- In the course of developing Android apps in Android Studio it will be necessary to compile and run an application multiple times.

- An Android application may be tested by installing and running it either on a physical device or in an Android Virtual Device (AVD) emulator environment.

- Before an AVD can be used, it must first be created and configured to match the specifications of a particular device model.

- The goal of this chapter, therefore, is to work through the steps involved in creating such a virtual device using the Pixel 3 phone as a reference example.

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.

- Custom configurations may be 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.

- 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 Pixel 3 model.

Figure 4-1

- 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.

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 > AVD Manager menu option from within the main window.

- Once launched, the tool will appear as outlined in Figure 4-2 if no existing AVD instances have been created:

Figure 4-2

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

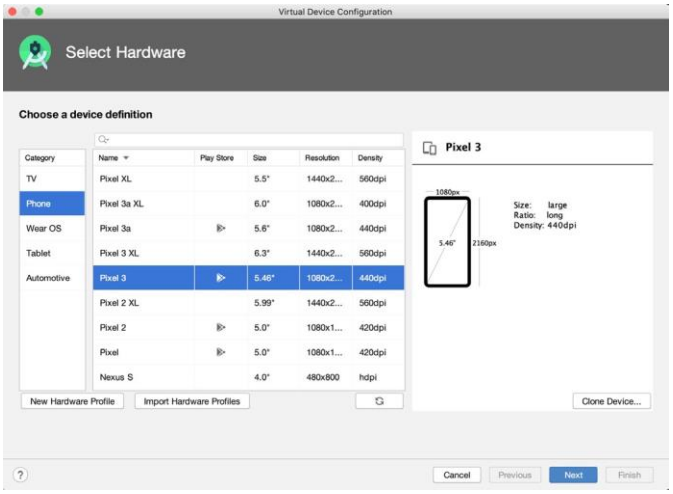

Figure 4-3

- Within the dialog, perform the following steps to create a Pixel 3 compatible emulator:

- From the Category panel, select the Phone option to display the list of available Android phone AVD templates.
- Select the Pixel 3 device option and click Next.
- On the System Image screen, select the latest version of Android 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 Pixel 3 API 29) 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 Pixel 3 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, make sure that the newly created Pixel 3 AVD is displayed in the device menu (marked A in Figure 4-4 below), then either click on the run button represented by a green triangle (B), select the Run > Run 'app' menu option or use the Ctrl-R keyboard shortcut:

Figure 4-4

- The device menu (A) may be used to select a different AVD instance or physical device as the run target, and also to run the app on multiple devices.

- The menu also provides access to the AVD Manager and device connection trouble shooting options:

Figure 4-5

- Once the application is installed and running, the user interface for the first fragment will appear within the emulator:

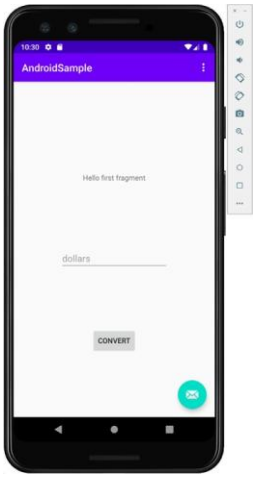

Figure 4-6

- 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 tool window 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:

Figure 4-7

- 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.

Stopping a Running Application

- To stop a running application, simply click on the stop button located in the main toolbar as shown in Figure 4-8:

Figure 4-8

- An app may also be terminated using the Run tool window. Begin by displaying the Run tool window using the window bar button that becomes available when the app is running.

- Once the Run tool window appears, click the stop button highlighted in Figure 4-9 below:

Figure 4-9

Supporting Dark Theme

- Android 10 introduced the much awaited dark theme, support for which is not enabled by default in Android Studio app projects.

