

Android App's Anatomy

Android Applications (Just Apps)

- Android applications are usually created using the **Java** programming language ^[1]
- Apps must import various **Android Libraries** (such as android.jar, maps.jar, etc) to gain the functionality needed to work inside the Android OS.
- Android apps are made of multiple elements such as: user-defined classes, android jars, third-party libraries, XML files defining the UIs or views, multimedia resources, data assets such as disk files, external arrays and strings, databases, and finally a *Manifest* summarizing the 'anatomy' and permissions requested by the app.
- The various app components are given to the compiler to obtain a single signed and deployable **Android Package** (an **.apk** file).
- Like “.class” files in Java, “.apk” files are the **byte-code** version of the app that finally will be 'executed' by interpretation inside either a **Dalvik Virtual Machine** (DVM) or an Android-Runtime Engine (**ART**).

[1] Visit <http://xamarin.com/monoforandroid> for a commercial iOS and Android IDE that works with C# and Windows .NET

Android's Byte-Code Execution

Dalvik Virtual Machine vs. Android Runtime (ART)

The **Dalvik Virtual Machine** is a Just-in-Time (JIT) runtime environment (similar to the Oracle's Java Virtual Machine JVM) that interprets Android byte-code only when it's needed (however it will be phased out soon).

The newer **ART** (introduced as an option in Android 4.4 KitKat) is an anticipatory or Ahead-of-Time (AOT) environment that compiles code before it is actually needed.

ART promises:

- enhanced performance and battery efficiency,
- improved garbage collection,
- better debugging facilities,
- Improved diagnostic detail in exceptions and crash reports.

Quoting from

<https://source.android.com/devices/tech/dalvik/art.html> (Aug-27-2014)

Important: *Dalvik must remain the default runtime or you risk breaking your Android implementations and third-party applications.*

Setting up Eclipse + ADT + SDK

You are a developer - Which is your SDK audience?

SDKs are named after types of desserts. Available versions at the time of writing are:

- 1.5 Cupcake,**
- 1.6 Donut,**
- 2.1 Eclair,**
- 2.2 Froyo,**
- 2.3 Gingerbread,**
- 3.x Honeycomb,**
- 4.0 Ice Cream Sandwich**
- 4.3 Jelly Bean**
- 4.4 Kitkat**
- 5.x Lollipop**
- 6.X Marshmallow**

Android SDK version	Current market share
4.4 (KitKat)	42.0 %
4.1-4.3 (Jelly Bean)	34.4 %
5.0-5.1 (Lollipop)	16.5 %
2.3 (Gingerbread)	3.5 %
4.0.x (ICS)	3.3 %
2.2 (Froyo)	0.2 %
3.0-3.2 (Honeycomb)	0.1 %
2.0-2.1 (Eclair)	0.0 %

Statistics accessed on **Sept 3, 2015** from AppBrain at <http://www.appbrain.com/stats/top-android-sdk-versions>

Tools for Constructing Android Apps

Development Workbenches

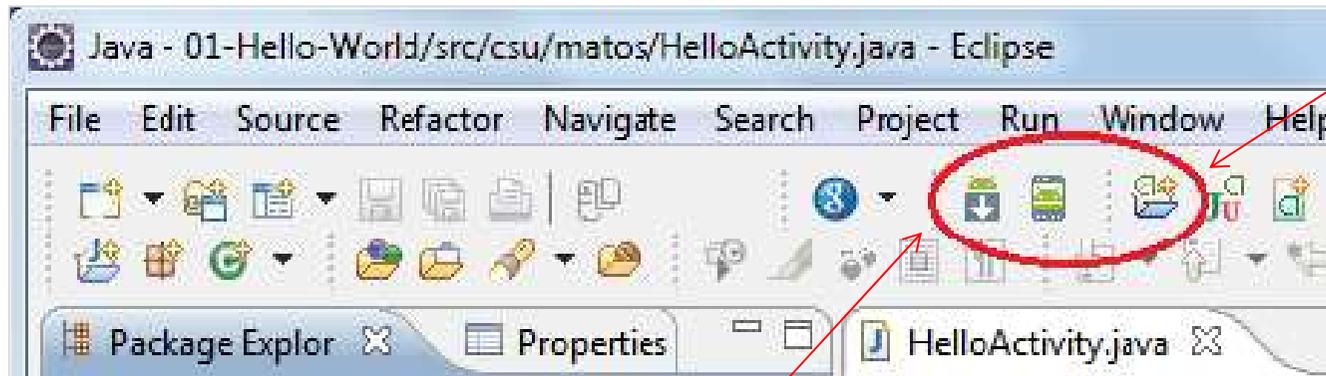
Android apps are made out of many components. The use of an IDE is *strongly* suggested to assist the developer in creating an Android solution. There are various options including:

- **Eclipse+ADT.** The classic general purpose Eclipse IDE can be enhanced (with the ADT plugin) to provide a ‘conventional’ way to create and debug Android Apps. The associated **SDK Manager** allows you to reach the various API libraries needed by the apps.
- **Android Studio** is a new Android-only development environment based on IntelliJ IDEA. It is the ‘preferred’ IDE platform for Android development.
- **Netbeans+Android.** Similar to Eclipse+ADT. Soon to be deprecated(?)



Eclipse + ADT + SDK

Typical Layout of the Eclipse-ADT IDE for Android Development



These icons are added to Eclipse by the ADT plugin

-  Opens Android SDK manager
-  Opens Android AVD Virtual Device Manager
-  Wizard creates a new Android Project



Opens DDMS Perspective
Dalvik Debugging Monitoring System

Note: The **DDMS** and **Hierarchy View** can be manually added by the user to Eclipse's tool bar

Android Studio

Typical Layout of Android-Studio IDE



The screenshot displays the Android Studio IDE interface. The main window shows a project named "CurrencyExchange" with the following structure:

- app
 - manifests
 - java
 - matos.csu.currencyexchd
 - MainActivity
 - res
 - drawable
 - layout
 - activity_main.xml
 - menu
 - menu_main.xml
 - mipmap
 - values
 - dimens.xml (2)
 - strings.xml
 - styles.xml (2)
 - Gradle Scripts

The code editor shows the following Java code for MainActivity.java:

```
import android.view.inputmethod.EditorInfo;
import android.widget.Button;
import android.widget.EditText;

import java.text.DecimalFormat;

public class MainActivity extends AppCompatActivity {
    //USA money format (12 d
    DecimalFormat usaDf = ne

    // naive currency conver
    private final double EUR
    private final char EUROS
    private final double COL
    private final char COLON
```

The "Welcome to Android Studio" dialog box is visible in the foreground, showing a list of recent projects and options to create or import a project. The dialog includes the following options:

- Recent Projects
 - My Application
- New Project...
- Import Project...
- Open Project
- VCS
 - Check out from Version Control
- Configure
- Docs and How-Tos

The bottom of the IDE shows the "Android" tab with "No Connected Devices" and "No Debuggable Applications" displayed. The logcat window shows the following output:

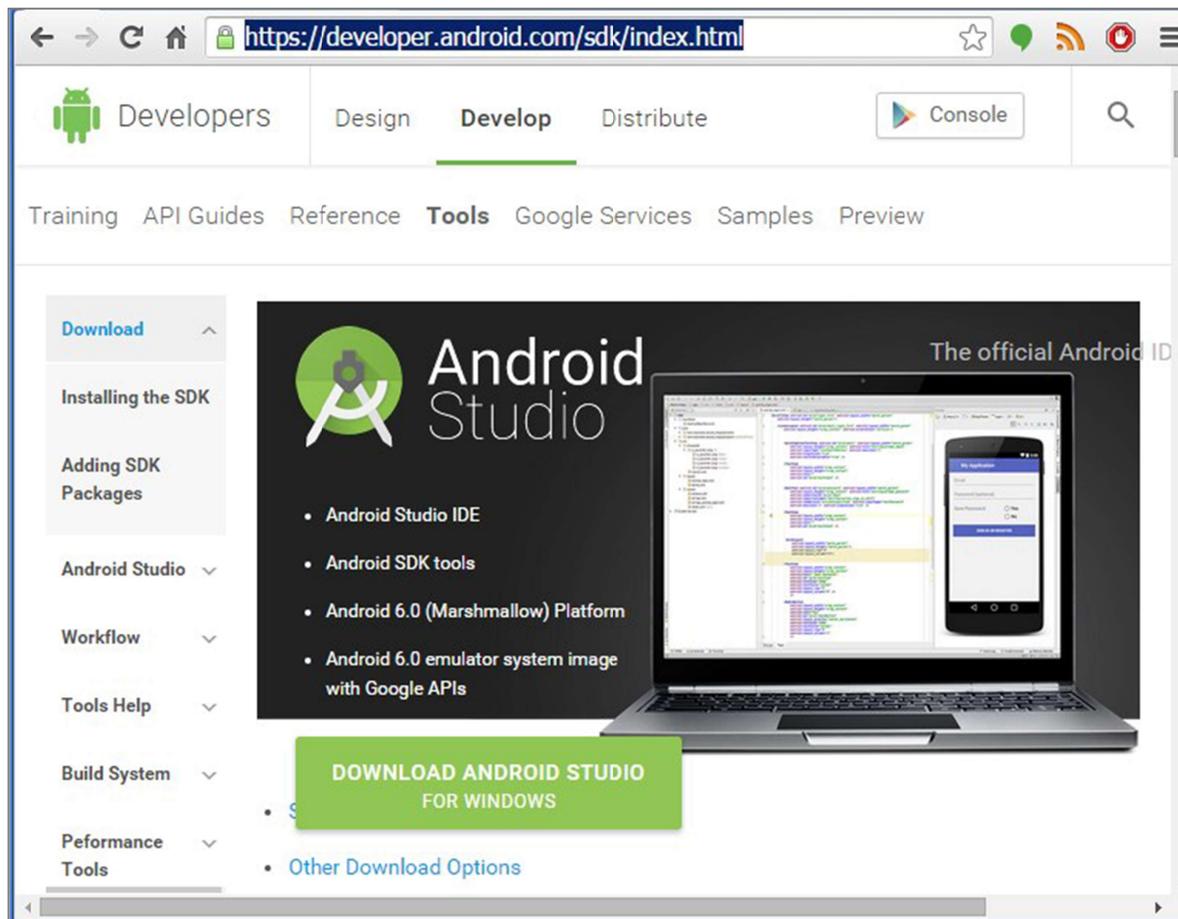
```
09-03 12:03:00.030 1392-1392/system_process I/ActivityManager:
09-03 12:03:00.040 1392-1392/system_process I/ActivityManager:
```

Setting up Android Studio

Downloading Android Studio IDE

Download IDE from: <https://developer.android.com/sdk/index.html>

Run the executable, you are (*almost*) done!



