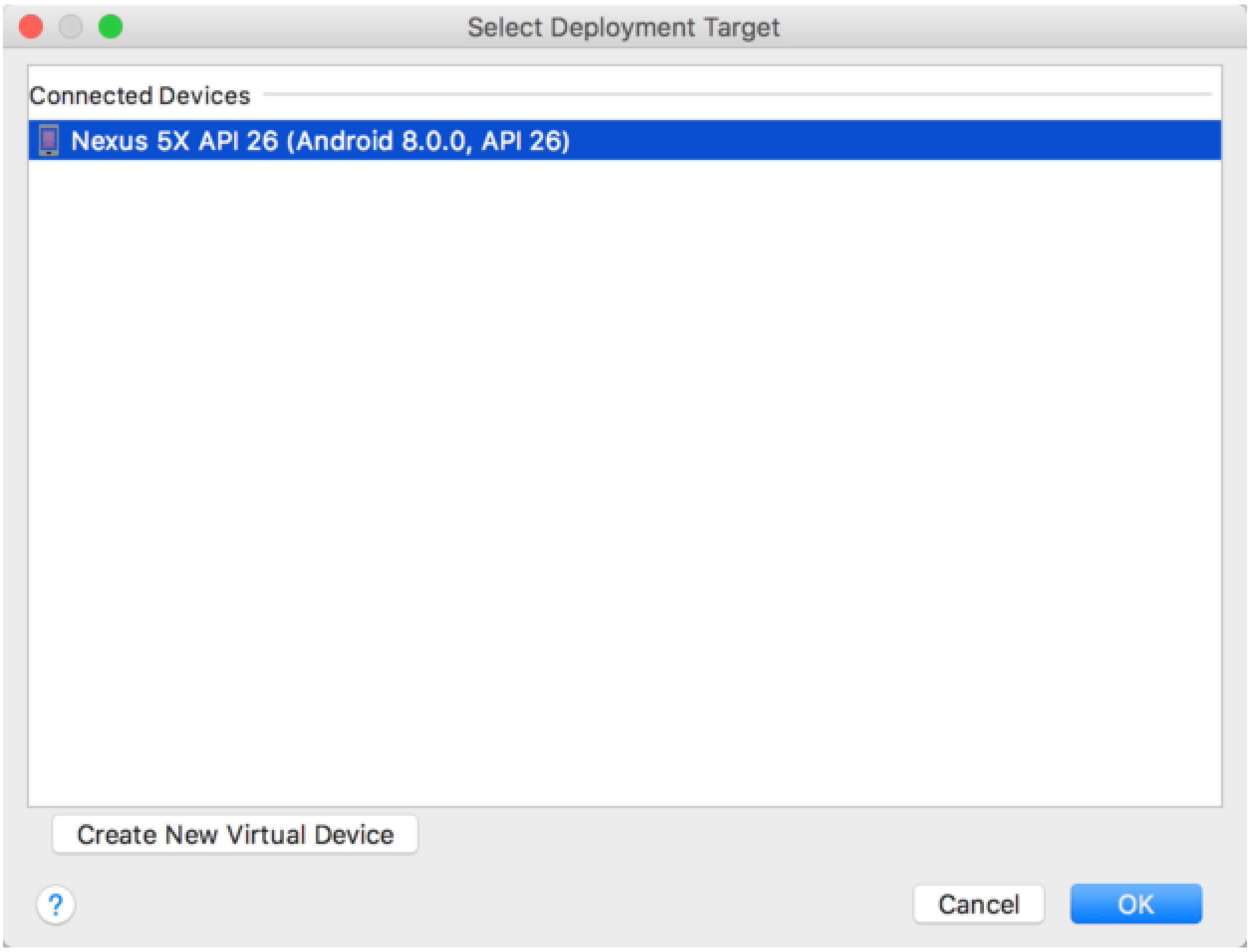
To save time in the next section of this chapter, leave the emulator running before proceeding.

Running the Application in the AVD

With an AVD emulator configured, the example AndroidSample application created in the earlier chapter now can be compiled and run. With the AndroidSample project loaded into Android Studio, simply click on the run button represented by a green triangle located in the Android Studio toolbar as shown in Figure 4-4 below, select the Run -> Run 'app' menu option or use the Ctrl-R keyboard shortcut:



By default, Android Studio will respond to the run request by displaying the Select Deployment Target dialog. This provides the option to execute the application on an AVD instance that is already running, or to launch a new AVD session specifically for this application. Figure 4-5 lists the previously created Nexus 5X AVD as a running device as a result of the steps performed in the preceding section. With this device selected in the dialog, click on OK to install and run the application on the emulator.