- To test dark theme in the AVD emulator, open the Settings app within the running Android instance in the emulator.
- There are a number of different ways to access the settings app.
- The quickest is to display the home screen and then click and drag upwards from the bottom of the screen (just below the search bar).
- This will display all of the apps installed on the device, one of which will be the Settings app.
- Within the Settings app, choose the Display category and enable the Dark Theme option as shown in Figure 4-10 so that the screen background turns black:

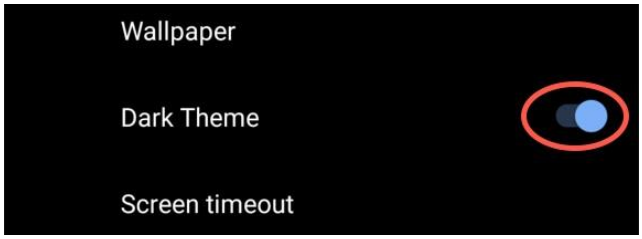

Figure 4-10

- With dark theme enabled, run the AndroidSample app and note that it appears as before and does not conform to the dark theme.
- In order for an app to adopt dark theme, it must be derived from the Android DayNight theme.
- By default, new projects use the Light.DarkActionBar theme.
- To change this setting, navigate to the res > values > styles.xml file in the Project window as shown in Figure 4-11 and double-click on it to load it into the editor:

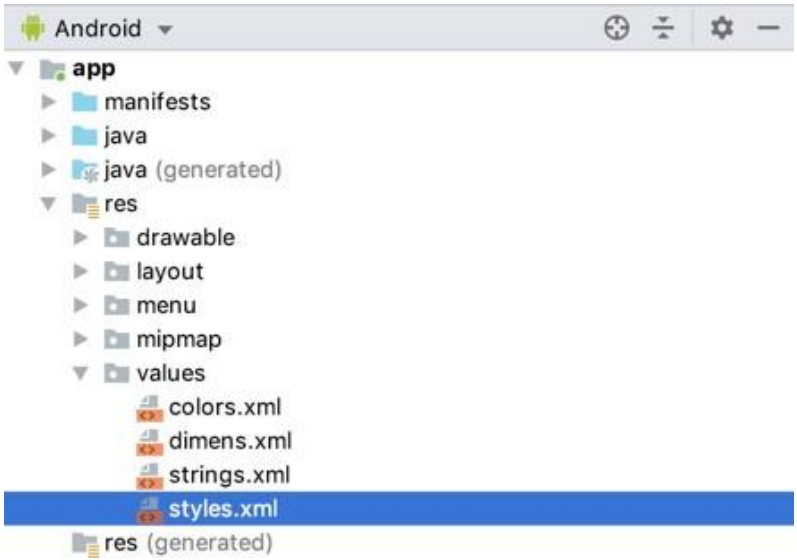

Figure 4-11

- Once loaded, edit the AppTheme style entry so that it reads as follows:

```
<resources>
  <!-- Base application theme. ->
  <style name="AppTheme" parent="Theme.AppCompat.DayNight">
  <!-- Customize your theme here. ->
  .
  .
```

- After making the change, re-run the app on the emulator and note that it now conforms to the dark theme as shown in Figure 4-12:

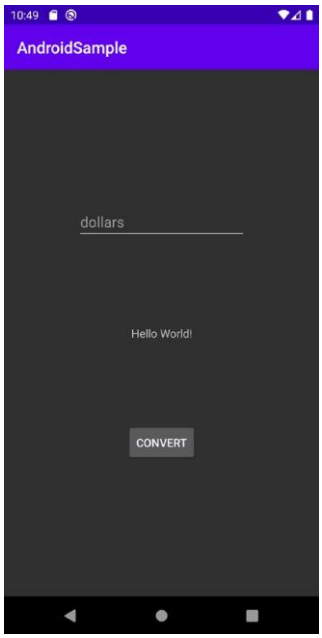

Figure 4-12

- Open the Settings app, turn off dark theme and return to the AndroidSample app.

- The app should have automatically switched back to light mode.

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:

  1. Launch Android Studio and open the AndroidSample project created earlier in our studies.

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

  3. 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.

  4. 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-29"
   Name: Android API 29
   Type: Platform
   API level: 29
   Revision: 1
----------
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 myAVD using the target ID for the Android API level 29 device using the x86 ABI, the following command may be used:

  avdmanager create avd -n myAVD -k "system-images;android-29;google_apis_ playstore;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/Geoffrey/.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.