Setting up Eclipse + ADT + SDK

ECLIPSE SETUP

Prepare your computer – Install SDK: Windows, Mac, Linux

We assume you have already installed the most recent Java JDK and Eclipse IDE in your computer



- Java JDK is available at:

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

- Eclipse IDE for Java EE Developers is available at:

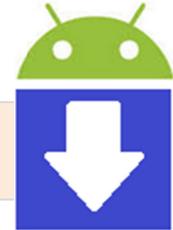
<http://www.eclipse.org/downloads/>



The next instructions are given to:

- (a) User wanting to add a newer SDK to their existing collection,
- (b) First time users (who may or not be Eclipse users).

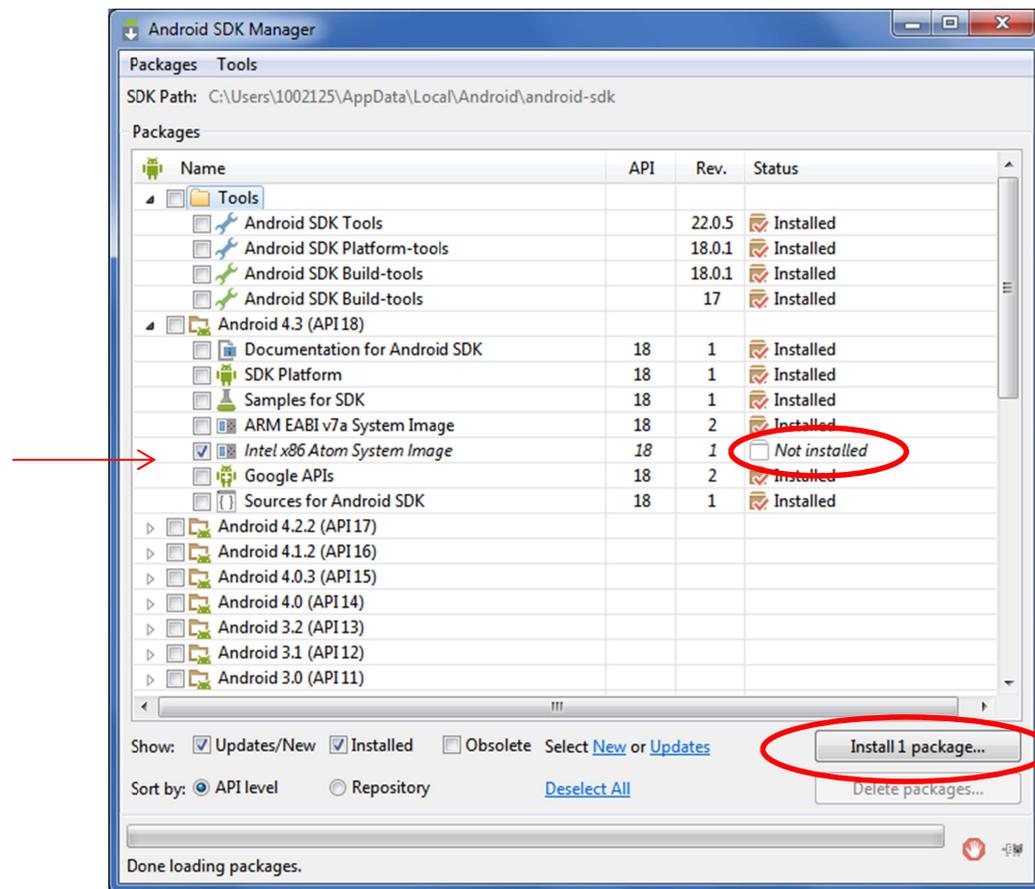
Setting up Eclipse + ADT + SDK



(a) Users Wanting to Update an Older Android Workbench

If you are currently using the Android SDK, you just need to *update* to the latest tools or platform using the already installed *Android SDK Manager*.

1. Click on the  *SDK Manager* icon.
2. You will see a form similar to the one on the right.
3. Select the SDK packages and independent components you want to install (click 'Install' button and wait until they are setup in your machine...)



Setting up Eclipse + ADT + SDK

(b) First Time Android Users who have Eclipse already installed

1. Obtain the appropriate (Windows, Mac, Linux) **Stand-alone SDK Tools for Windows** from the page <http://developer.android.com/sdk/index.html>
Execute the program, *remember the folder's name and location* in which the SDK is stored, you will have to supply this path to Eclipse.
2. Install the **ADT Plugin** for Eclipse (it must be already available in your machine)
 1. Start Eclipse, then select **Help > Install New Software....**
 2. Click **Add** button (top-right corner)
 3. In the next dialog-box enter "**ADT Plugin**" for the *Name* and the following URL for the *Location*: <https://dl-ssl.google.com/android/eclipse/>
 4. Click **OK**
 5. Select the checkbox next to **Developer Tools** and click **Next > Next**
 6. Accept the license agreements, then click **Finish**.
 7. After the installation end you need to restart Eclipse.
3. Add **Android platforms** and other components to your SDK (see previous option (a))

Setting up Eclipse + ADT + SDK

Configure the ADT Plugin

4. The next step is to inform your Eclipse+ADT workbench of the **android-sdk** directory's location (this is the path you saved on Step1)

1. In Eclipse, select **Window > Preferences...** to open the Preferences panel (Mac OS X: **Eclipse > Preferences**).
2. Select **Android** from the left panel.
3. To set the box *SDK Location* that appears in the main panel, click **Browse...** and locate your downloaded SDK directory (usually **C:\Program Files (x86)\Android\android-sdk**)
4. Click **Apply**, then **OK**.

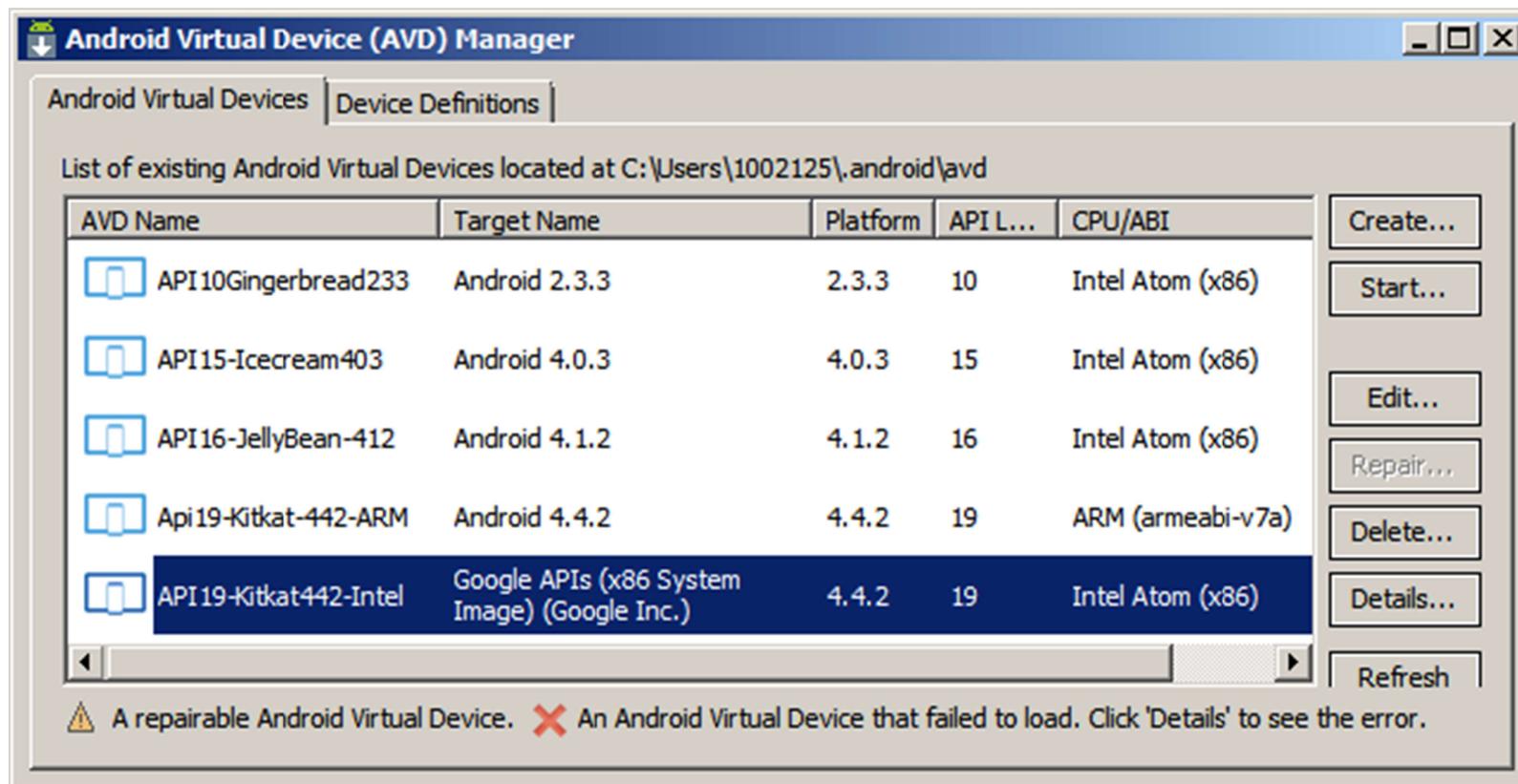
Done!

Setting up Eclipse + ADT + SDK



Working with Virtual Devices (AVDs)

Ideally you should test your applications on a device (a physical phone or tablet). However, the SDK allows you to create realistic virtual devices on which your applications could be executed/debugged before they are deployed on actual hardware.