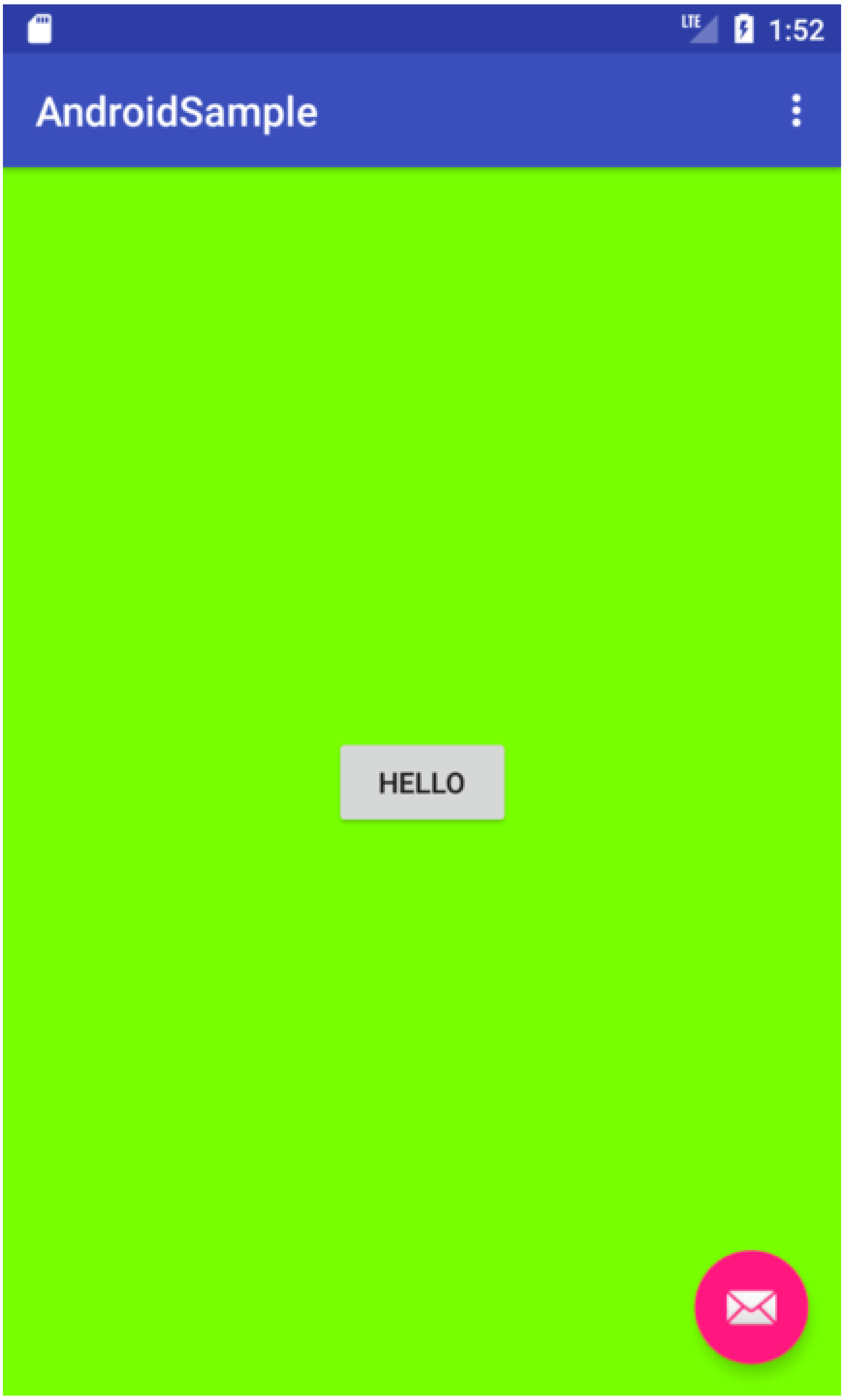


Select Deployment Target

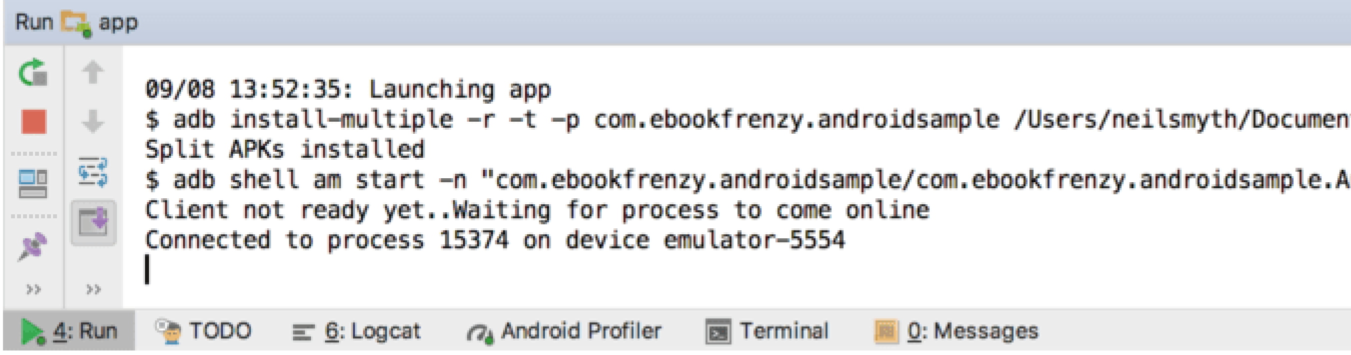
Connected Devices

Nexus 5X API 26 (Android 8.0.0, API 26)

Once the application is installed and running, the user interface for the AndroidSampleActivity class will appear within the emulator:



In the event that the activity does not automatically launch, check to see if the launch icon has appeared among the apps on the emulator. If it has, simply click on it to launch the application. Once the run process begins, the Run and Logcat tool windows will become available. The Run tool window will display diagnostic information as the application package is installed and launched. Figure 4-7 shows the Run tool window output from a successful application launch:



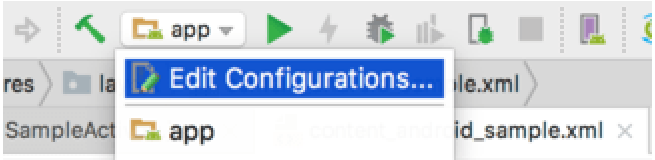
```
09/08 13:52:35: Launching app
$ adb install-multiple -r -t -p com.ebookfrenzy.androidsample /Users/neilsmyth/Documen
Split APKs installed
$ adb shell am start -n "com.ebookfrenzy.androidsample/com.ebookfrenzy.androidsample.A
Client not ready yet..Waiting for process to come online
Connected to process 15374 on device emulator-5554
```

If problems are encountered during the launch process, the Run tool window will provide information that will hopefully help to isolate the cause of the problem.