Using and Configuring the Android Studio AVD Emulator

- The Android Virtual Device (AVD) emulator environment bundled with Android Studio 1.x was an uncharacteristically weak point in an otherwise reputable application development environment.

- Regarded by many developers as slow, inflexible and unreliable, the emulator was long overdue for an overhaul.

- Fortunately, Android Studio 2 introduced an enhanced emulator environment providing significant improvements in terms of configuration flexibility and overall performance.

- Further enhancements have been made in subsequent releases.

- Before the next chapter explores testing on physical Android devices, this chapter will take some time to provide an overview of the Android Studio AVD emulator and highlight many of the configuration features that are available to customize the environment.

The Emulator Environment

- When launched, the emulator displays an initial splash screen during the loading process.

- Once loaded, the main emulator window appears containing a representation of the chosen device type (in the case of Figure 4-13 this is a Nexus 5X device):

Figure 4-13

- Positioned along the right-hand edge of the window is the toolbar providing quick access to the emulator controls and configuration options.

The Emulator Toolbar Options

- The emulator toolbar (Figure 4-14) provides access to a range of options relating to the appearance and behavior of the emulator environment.

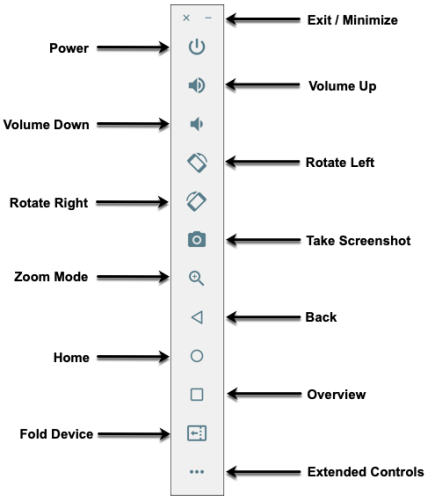

Figure 4-14

- Each button in the toolbar has associated with it a keyboard accelerator which can be identified either by hovering the mouse pointer over the button and waiting for the tooltip to appear, or via the help option of the extended controls panel.

- Though many of the options contained within the toolbar are self-explanatory, each option will be covered for the sake of completeness:

  - Exit / Minimize – The uppermost 'x' button in the toolbar exits the emulator session when selected while the '-' option minimizes the entire window.

  - Power – The Power button simulates the hardware power button on a physical Android device. Clicking and releasing this button will lock the device and turn off the screen. Clicking and holding this button will initiate the device "Power off " request sequence.

  - Volume Up / Down – Two buttons that control the audio volume of playback within the simulator environment.

  - Rotate Left/Right – Rotates the emulated device between portrait and landscape orientations.

  - Take Screenshot – Takes a screenshot of the content currently displayed on the device screen. The captured image is stored at the location specified in the Settings screen of the extended controls panel as outlined later in this chapter.

  - Zoom Mode – This button toggles in and out of zoom mode, details of which will be covered later in this chapter.

  - Back – Simulates selection of the standard Android "Back" button. As with the Home and Overview buttons outlined below, the same results can be achieved by selecting the actual buttons on the emulator screen.

  - Home – Simulates selection of the standard Android "Home" button.

  - Overview – Simulates selection of the standard Android "Overview" button which displays the currently running apps on the device.

  - Fold Device – Simulates the folding and unfolding of a foldable device. This option is only available if the emulator is running a foldable device system image.

  - Extended Controls – Displays the extended controls panel, allowing for the configuration of options such as simulated location and telephony activity, battery strength, cellular network type and fingerprint identification.

Working in Zoom Mode

- The zoom button located in the emulator toolbar switches in and out of zoom mode.

- When zoom mode is active the toolbar button is depressed and the mouse pointer appears as a magnifying glass when hovering over the device screen.

- Clicking the left mouse button will cause the display to zoom in relative to the selected point on the

screen, with repeated clicking increasing the zoom level.