Setting up Eclipse + ADT + SDK



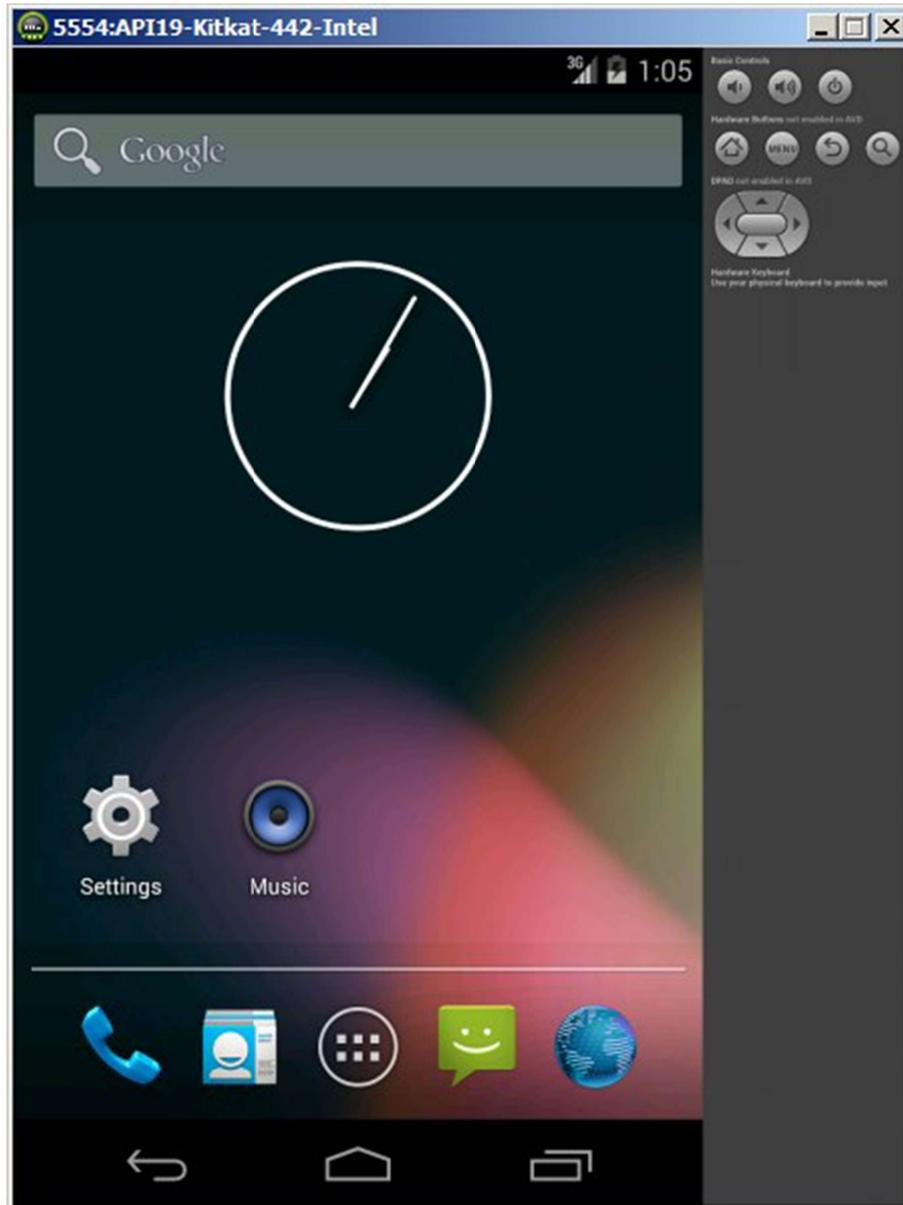
Creating a Virtual Device (AVD)

An AVD allows you to simulate devices and prototype your solution on a variety of SDKs. To create a virtual unit follow the next steps:

1. Click on the *AVD Manager* > Create. The **Create New AVD** wizard appears requesting your input.
2. Type the name of the emulator, enter a value such as “**API19-Kitkat-442-Intel**” (see figure on the right)
3. Select from the drop-downlist a Device (**Nexus 4...**) and CPU/ABI such as **Intel Atom (x86)**
4. Choose a target from the already installed SDKs (eg. “**Android 4.4.2 - API Level19**”).
5. Tick the *Keyboard* box to enable your PC’s keyboard.
6. Choose a skin of your preference (**...dynamic hard ...**)
7. Set memory RAM to no more than **768** MB
8. Indicate how much internal storage the simulator will use (**200MB**).
9. Add a small SD card (**9MB**)
9. Click **OK** to create the **AVD**.

A screenshot of the 'Edit Android Virtual Device (AVD)' dialog box. The dialog is titled 'Edit Android Virtual Device (AVD)' and contains several configuration options. The 'AVD Name' field is filled with 'API19-Kitkat-442-Intel'. The 'Device' dropdown is set to 'Nexus 4 (4.7", 768 x 1280: xhdpi)'. The 'Target' dropdown is set to 'Android 4.4.2 - API Level 19'. The 'CPU/ABI' dropdown is set to 'Intel Atom (x86)'. The 'Keyboard' section has a checked box for 'Hardware keyboard present'. The 'Skin' dropdown is set to 'Skin with dynamic hardware controls'. The 'Front Camera' and 'Back Camera' dropdowns are both set to 'None'. The 'Memory Options' section has 'RAM' set to '768' and 'VM Heap' set to '64'. The 'Internal Storage' section has a value of '200' and a unit dropdown set to 'MiB'. The 'SD Card' section has 'Size' set to '9' and a unit dropdown set to 'MiB', with a 'File' field and a 'Browse...' button. The 'Emulation Options' section has checkboxes for 'Snapshot' and 'Use Host GPU', both of which are unchecked. At the bottom, there is a checkbox for 'Override the existing AVD with the same name' which is also unchecked. The dialog has 'OK' and 'Cancel' buttons at the bottom right.

Setting up Eclipse + ADT + SDK



AVDs are saved in the folder:
c:\Users\yourName\.android\avd\API19-Kitkat-442-Intel.avd

```
File Edit Options Encoding Help
avd.ini.encoding=ISO-8859-1
abi.type=x86
disk.dataPartition.size=200M
hw.accelerometer=yes
hw.audioInput=yes
hw.battery=yes
hw.camera.back=none
hw.camera.front=none
hw.cpu.arch=x86
hw.dPad=no
hw.device.hash2=MD5:6930e145748b87e87d3f40cabd140a41
hw.device.manufacturer=Google
hw.device.name=Nexus 4
hw.gps=yes
hw.keyboard=yes
hw.lcd.density=320
hw.mainKeys=no
hw.ramSize=768
hw.sdCard=yes
hw.sensors.orientation=yes
hw.sensors.proximity=yes
hw.trackBall=no
image.sysdir.1=system-images\android-19\default\x86\
sdcard.size=9M
skin.dynamic=yes
skin.name=768x1280
skin.path=768x1280
tag.display=Default
tag.id=default
vm.heapSize=64
```

A summary of the AVD specs is saved in the /config.ini file

Setting up Eclipse + ADT + SDK



Creating a Virtual Device (AVD)

Some examples:



On top, a phone emulator running **IceCream 4.x** wearing a HVGA skin



Tablet showing **Honeycomb 3.x**

Gingerbread 2.3 running on a custom skin for Nexus-S. See page: <http://heikobehrens.net/2011/03/15/android-skins/>

Setting up Eclipse + ADT + SDK

Testing a Virtual Device (AVD)

1. Invoke the AVD Manager.
2. Choose an emulator, click **Start**.

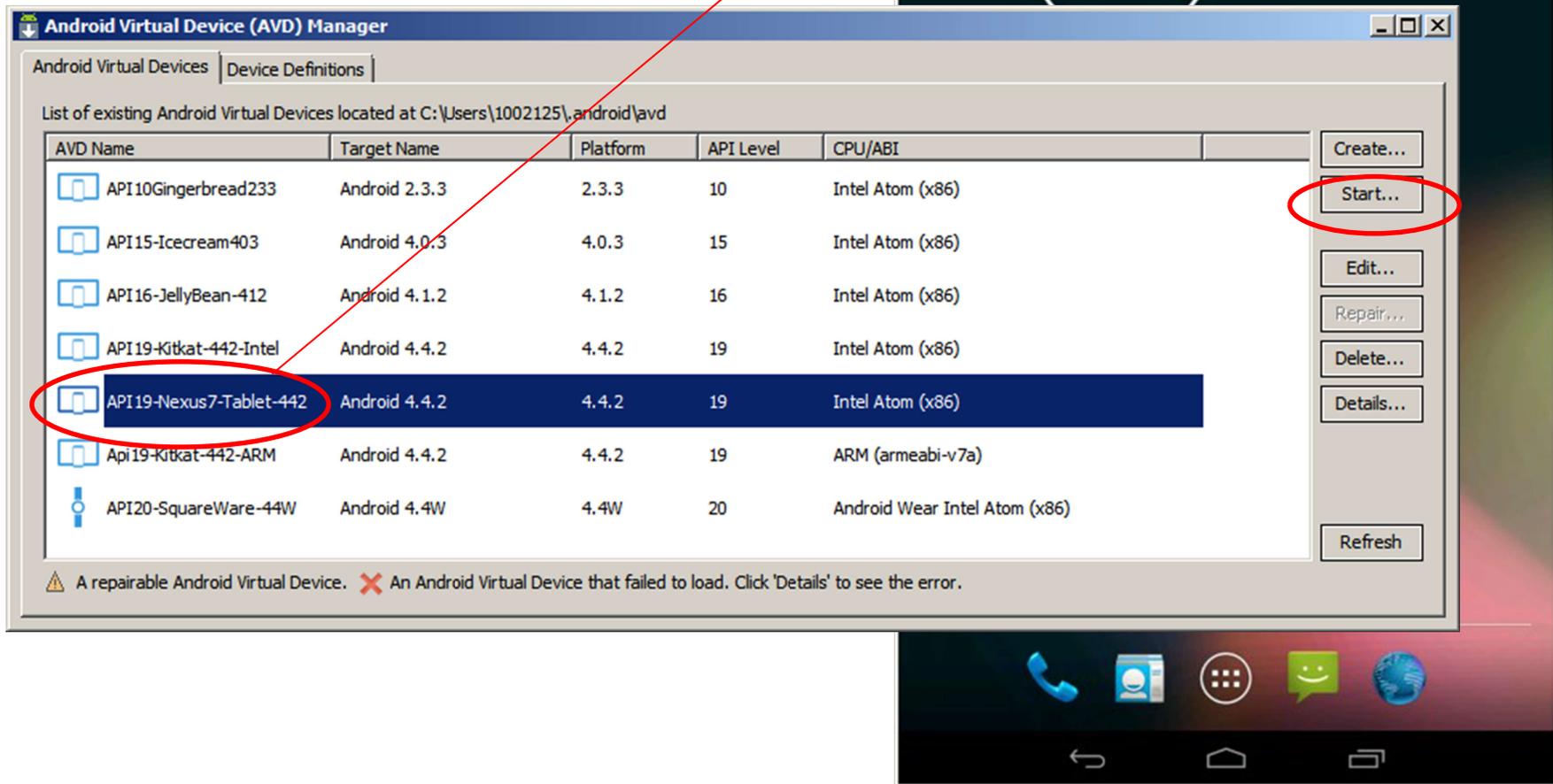
The image shows the Android Virtual Device (AVD) Manager interface and an Android emulator window. The AVD Manager window displays a list of existing virtual devices. The device 'API16-JellyBean-412' is selected and highlighted. The 'Start...' button is circled in red. The emulator window shows the Android home screen with the title bar '5556:API16-JellyBean-412' circled in red. A red arrow points from the 'Start...' button in the AVD Manager to the emulator window.