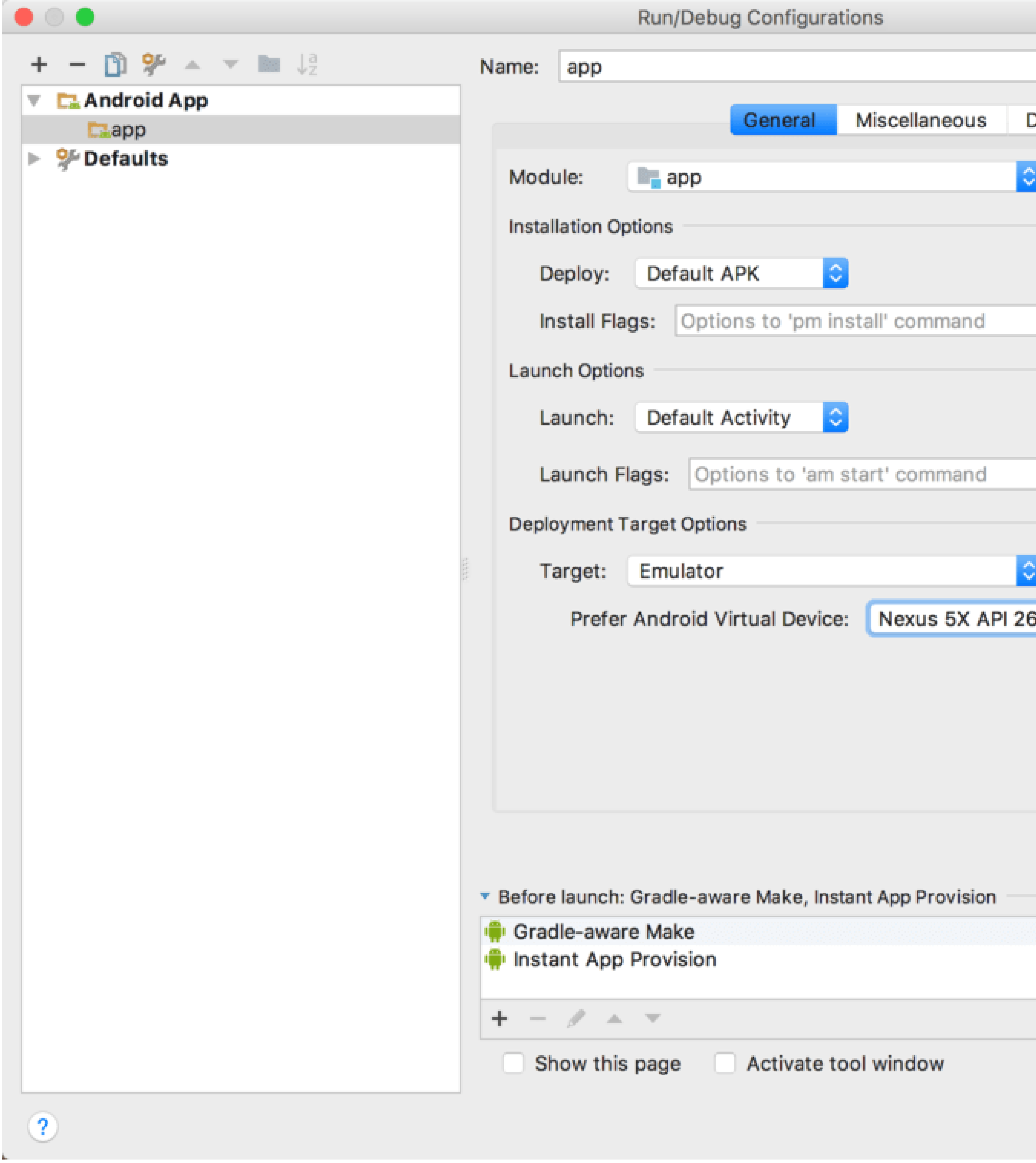
Assuming that the application loads into the emulator and runs as expected, we have safely verified that the Android development environment is correctly installed and configured.

Run/Debug Configurations

A particular project can be configured such that a specific device or emulator is used automatically each time it is run from within Android Studio. This avoids the necessity to make a selection from the device chooser each time the application is executed. To review and modify the Run/Debug configuration, click on the button to the left of the run button in the Android Studio toolbar and select the Edit Configurations… option from the resulting menu:



In the Run/Debug Configurations dialog, the application may be configured to always use a preferred emulator by selecting Emulator from the Target menu located in the Deployment Target Options section and selecting the emulator from the drop down menu. Figure 4-9, for example, shows the AndroidSample application configured to run by default on the previously created Nexus 5X emulator:



Run/Debug Configurations

Name: app

Module: app

Installation Options

Deploy: Default APK

Install Flags: Options to 'pm install' command

Launch Options

Launch: Default Activity

Launch Flags: Options to 'am start' command

Deployment Target Options

Target: Emulator

Prefer Android Virtual Device: Nexus 5X API 26

Before launch: Gradle-aware Make, Instant App Provision

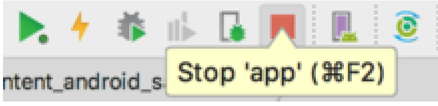
- Gradle-aware Make
- Instant App Provision