- Conversely, clicking the right mouse button decreases the zoom level.
- Toggling the zoom button off reverts the display to the default size.
- Clicking and dragging while in zoom mode will define a rectangular area into which the view will zoom when the mouse button is released.
- While in zoom mode the visible area of the screen may be panned using the horizontal and vertical scrollbars located within the emulator window.

Resizing the Emulator Window

The size of the emulator window (and the corresponding representation of the device) can be changed at any time by clicking and dragging on any of the corners or sides of the window.

Extended Control Options

- The extended controls toolbar button displays the panel illustrated in Figure 4-15.
- By default, the location settings will be displayed. Selecting a different category from the left-hand panel will display the corresponding group of controls:

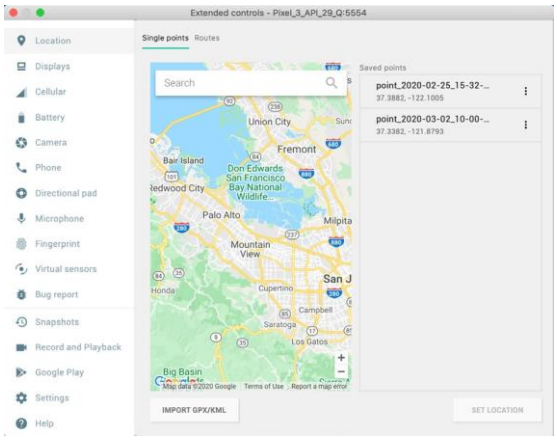

Figure 4-15

Location

- The location controls allow simulated location information to be sent to the emulator in the form of decimal or sexigesimal coordinates.
- Location information can take the form of a single location, or a sequence of points representing movement of the device, the latter being provided via a file in either GPS Exchange (GPX) or Keyhole Markup Language (KML) format.
- Alternatively, the integrated Google Maps panel may be used to visually select single points or travel routes.

Displays

- In addition to the main display shown within the emulator screen, the Displays option allows additional displays to be added running within the same Android instance.
- This can be useful for testing apps for dual screen devices such as the Microsoft Surface Duo.
- These additional screens can be configured to be any required size and appear within the same emulator window as the main screen.

Cellular

- The type of cellular connection being simulated can be changed within the cellular settings screen.
- Options are available to simulate different network types (CSM, EDGE, HSDPA etc) in addition to a range of voice and data scenarios such as roaming and denied access.

Camera

- The emulator simulates a 3D scene when the camera is active.
- This takes the form of the interior of a virtual building through which you can navigate by holding down the Option key (Alt on Windows) while using the mouse pointer and keyboard keys when recording video or before taking a photo within the emulator.
- This extended configuration option allows different images to be uploaded for display within the virtual environment.

Battery

A variety of battery state and charging conditions can be simulated on this panel of the extended controls screen, including battery charge level, battery health and whether the AC charger is currently connected.

Phone

- The phone extended controls provide two very simple but useful simulations within the emulator.
- The first option allows for the simulation of an incoming call from a designated phone number.
- This can be of particular use when testing the way in which an app handles high level interrupts of this nature.
- The second option allows the receipt of text messages to be simulated within the emulator session.
- As in the real world, these messages appear within the Message app and trigger the standard notifications within the emulator.

Directional Pad

- A directional pad (D-Pad) is an additional set of controls either built into an Android device or connected

externally (such as a game controller) that provides directional controls (left, right, up, down).

- The directional pad settings allow D-Pad interaction to be simulated within the emulator.

Microphone

- The microphone settings allow the microphone to be enabled and virtual headset and microphone connections to be simulated.

- A button is also provided to launch the Voice Assistant on the emulator.

Fingerprint

- Many Android devices are now supplied with built-in fingerprint detection hardware.

- The AVD emulator makes it possible to test fingerprint authentication without the need to test apps on a physical device containing a fingerprint sensor.

- Details on how to configure fingerprint testing within the emulator will be covered in detail later in this chapter.