AVD Name	Target Name	Platform	API L...	CPU/ABI
API10Gingerbread233	Android 2.3.3	2.3.3	10	Intel Atom (x86)
API15-Icecream403	Android 4.0.3	4.0.3	15	Intel Atom (x86)
API16-JellyBean-412	Android 4.1.2	4.1.2	16	Intel Atom (x86)
Api19-Kitkat-442-ARM	Android 4.4.2	4.4.2	19	ARM (armeabi-v7a)
API19-Kitkat442-Intel	Google APIs (x86 System Image) (Google Inc.)	4.4.2	19	Intel Atom (x86)

Setting up Eclipse + ADT + SDK

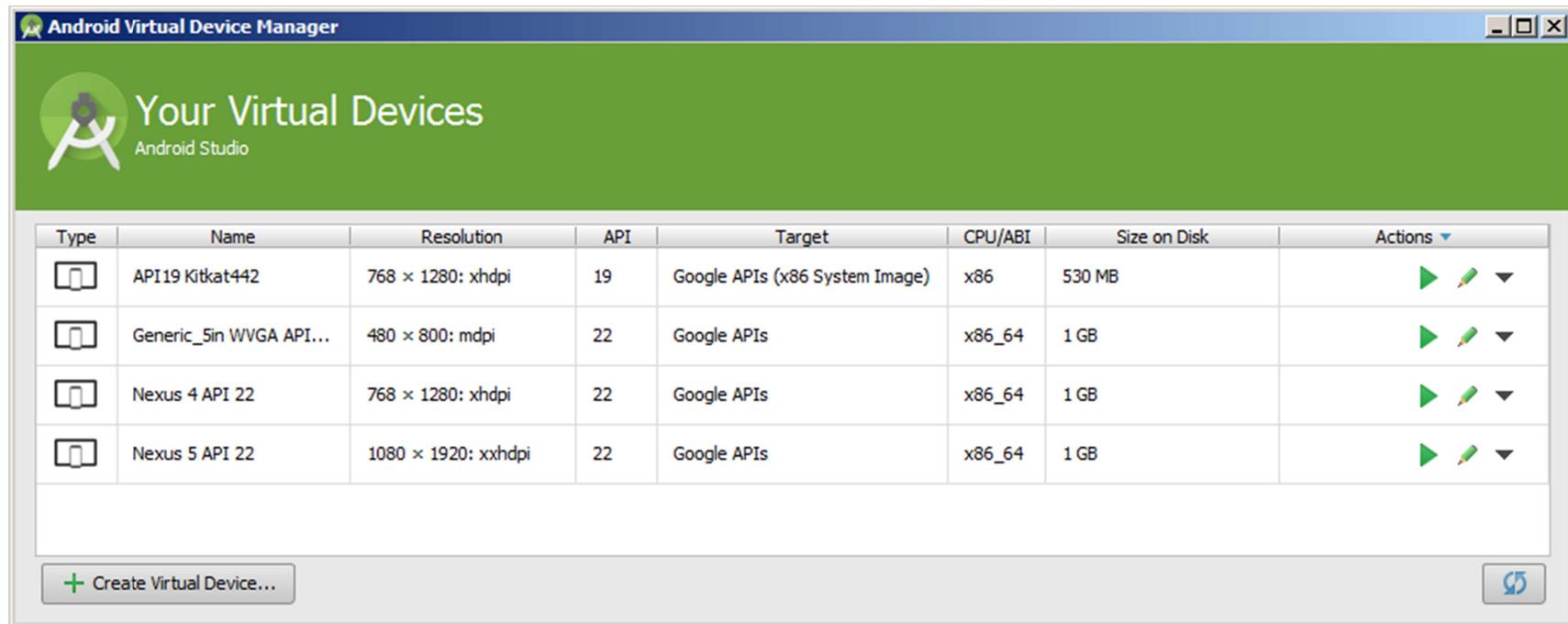
Running a Virtual Device (AVD)

1. Invoke the AVD Manager.
2. Choose an emulator, click **Start**.



Setting up Android Studio

Working with Virtual Devices (AVDs)



The Android Studio process to create, edit, remove, and execute AVDs is similar to the strategy already discussed for Eclipse-ADT (only cosmetic differences on the GUI)

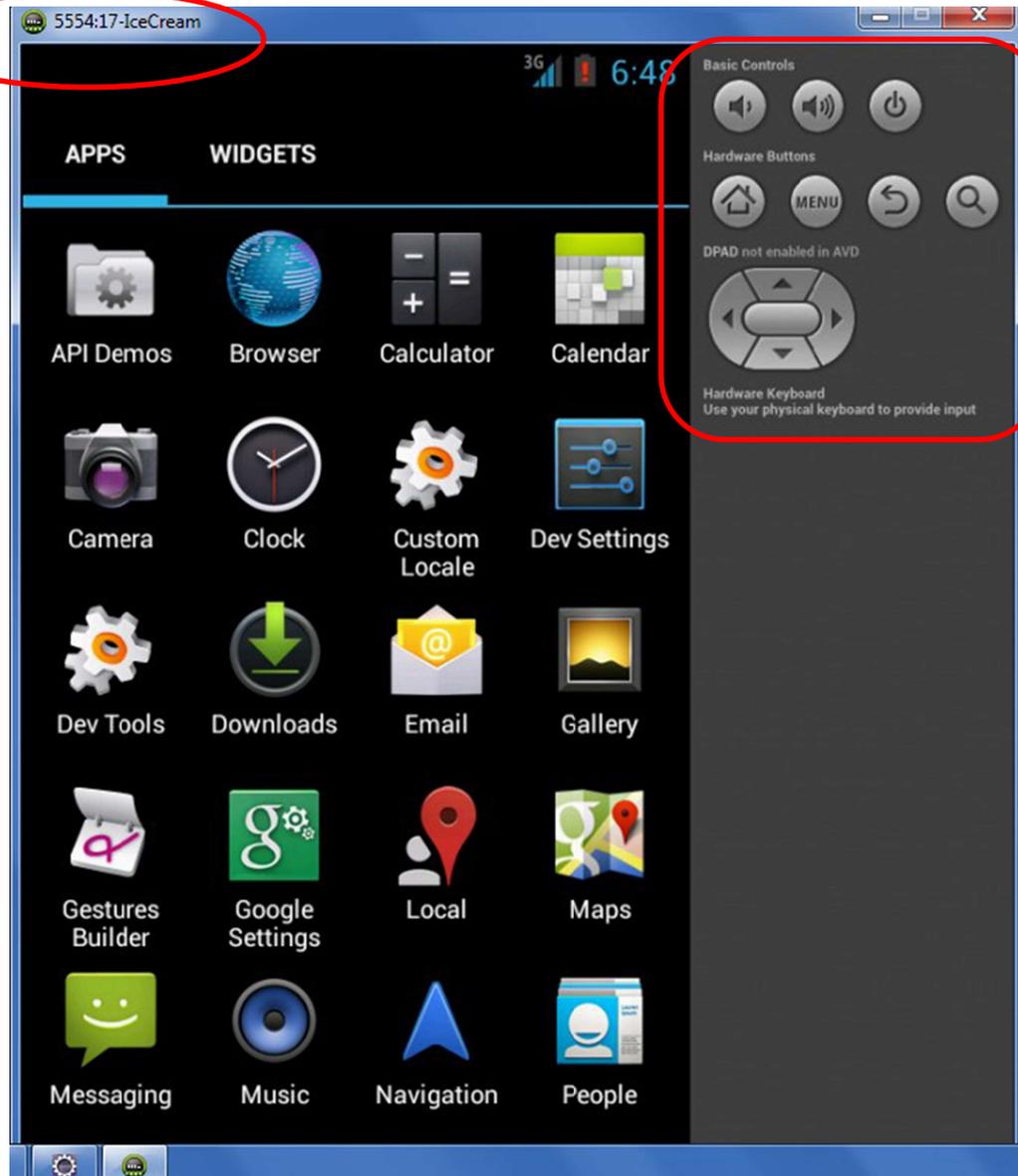
Example of an AVD Emulator wearing a HVGA Skin

ID number
5554



AVD – Emulator wearing: Skin with dynamic hardw. controls

Numeric ID:
5554

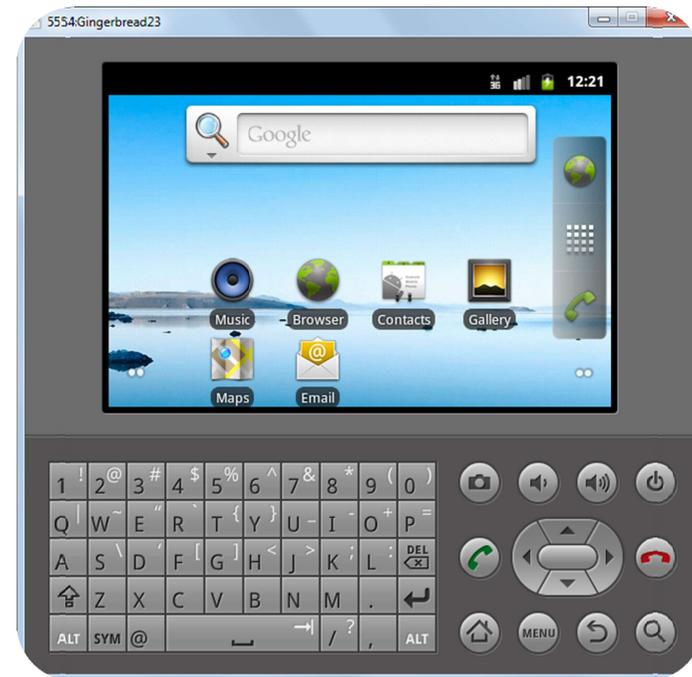


Controlling the AVD Operations

Keyboard	OS function
Escape	Back button
Home	Home button
F2, PageUp	Menu (Soft-Left) button
Shift-F2, PageDown	Start (Soft-Right) button
F3	Call/Dial button
F4	Hangup / EndCall button
F5	Search button
F7	Power button
Ctrl-F3, Ctrl-KEYPAD_5	Camera button
Ctrl-F5, KEYPAD_PLUS	Volume up button
Ctrl-F6, KEYPAD_MINUS	Volume down button
KEYPAD_5	DPad center
KEYPAD_4	DPad left
KEYPAD_6	DPad right
KEYPAD_8	DPad up
KEYPAD_2	DPad down
F8	toggle cell network on/off
F9	toggle code profiling
Alt-ENTER	toggle FullScreen mode
Ctrl-T	toggle trackball mode
Ctrl-F11, KEYPAD_7	switch to previous layout
Ctrl-F12, KEYPAD_9	switch to next layout

Controlling an Android Emulator through *your computer's* keyboard

Note: Keypad keys only work when *NumLock* is deactivated.