□ Show this page □ Activate tool window

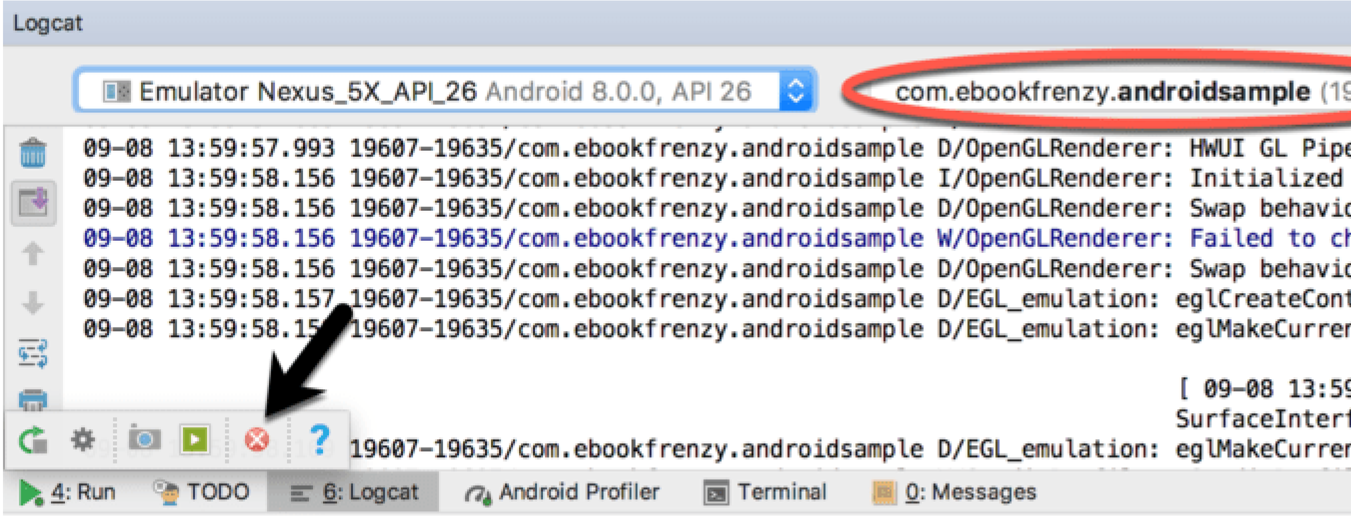
Be sure to switch the Target menu setting back to “Open Select Deployment Target Dialog” mode before moving on to the next chapter of the book.

Stopping a Running Application

To stop a running application, simply click on the stop button located in the main toolbar as shown in Figure 4-10:



An app may also be terminated using the Logcat tool window. Begin by displaying the Logcat tool window using the window bar button that becomes available when the app is running. Once the Logcat tool window appears, select the androidsample app menu highlighted in Figure 4-11 below:


Logcat
Emulator Nexus_5X_API_26 Android 8.0.0, API 26
com.ebookfrenzy.androidsample (19
4: Run
TODO
6: Logcat
Android Profiler
Terminal
0: Messages

```
09-08 13:59:57.993 19607-19635/com.ebookfrenzy.androidsample D/OpenGLRenderer: HWUI GL Pipe
09-08 13:59:58.156 19607-19635/com.ebookfrenzy.androidsample I/OpenGLRenderer: Initialized
09-08 13:59:58.156 19607-19635/com.ebookfrenzy.androidsample D/OpenGLRenderer: Swap behavic
09-08 13:59:58.156 19607-19635/com.ebookfrenzy.androidsample W/OpenGLRenderer: Failed to ch
09-08 13:59:58.156 19607-19635/com.ebookfrenzy.androidsample D/OpenGLRenderer: Swap behavic
09-08 13:59:58.157 19607-19635/com.ebookfrenzy.androidsample D/EGL_emulation: eglCreateCont
09-08 13:59:58.15  19607-19635/com.ebookfrenzy.androidsample D/EGL_emulation: eglMakeCurren

                                                                              [ 09-08 13:59
                                                                              SurfaceInterf
                   19607-19635/com.ebookfrenzy.androidsample D/EGL_emulation: eglMakeCurren
```

With the process selected, stop it by clicking on the red Terminate Application button in the toolbar to the left of the process list indicated by the arrow in the above figure.

AVD Command-line Creation

As previously discussed, in addition to the graphical user interface it is also possible to create a new AVD directly from the command-line. This is achieved using the avdmanager tool in conjunction with some command-line options. Once initiated, the tool will prompt for additional information before creating the new AVD.

The avdmanager tool requires access to the Java Runtime Environment (JRE) in order to run. If, when attempting run avdmanager, an error message appears indicating that the 'java' command cannot be found, the command prompt or terminal window within which you are running the command can be configured to use the OpenJDK environment bundled with Android Studio. Begin by identifying the location of the OpenJDK JRE as follows:

Launch Android Studio and open the AndroidSample project created earlier in the book.