Virtual Sensors

The virtual sensors option allows the accelerometer and magnetometer to be simulated to emulate the effects of the physical motion of a device such as rotation, movement and tilting through yaw, pitch and roll settings.

Snapshots

- Snapshots contain the state of the currently running AVD session to be saved and rapidly restored making it easy to return the emulator to an exact state.

- Snapshots are covered in detail later in this chapter.

Record and Playback

Allows the emulator screen and audio to be recorded and saved in either WebM or animated GIF format.

Google Play

- If the emulator is running a version of Android with Google Play Services installed, this option displays the current Google Play version.

- It also provides the option to update the emulator to the latest version.

Settings

- The settings panel provides a small group of configuration options.

- Use this panel to choose a darker theme for the toolbar and extended controls panel, specify a file system

location into which screenshots are to be saved, configure OpenGL support levels, and to configure the emulator window to appear on top of other windows on the desktop.

Help

The Help screen contains three sub-panels containing a list of keyboard shortcuts, links to access the emulator online documentation, file bugs and send feedback, and emulator version information.

Working with Snapshots

- When an emulator starts for the very first time it performs a cold boot much like a physical Android device when it is powered on.
- This cold boot process can take some time to complete as the operating system loads and all the background processes are started.
- To avoid the necessity of going through this process every time the emulator is started, the system is configured to automatically save a snapshot (referred to as a quick-boot snapshot) of the emulator's current state each time it exits.
- The next time the emulator is launched, the quick-boot snapshot is loaded into memory and execution resumes from where it left off previously, allowing the emulator to restart in a fraction of the time needed for a cold boot to complete.
- The Snapshots screen of the extended controls panel can be used to store additional snapshots at any point during the execution of the emulator.
- This saves the exact state of the entire emulator allowing the emulator to be restored to the exact point in time that the snapshot was taken.
- From within the screen, snapshots can be taken using the Take Snapshot button (marked A in Figure 4-16).
- To restore an existing snapshot, select it from the list (B) and click the run button (C) located at the bottom of the screen.
- Options are also provided to edit (D) the snapshot name and description and to delete (E) the currently selected snapshot:

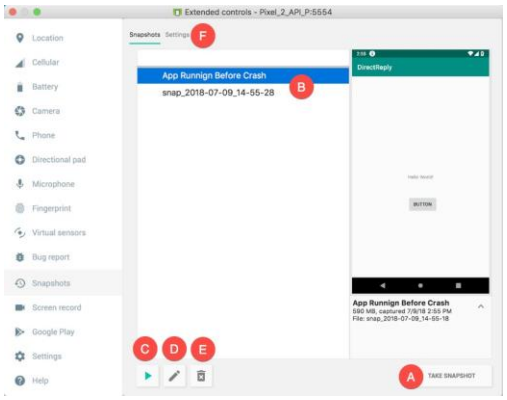

Figure 4-16

- The Settings option (F) provides the option to configure the automatic saving of quick-boot snapshots (by default the emulator will ask whether to save the quick boot snapshot each time the emulator exits) and to reload the most recent snapshot.

- To force an emulator session to perform a cold boot instead of using a previous quick-boot snapshot, open the AVD Manager (Tools -> AVD Manager), click on the down arrow in the actions column for the emulator and select the Cold Boot Now menu option.

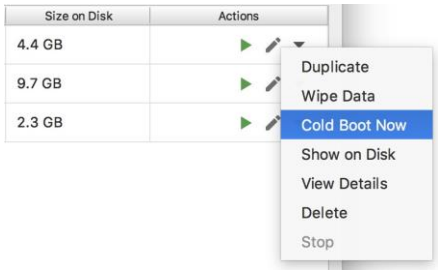

Figure 4-17

Configuring Fingerprint Emulation

- The emulator allows up to 10 simulated fingerprints to be configured and used to test fingerprint authentication within Android apps.

- To configure simulated fingerprints begin by launching the emulator, opening the Settings app and selecting the Security & Location option.

- Within the Security settings screen, select the Use fingerprint option.

- On the resulting information screen click on the Next button to proceed to the Fingerprint setup screen.