AVD – Emulator : Disk Images

Working with Emulator Disk Images

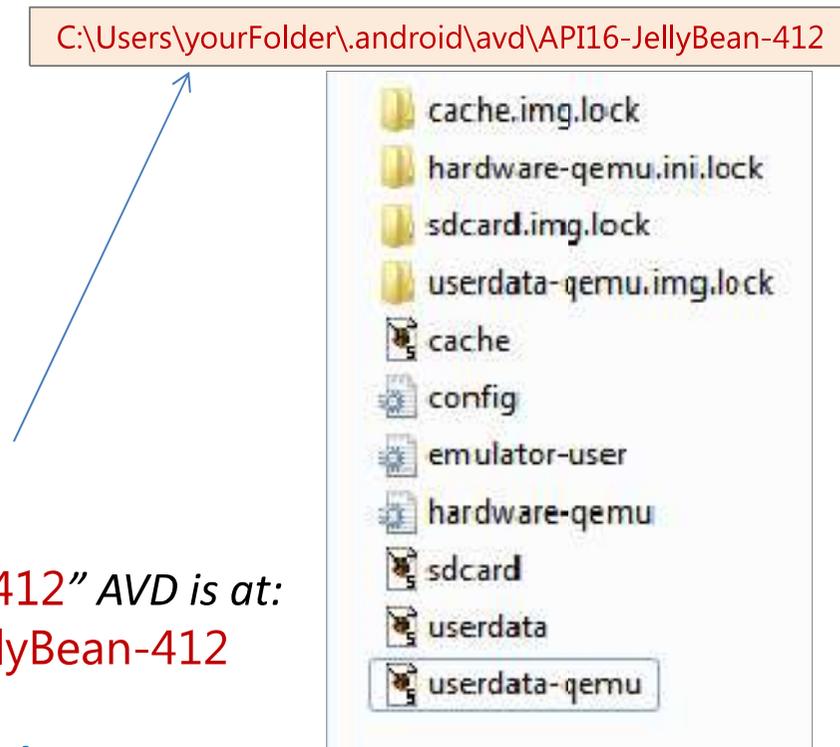
- *The Android simulator uses QEMU technology [Website: www.qemu.org]*
- *QEMU is an open source machine emulator which allows the operating system and programs made for one machine (e.g. an ARM CPU) run efficiently on a different machine (e.g. your Windows PC).*

When you create a **Virtual Device**, the SDK Makes several **disk images** containing among others:

- (1) OS kernel,
- (2) the Android system,
- (3) user data ([userdata-qemu.img](#))
- (4) simulated SD card ([sdcard.img](#)).

By default, the Emulator searches for the disk images in the private storage area of the AVD in use, for instance the “API16-JellyBean-412” AVD is at:
C:\Users\yourFolder\.android\avd\API16-JellyBean-412

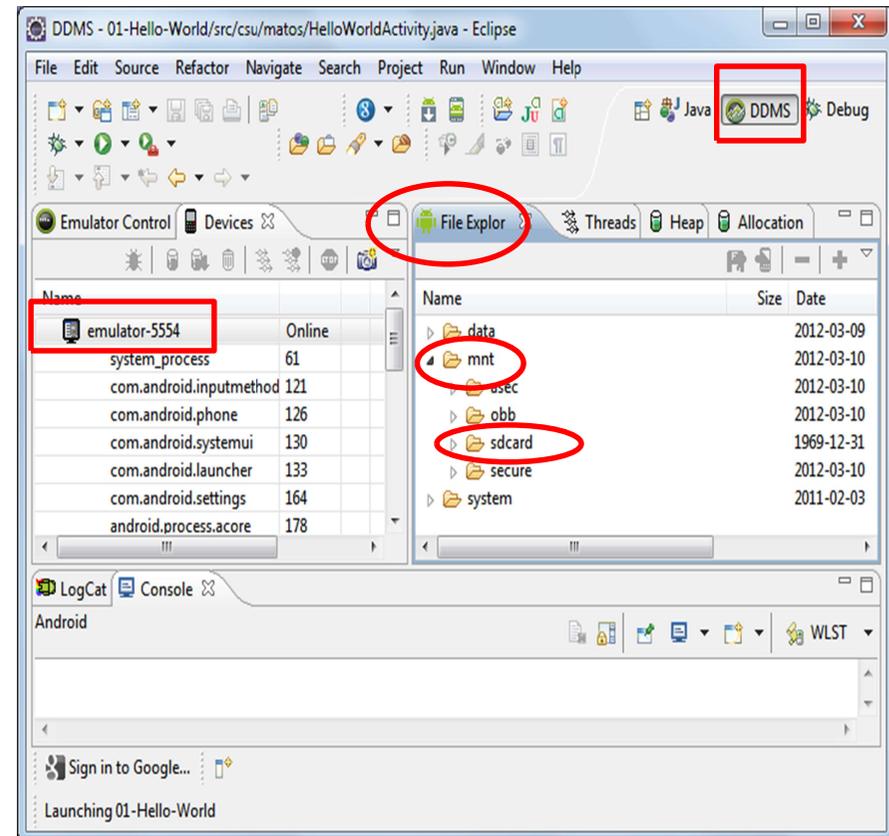
Mac OS users should look into **~/[.android/avd](#)**



Transferring Files to/from Emulator's SD Card

Upload/download Data, Music and Picture files to the Emulator's SDcard

1. Eclipse developers needs to add the **DDMS** perspective.
2. Android-Studio uses the equivalent '**Android Device Monitor**' button.
3. Change to the DDMS perspective. Make sure your AVD has started (You will see a layout similar to the figure on the lower right side)
4. Click on the **File Explorer** tab.
5. Expand the **mnt** (mounted devices) folder.
6. Expand the **sdcard** folder
7. Open your Window's **Explorer**.
8. Choose a file stored in your PC. Transfer a copy to the emulator by dragging and dropping it on top of the **sdcard** folder.



Transferring Files to/from Emulator's SD Card

Upload/download Data, Music and Picture files to the Emulator's SDcard

The screenshot shows the Eclipse IDE interface with the following components:

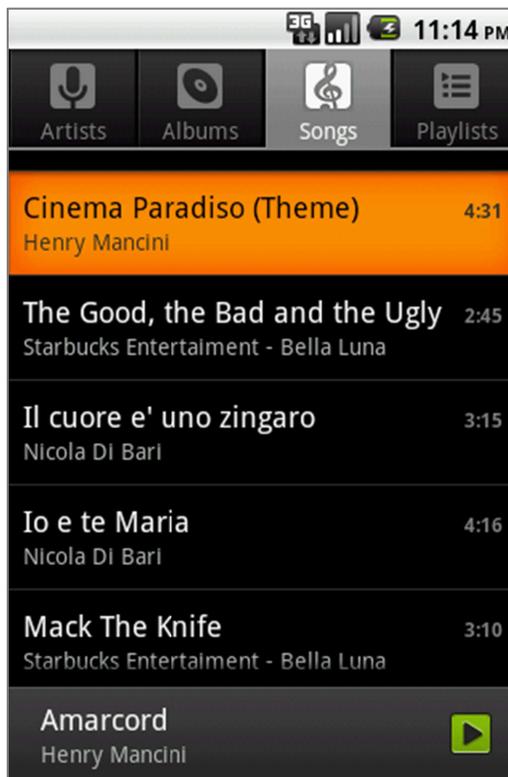
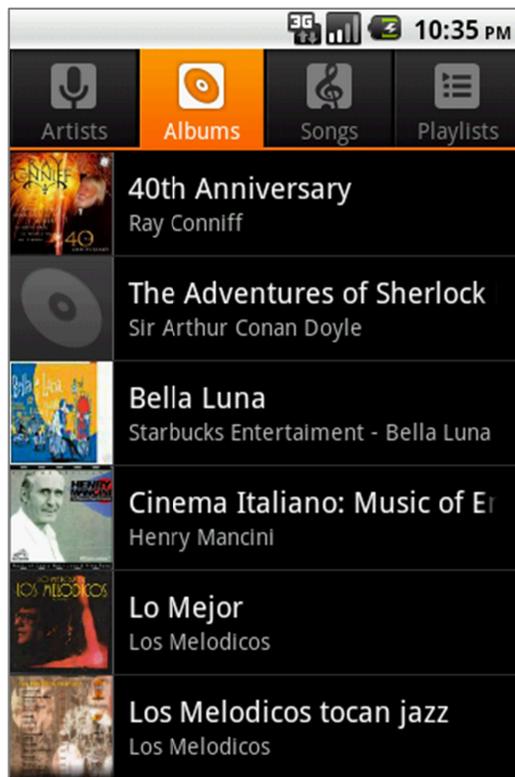
- File Explorer (Eclipse):** Located in the middle-right pane, it displays the file system of the emulator. The 'sdcard' folder is circled in red, and a red arrow points to it from the Windows File Explorer window. Other folders visible include 'data', 'mnt', 'asec', 'obb', 'secure', and 'system'.
- Device List:** Located in the middle-left pane, it shows a table of running processes. The 'emulator-5554' device is circled in red.
- Windows File Explorer:** Located on the right side, it shows the 'Sample Pictures' library. The 'Penquins' image is circled in red, and a red arrow points from it to the 'sdcard' folder in the Eclipse File Explorer.
- LogCat/Console:** Located at the bottom, it shows the text 'Launching 01-Hello-World'.

Name	PID	State
emulator-5554		Online
system_process	61	
com.android.inputmethod	121	
com.android.phone	126	
com.android.systemui	130	
com.android.launcher	133	
com.android.settings	164	
android.process.acore	178	

Transferring Files to/from Emulator's SD Card

Upload/download Data, Music and Picture files to the Emulator's SDcard

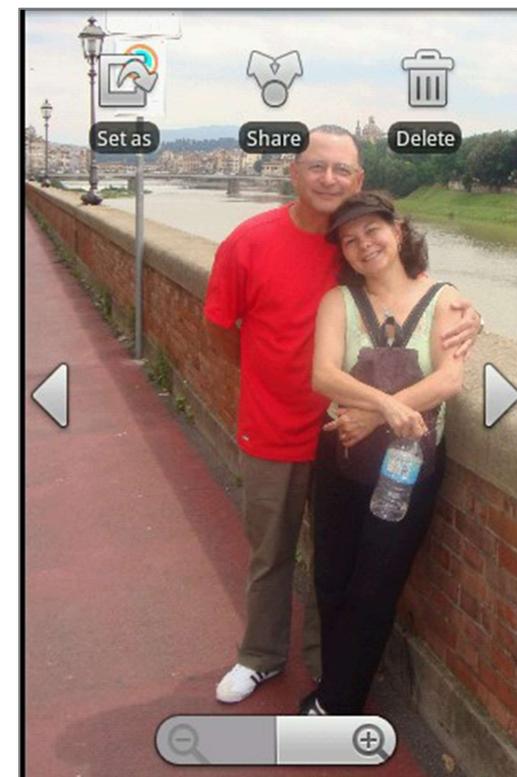
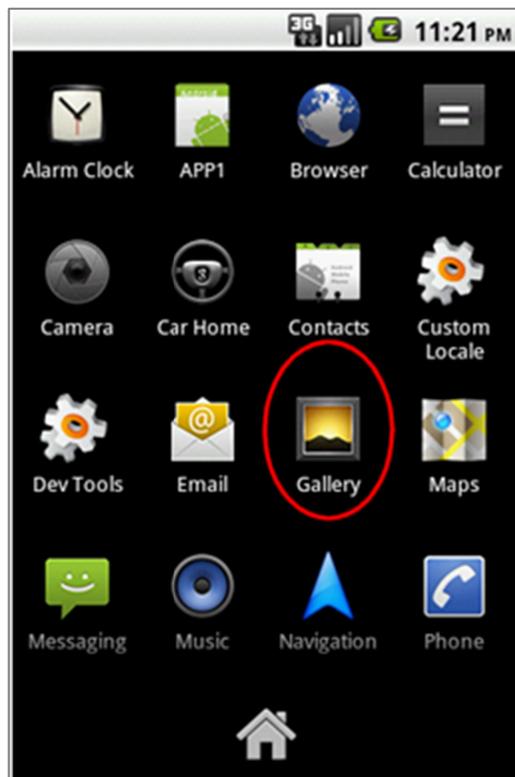
- Return to the emulator. This time you may use native apps such as 'Music' and 'Gallery' to see your recently uploaded multimedia files. For instance...