Select the File -> Project Structure... menu option.

Copy the path contained within the JDK location field of the Project Structure dialog. This represents the location of the JRE bundled with Android Studio.

On Windows, execute the following command within the command prompt window from which avdmanager is to be run (where <path to jre> is replaced by the path copied from the Project Structure dialog above):

```
set JAVA_HOME=<path to jre>
```

On macOS or Linux, execute the following command:

```
export JAVA_HOME="<path to jre>"
```

If you expect to use the avdmanager tool frequently, follow the environment variable steps for your operating system outlined in the chapter entitled “Setting up an Android Studio Development Environment” to configure JAVA_HOME on a system-wide basis.

Assuming that the system has been configured such that the Android SDK tools directory is included in the PATH environment variable, a list of available targets for the new AVD may be obtained by issuing the following command in a terminal or command window:

```
avdmanager list targets
```

The resulting output from the above command will contain a list of Android SDK versions that are available on the system. For example:

```
Available Android targets:
----------
id: 1 or "android-28"
    Name: Android API 28
    Type: Platform
    API level: 28
    Revision: 3
----------
id: 2 or "android-26"
    Name: Android API 26
    Type: Platform
    API level: 26
```

```
     Revision: 1
```

The avdmanager tool also allows new AVD instances to be created from the command line. For example, to create a new AVD named Nexus9 using the target ID for the Android API level 26 device using the x86 ABI, the following command may be used:

```
avdmanager create avd -n Nexus9 -k "system-images;android-26;google_apis;x86"
```

The android tool will create the new AVD to the specifications required for a basic Android 8 device, also providing the option to create a custom configuration to match the specification of a specific device if required. Once a new AVD has been created from the command line, it may not show up in the Android Device Manager tool until the Refresh button is clicked.

In addition to the creation of new AVDs, a number of other tasks may be performed from the command line. For example, a list of currently available AVDs may be obtained using the list avd command line arguments:

```
avdmanager list avd

Available Android Virtual Devices:
    Name: Pixel_XL_API_28_No_Play
  Device: pixel_xl (Google)
    Path: /Users/neilsmyth/.android/avd/Pixel_XL_API_28_No_Play.avd
  Target: Google APIs (Google Inc.)
          Based on: Android API 28 Tag/ABI: google_apis/x86
    Skin: pixel_xl_silver
  Sdcard: 512M
```

Similarly, to delete an existing AVD, simply use the delete option as follows:

```
avdmanager delete avd –n <avd name>
```

Android Virtual Device Configuration Files

By default, the files associated with an AVD are stored in the .android/avd sub-directory of the user’s home directory, the structure of which is as follows (where <avd name> is replaced by the name assigned to the AVD):

```
<avd name>.avd/config.ini
```

```
<avd name>.avd/userdata.img

<avd name>.ini
```

The config.ini file contains the device configuration settings such as display dimensions and memory specified during the AVD creation process. These settings may be changed directly within the configuration file and will be adopted by the AVD when it is next invoked.

The <avd name>.ini file contains a reference to the target Android SDK and the path to the AVD files. Note that a change to the image.sysdir value in the config.ini file will also need to be reflected in the target value of this file.

Moving and Renaming an Android Virtual Device

The current name or the location of the AVD files may be altered from the command line using the avdmanager tool's move avd argument. For example, to rename an AVD named Nexus9 to Nexus9B, the following command may be executed:

```
avdmanager move avd -n Nexus9 -r Nexus9B
```

To physically relocate the files associated with the AVD, the following command syntax should be used:

```
avdmanager move avd -n <avd name> -p <path to new location>
```

For example, to move an AVD from its current file system location to /tmp/Nexus9Test:

```
avdmanager move avd -n Nexus9 -p /tmp/Nexus9Test
```

Note that the destination directory must not already exist prior to executing the command to move an AVD.

Summary

A typical application development process follows a cycle of coding, compiling and running in a test environment. Android applications may be tested on either a physical Android device or using an Android Virtual Device (AVD) emulator. AVDs are created and managed using the Android AVD Manager tool which may be used either as a command line tool or using a graphical user interface. When creating an AVD to simulate a specific Android device model it is important that the virtual device be configured with a hardware specification that matches that of the physical device.