- Before fingerprint security can be enabled a backup screen unlocking method (such as a PIN number) must be configured. Click on the Fingerprint + PIN button and, when prompted, choose not to require the PIN on device startup.

- Enter and confirm a suitable PIN number and complete the PIN entry process by accepting the default notifications option.

- Proceed through the remaining screens until the Settings app requests a fingerprint on the sensor. At this point display the extended controls dialog, select the Fingerprint category in the left-hand panel and make sure that Finger 1 is selected in the main settings panel:

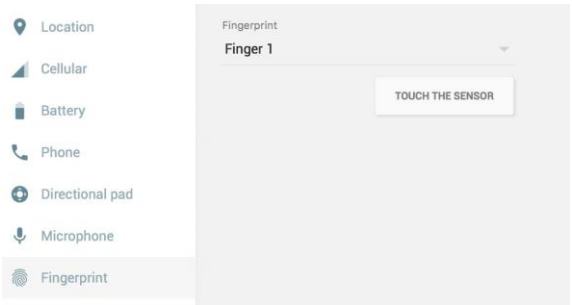

Figure 4-18

- Click on the Touch the Sensor button to simulate Finger 1 touching the fingerprint sensor.

- The emulator will report the successful addition of the fingerprint:

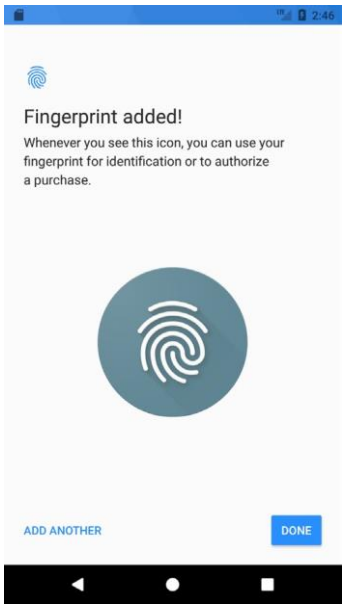

Figure 4-19

- To add additional fingerprints click on the Add Another button and select another finger from the extended controls panel menu before clicking on the Touch the Sensor button once again.

- The topic of building fingerprint authentication into an Android app is covered in detail in the chapter entitled "An Android Biometric Authentication Tutorial".

Testing Android Studio Apps on a Physical Android Device

- While much can be achieved by testing applications using an Android Virtual Device (AVD), there is no substitute for performing real world application testing on a physical Android device and there are a number of Android features that are only available on physical Android devices.

- Communication with both AVD instances and connected Android devices is handled by the Android Debug Bridge (ADB).

- In this chapter we will work through the steps to configure the adb environment to enable application testing on a physical Android device with macOS, Windows and Linux based systems.

An Overview of the Android Debug Bridge (ADB)

- The primary purpose of the ADB is to facilitate interaction between a development system, in this case Android Studio, and both AVD emulators and physical Android devices for the purposes of running and debugging applications.

- The ADB consists of a client, a server process running in the background on the development system and a daemon background process running in either AVDs or real Android devices such as phones and tablets.

- The ADB client can take a variety of forms.
- For example, a client is provided in the form of a command-line tool named adb located in the Android SDK platform-tools sub-directory.
- Similarly, Android Studio also has a built-in client.
- A variety of tasks may be performed using the adb command-line tool.
- For example, a listing of currently active virtual or physical devices may be obtained using the devices command-line argument.
- The following command output indicates the presence of an AVD on the system but no physical devices:

```
$ adb devices
List of devices attached emulator-5554  device
```

Enabling ADB on Android based Devices

- Before ADB can connect to an Android device, that device must first be configured to allow the connection.
- On phone and tablet devices running Android 6.0 or later, the steps to achieve this are as follows:

1. Open the Settings app on the device and select the About tablet or About phone option (on newer versions of Android this can be found on the System page of the Settings app).
2. On the About screen, scroll down to the Build number field (Figure 4-20) and tap on it seven times until a message appears indicating that developer mode has been enabled.
3. If the build number is not displayed, unfold the Advanced section of the list.