Transferring Files to/from Emulator's SD Card

Upload/download Data, Music and Picture files to the Emulator's SDcard

9. Pictures may be displayed by clicking the *Application Pad* and invoking the **Gallery** application

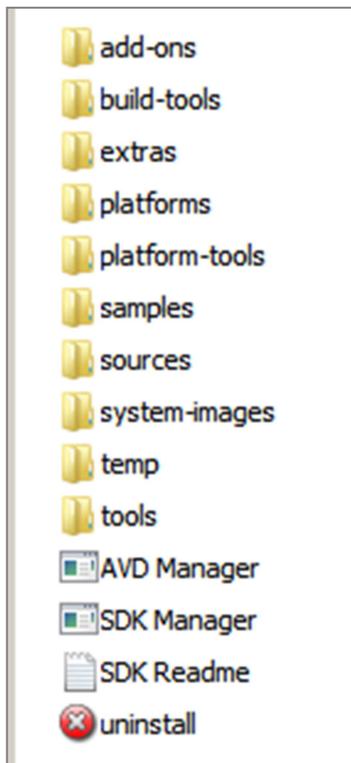


Setting up Eclipse + ADT + SDK

Locate your 'android-sdk' & AVD folder

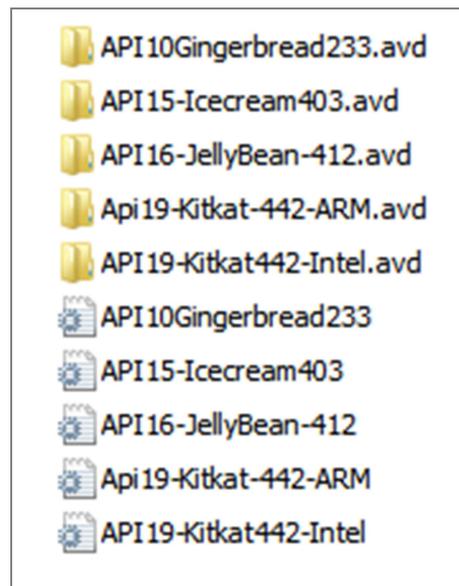
After you complete your setup look for the following two subdirectories in your PC's file system

C:\Program Files (x86)\Android\android-sdk



This folder contains your Android SDK, tools, and platforms

C:\Users\yourWindowsUserName\.android\avd



This directory holds your Virtual Devices (AVDs)

Android Studio: Hello World App

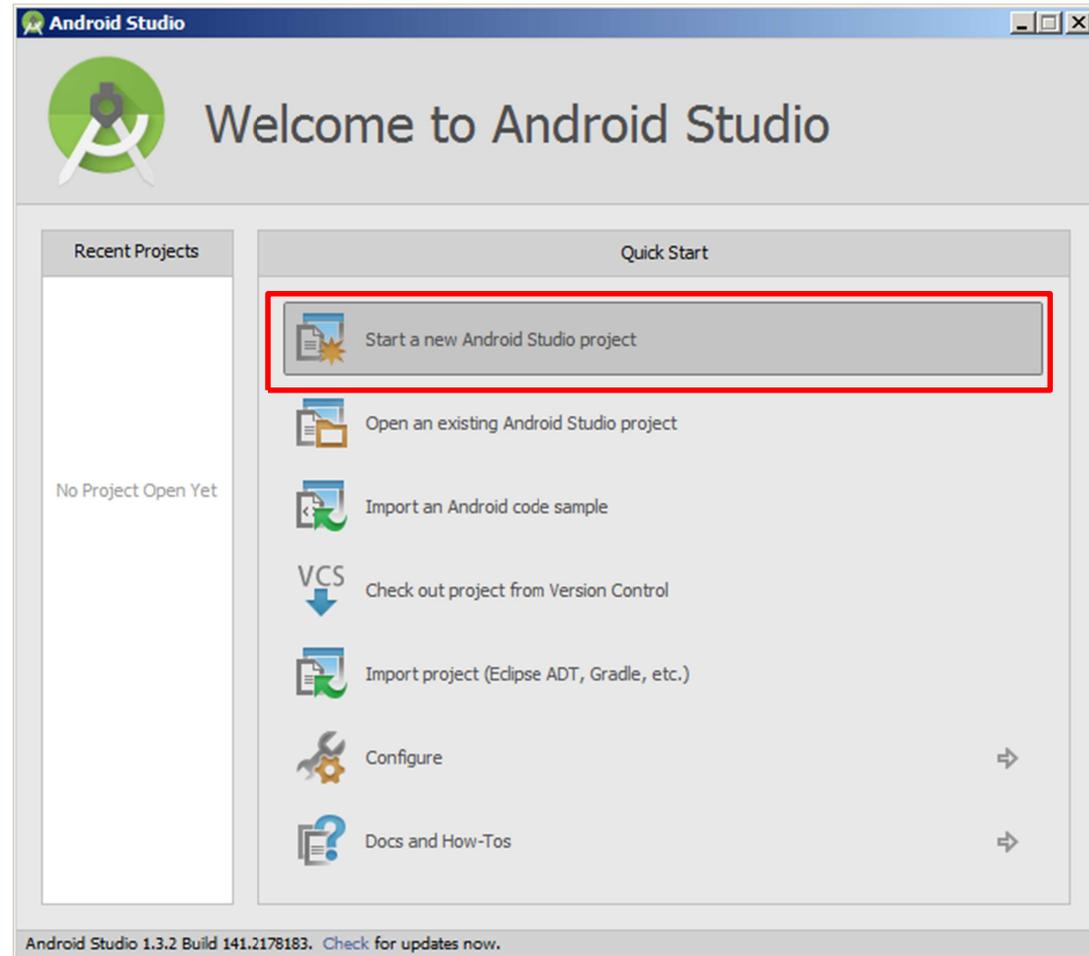
Example 2.1 : HelloWorld App

We will use **Android Studio IDE** to create a bare bone app.

Click on the entry: *'Start new Android Studio Project'* .

A wizard will guide you providing a sequence of menu driven selections.

The final product is the skeleton of your Android app.



Android apps are usually made of a rich collection of various type of components including Java code, multimedia resources, XML files, etc. The *New Android Studio Project Wizard* facilitates the assembly of those parts and organizes the components in various sub-directories.

Android Studio: Hello World App

Example 2.1 : HelloWorld App

1. Enter in the *Application Name* box:
HelloApp
2. Enter *Company Domain*: **csu.matos**
(usually a dot-separated string consisting of company and programmer's name)
3. Click **Next**

The screenshot shows the 'Create New Project' dialog in Android Studio. The dialog has a green header with the Android Studio logo and the text 'New Project Android Studio'. Below the header, the text 'Configure your new project' is displayed. There are three input fields: 'Application name' with the value 'HelloApp', 'Company Domain' with the value 'csu.matos', and 'Package name' with the value 'matos.csu.helloapp'. The 'Package name' field has an 'Edit' link to its right. Below these fields is a 'Project location' field with the value 'C:\Users\1002125\AndroidStudioProjects\HelloApp' and a browse button (...). At the bottom of the dialog, there are four buttons: 'Previous', 'Next', 'Cancel', and 'Finish'. The 'Next' button is highlighted with a red box.

Android Studio: Hello World App

Example 2.1 : HelloWorld App

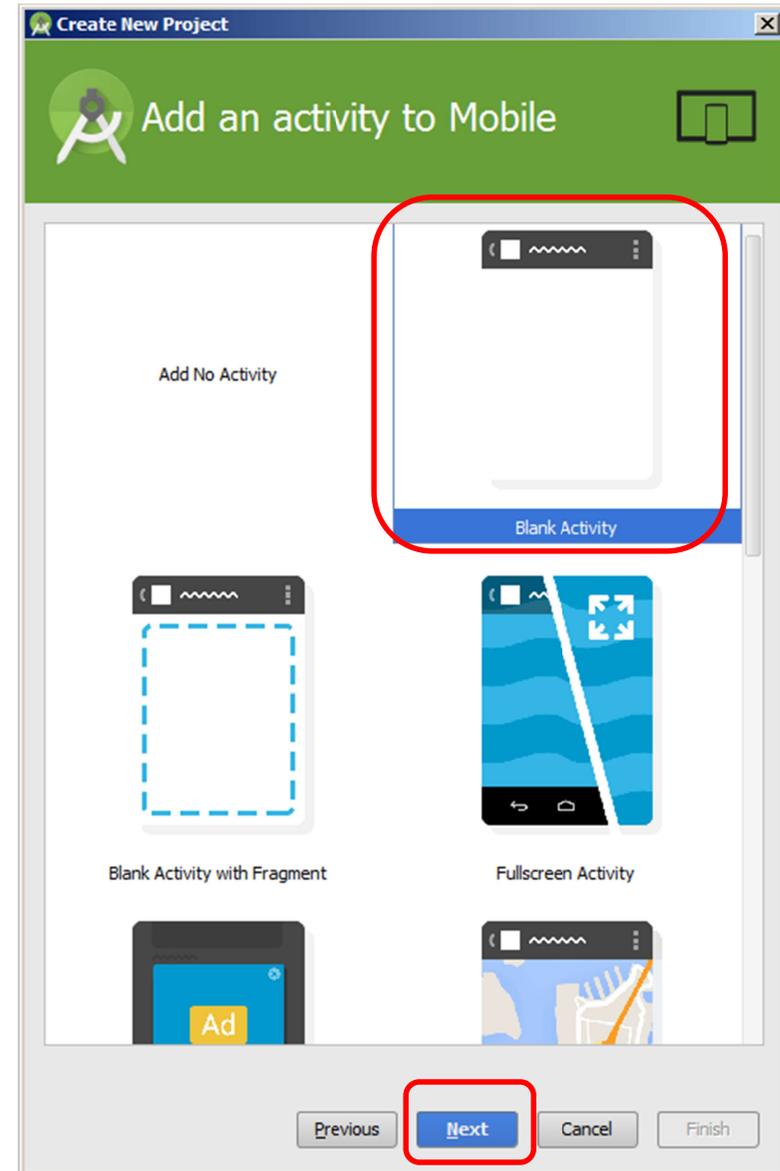
4. Select Target Android Device. In this example **Phone and Tablet** is already checked. Other options are: Wear, TV, Auto, Glasses.
5. Choose from drop-down list the Minimum SDK on which the app will work. In this example we have selected: **API22 Android 5.1 (Lollipop)** (**Lollipop**)
6. Click **Next**

The screenshot shows the 'Create New Project' dialog in Android Studio. The title bar reads 'Create New Project'. Below the title bar is a green header with the Android logo and the text 'Target Android Devices'. The main content area is titled 'Select the form factors your app will run on' and includes the subtitle 'Different platforms may require separate SDKs'. There are four radio button options: 'Phone and Tablet' (checked), 'Wear', 'TV', and 'Glass (Not Installed)'. Each option has a 'Minimum SDK' dropdown menu. The 'Phone and Tablet' dropdown is set to 'API 22: Android 5.1 (Lollipop)'. Below the 'Phone and Tablet' dropdown, there is a warning message: 'Lower API levels target more devices, but have fewer features available. By targeting API 22 and later, your app will run on < 1% of the devices that are active on the Google Play Store.' with a 'Help me choose' link. At the bottom of the dialog, there are four buttons: 'Previous', 'Next', 'Cancel', and 'Finish'. The 'Next' button is highlighted with a red box.