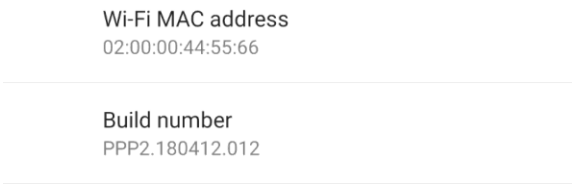

Figure 4-20

4. Return to the main Settings screen and note the appearance of a new option titled Developer options. Select this option and locate the setting on the developer screen entitled USB debugging.
5. Enable the switch next to this item as illustrated in Figure 4-21:

Figure 4-21

6. Swipe downward from the top of the screen to display the notifications panel (Figure 4-22) and note that the device is currently connected for debugging.

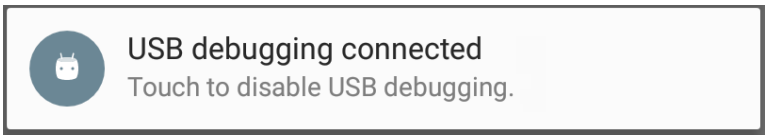

Figure 4-22

- At this point, the device is now configured to accept debugging connections from adb on the development system.

- All that remains is to configure the development system to detect the device when it is attached.

- While this is a relatively straightforward process, the steps involved differ depending on whether the development system is running Windows, macOS or Linux.

- Note that the following steps assume that the Android SDK platform-tools directory is included in the operating system PATH environment variable as described in the chapter entitled "Setting up an Android Studio Development Environment".

macOS ADB Configuration

- In order to configure the ADB environment on a macOS system, connect the device to the computer system using a USB cable, open a terminal window and execute the following command to restart the adb server:

```
$ adb kill-server
$ adb start-server
daemon not running. starting it now on port 5037 *
daemon started successfully *
```

- Once the server is successfully running, execute the following command to verify that the device has been detected:

```
$ adb devices
List of devices attached
74CE000600000001      offline
```

- If the device is listed as offline, go to the Android device and check for the presence of the dialog shown in Figure 4-23 seeking permission to Allow USB debugging. Enable the checkbox next to the option that reads
- *Always allow from this computer*, before clicking on OK.

- Repeating the adb devices command should now list the device as being available:

```
List of devices attached
015d41d4454bf80c      device
```

- In the event that the device is not listed, try logging out and then back in to the macOS desktop and, if the problem persists, rebooting the system.

Windows ADB Configuration

- The first step in configuring a Windows based development system to connect to an Android device using ADB is to install the appropriate USB drivers on the system.

- The USB drivers to install will depend on the model of Android Device.

- If you have a Google Nexus device, then it will be necessary to install and configure the Google USB Driver package on your Windows system.

- Detailed steps to achieve this are outlined on the following web page:

  ```
  https://developer.android.com/sdk/win-usb.html
  ```

- For Android devices not supported by the Google USB driver, it will be necessary to download the drivers provided by the device manufacturer.

- A listing of drivers together with download and installation information can be obtained online at:

  ```
  https://developer.android.com/tools/extras/oem-usb.html
  ```

- With the drivers installed and the device now being recognized as the correct device type, open a Command Prompt window and execute the following command:

  ```
  adb devices
  ```

- This command should output information about the connected device similar to the following:

  ```
  List of devices attached
  HT4CTJT01906    offline
  ```

- If the device is listed as offline or unauthorized, go to the device display and check for the dialog shown in Figure 4-23 seeking permission to Allow USB debugging.

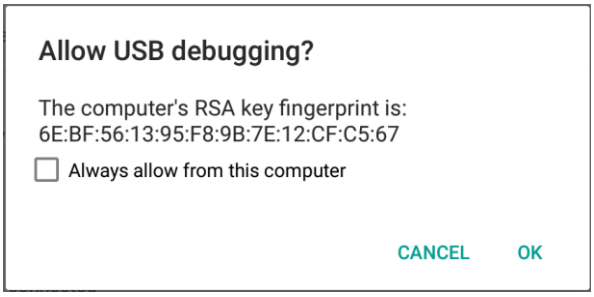

Figure 4-23

- Enable the checkbox next to the option that reads Always allow from this computer, before clicking on OK. Repeating the adb devices command should now list the device as being ready:

  ```
  List of devices attached
  HT4CTJT01906    device
  ```

- In the event that the device is not listed, execute the following commands to restart the ADB server:

```
adb kill-server
adb start-server
```

- If the device is still not listed, try executing the following command:

```
android update adb
```

- Note that it may also be necessary to reboot the system.

Linux adb Configuration

- For the purposes of this chapter, we will once again use Ubuntu Linux as a reference example in terms of configuring adb on Linux to connect to a physical Android device for application testing.

- Physical device testing on Ubuntu Linux requires the installation of a package named android-tools-adb which, in turn, requires that the Android Studio user be a member of the plugdev group.

- This is the default for user accounts on most Ubuntu versions and can be verified by running the id command.

- If the plugdev group is not listed, run the following command to add your account to the group:

  sudo usermod -aG plugdev $LOGNAME

- After the group membership requirement has been met, the android-tools-adb package can be installed by executing the following command:

  sudo apt-get install android-tools-adb

- Once the above changes have been made, reboot the Ubuntu system. Once the system has restarted, open a Terminal window, start the adb server and check the list of attached devices:

  $ adb start-server
  daemon not running. starting it now on port 5037 *
  daemon started successfully *
  $ adb devices
  List of devices attached 015d41d4454bf80c offline

- If the device is listed as offline or unauthorized, go to the Android device and check for the dialog shown in Figure 4-23 above seeking permission to Allow USB debugging.

Testing the adb Connection

- Assuming that the adb configuration has been successful on your chosen development platform, the next step is to try running the test application created in the chapter entitled “Creating an Example Android App in Android Studio” on the device.

- Launch Android Studio, open the AndroidSample project and verify that the device appears in the device selection menu as highlighted in Figure 4-24:

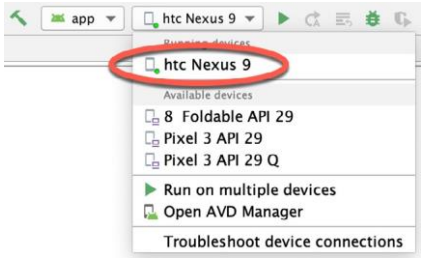

Figure 4-24

- Note that this menu also includes the option to test the app on multiple devices and emulators simultaneously.

- When selected, this option displays the dialog shown in Figure 4-25 where multiple deployment targets may be selected.

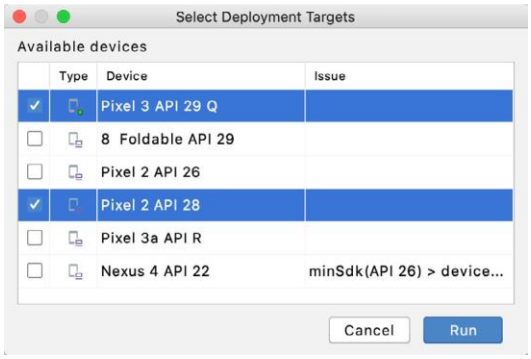

Figure 4-25

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.

- Android Studio 4.0 contains an Android Virtual Device emulator environment designed to make it easier to test applications without the need to run on a physical Android device.

- This chapter has provided a brief tour of the emulator and highlighted key features that are available to configure and customize the environment to simulate different testing conditions.

- While the Android Virtual Device emulator provides an excellent testing environment, it is important to keep in mind that there is no real substitute for making sure an application functions correctly on a physical Android device.

- This, after all, is where the application will be used in the real world.

- By default, however, the Android Studio environment is not configured to detect Android devices as a target testing device.

- It is necessary, therefore, to perform some steps in order to be able to load applications directly onto an Android device from within the Android Studio development environment.

- The exact steps to achieve this goal differ depending on the development platform being used.

- In this chapter, we have covered those steps for Linux, macOS and Windows based platforms.