Android Studio: Hello World App

Example 2.1 : HelloWorld App

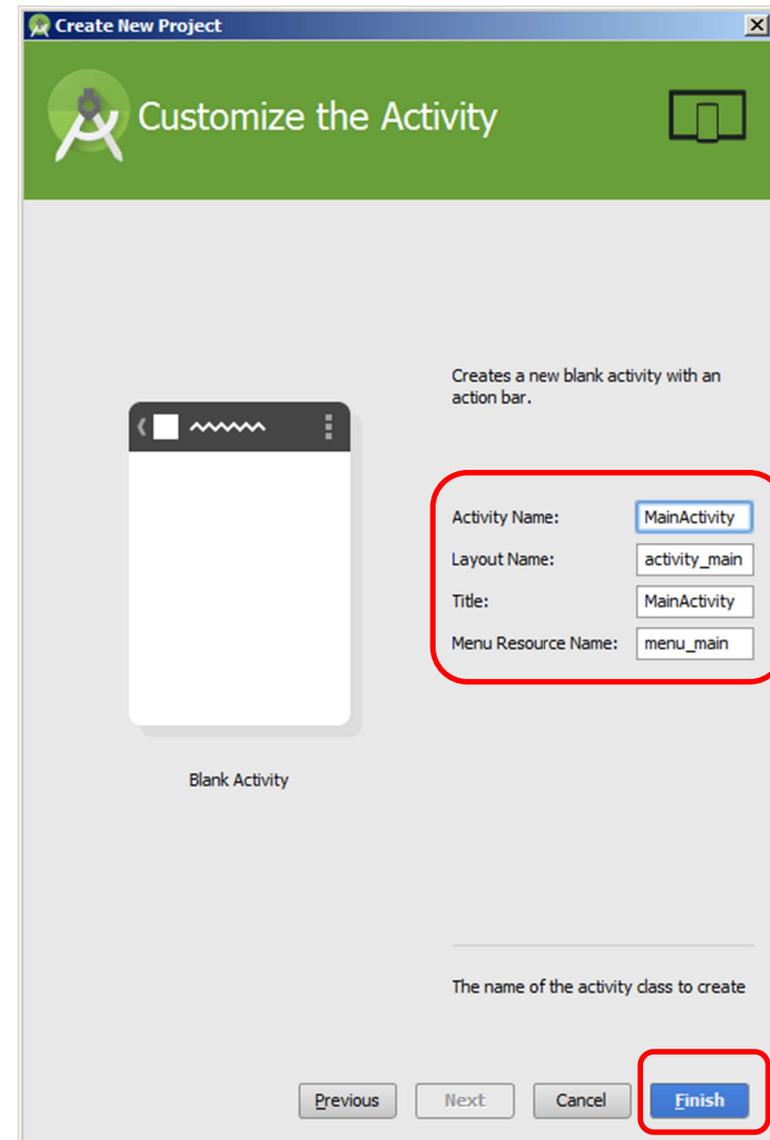
7. Select the pre-defined app template to apply. In this example we choose: **Blank Activity**
8. Click **Next**



Android Studio: Hello World App

Example 2.1 : HelloWorld App

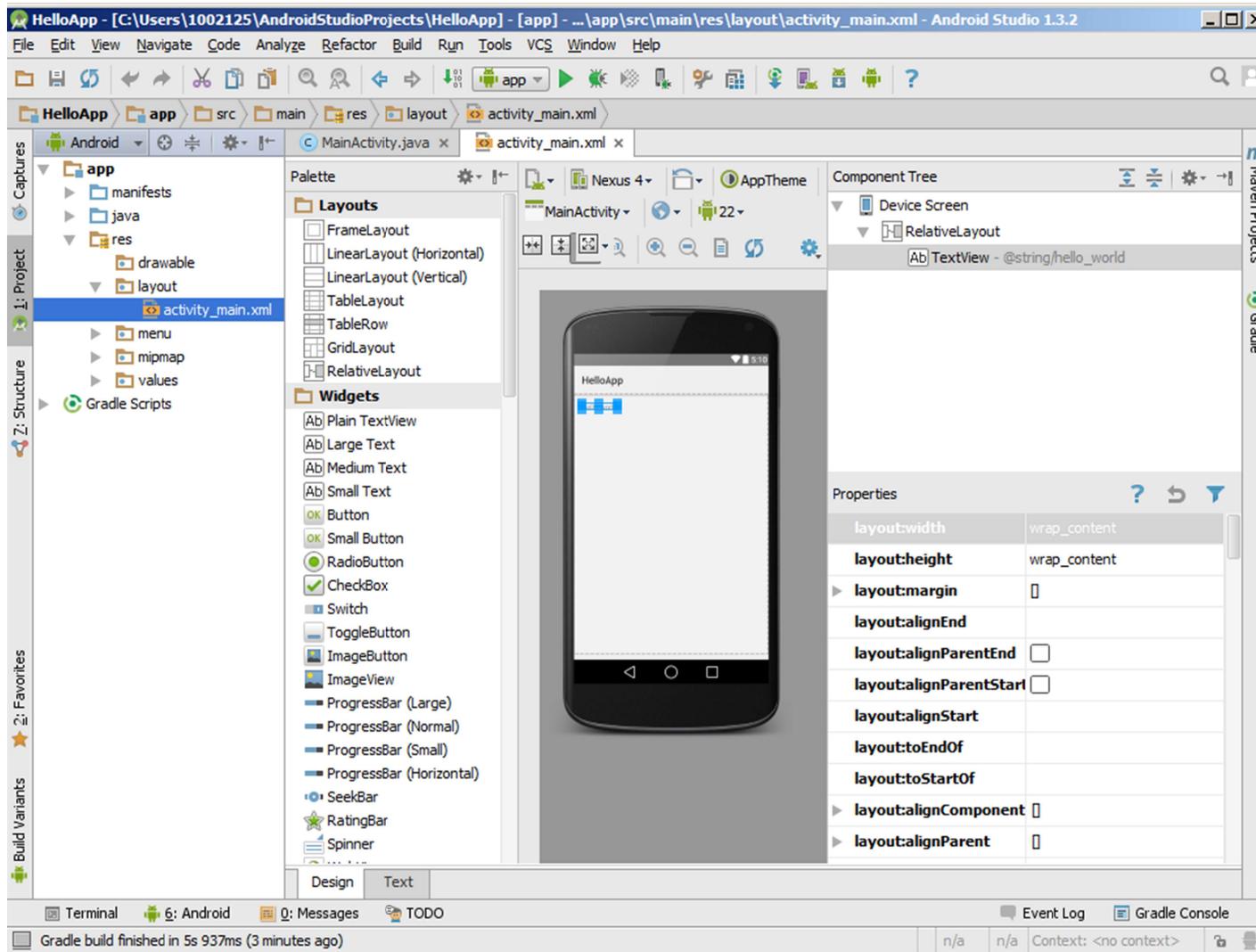
9. The wizard is ready to construct the solution. The text-boxes give you an opportunity to change any of the default names given to the main activity, the app's layout, its title, and menu. *Please do not change anything now.*
10. Click **Finish**
11. You are done! (your next step is to try the app on the emulator – explained later in this lesson)



Android Studio: Hello World App

Example 2.1 : HelloWorld App

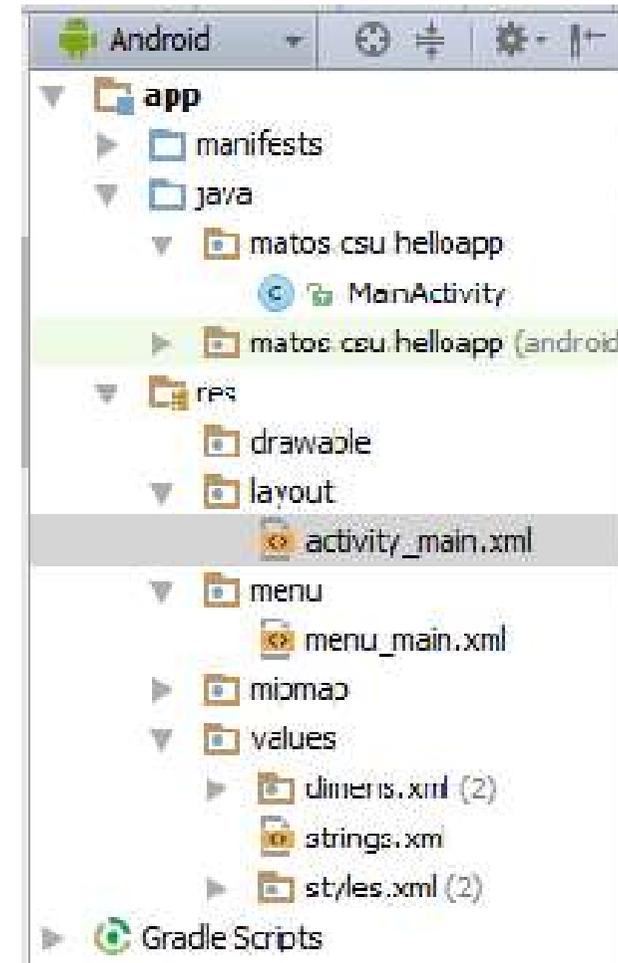
The app's GUI and the Palette (graphical toolbox) are shown. On the left pane, the Project Explorer shows the application's file structure.



Android Studio: Hello World App

Example 2.1 : HelloWorld App

- **Java/** Holds your Main-Activity Java code. All other Java files for your application go here.
- **res/** This folder stores application resources such as *drawable* files, *UI layout* files, *string* values, *menus*, multimedia, etc.
- **manifests** The Android Manifest for your project.



Android Studio: Hello World App

Example 2.1 : HelloWorld App – Java Code: MainActivity.java

```
package matos.csu.helloapp;
import ...

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }

        return super.onOptionsItemSelected(item);
    }
}
```

Android Studio: Hello World App

Example 2.1 : HelloWorld App - Layout: activity_main.xml

The screenshot displays the Android Studio interface. On the left, the XML editor shows the following code for activity_main.xml:

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    tools:context=".MainActivity">

    <TextView
        android:text="Hello world!"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

</RelativeLayout>
```

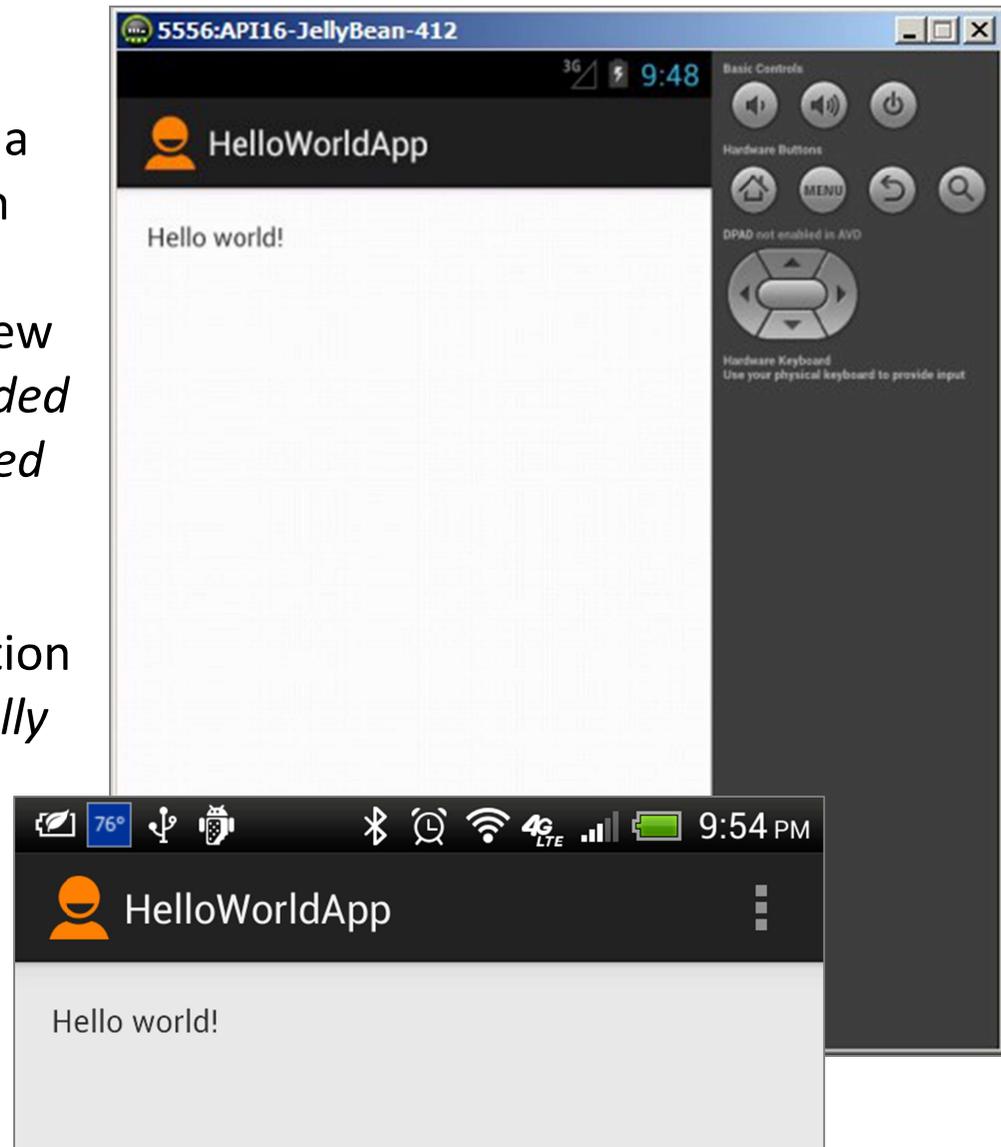
At the bottom left, the 'Design' and 'Text' tabs are visible, with 'Design' selected. On the right, the 'Preview' window shows a Nexus 4 device with the app running. The device screen displays 'HelloApp' at the top and 'Hello world!' in the center.

Eclipse: Using the 'New Android Application' Wizard

Example2.1 (again...) : HelloWorld App

We will use **Eclipse + ADT** to create a bare bone app. All it is needed from the developer is to feed the *New Android Application* wizard with a few selections (*no extra code will be added to the default app skeleton generated by the IDE+SDK*).

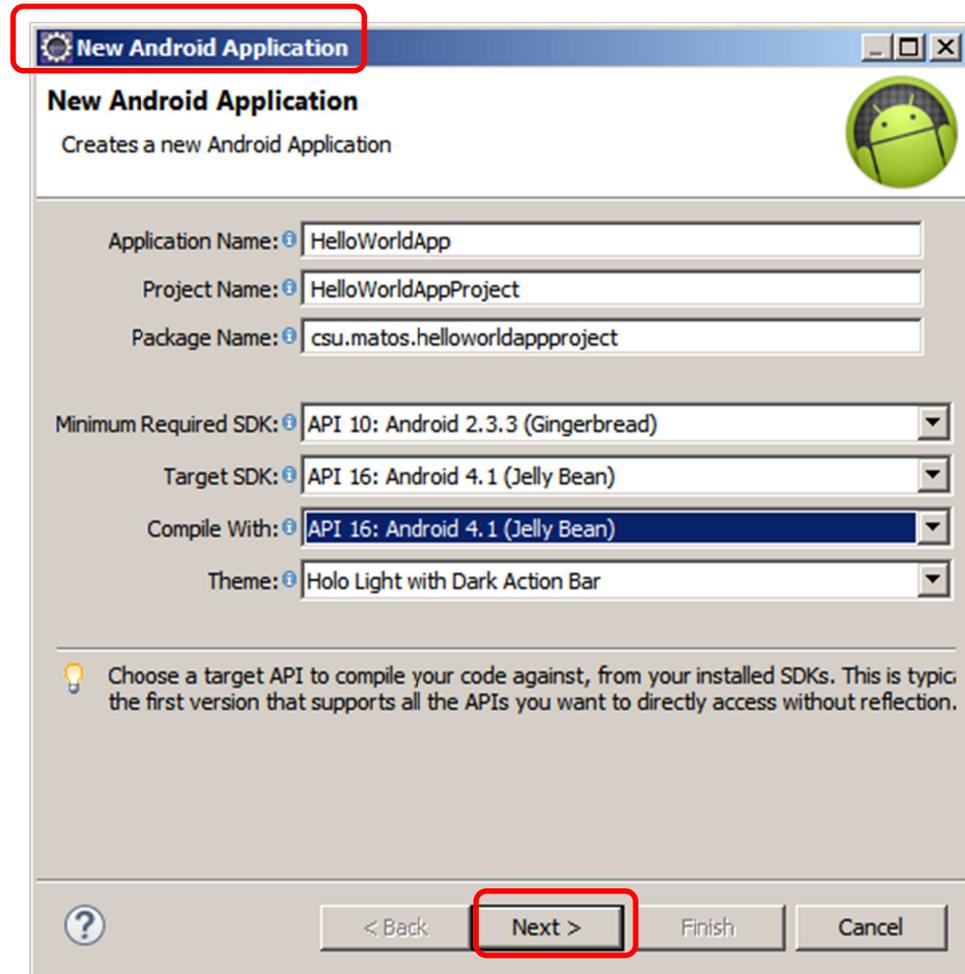
The adjacent figures show the solution made by the wizard running on a *Jelly Bean* emulator and device.



Eclipse: Using the 'New Android Application' Wizard

Example : HelloWorld App

1. Start **Eclipse**
2. From menu choose **File > New > Android Application Project**
3. Enter in the *Application Name* box:
HelloWorldApp
4. Enter Project name:
HelloWorldAppProject
5. Modify *Package Name* prefix to:
csu.matos.helloworldappproject
6. For *Minimum Required SDK* choose:
API 10: Android 2.3.3 (Gingerbread)
7. For *Target SDK* select the option:
API 16:Android 4.1 (Jelly Bean)
8. Select for *Compile With* the option:
API 16:Android 4.1 (Jelly Bean)
9. Click **Next**
10. Click **Next**

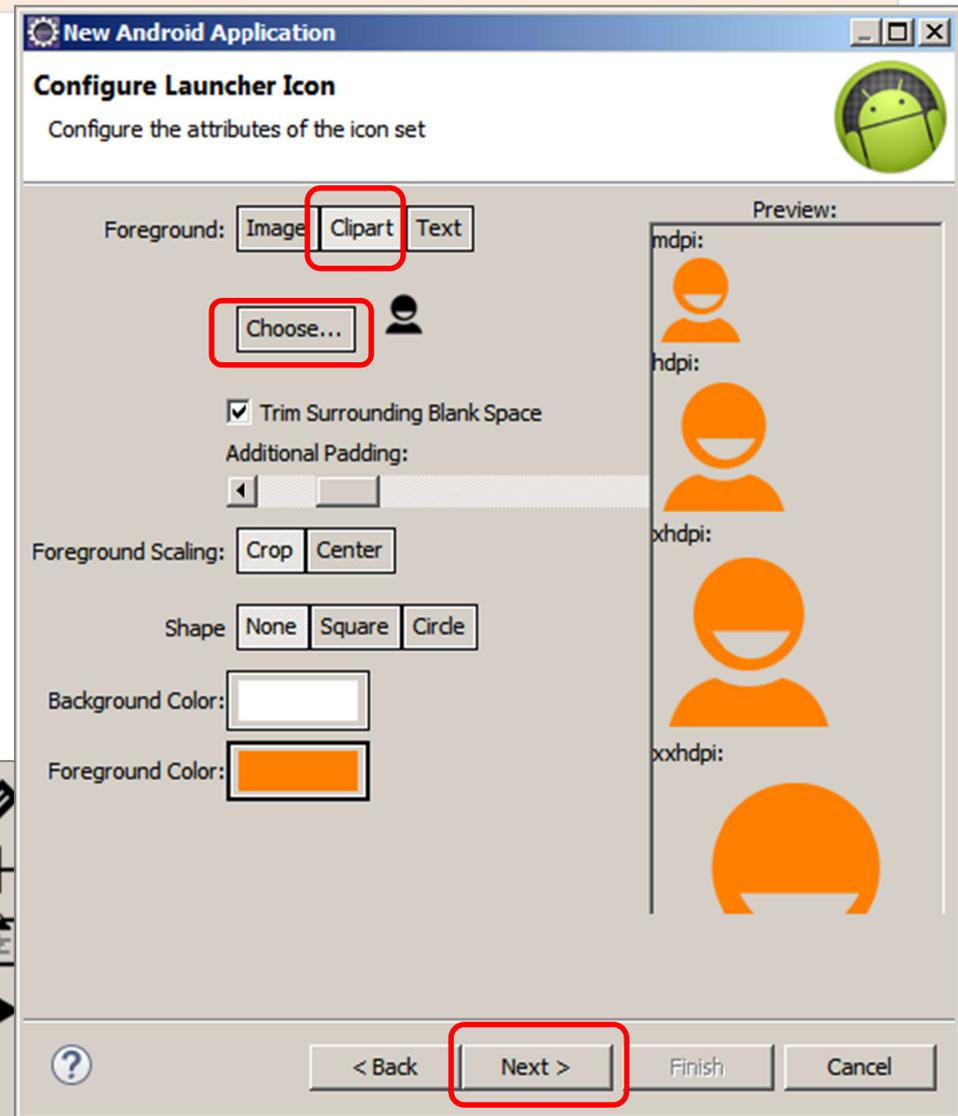
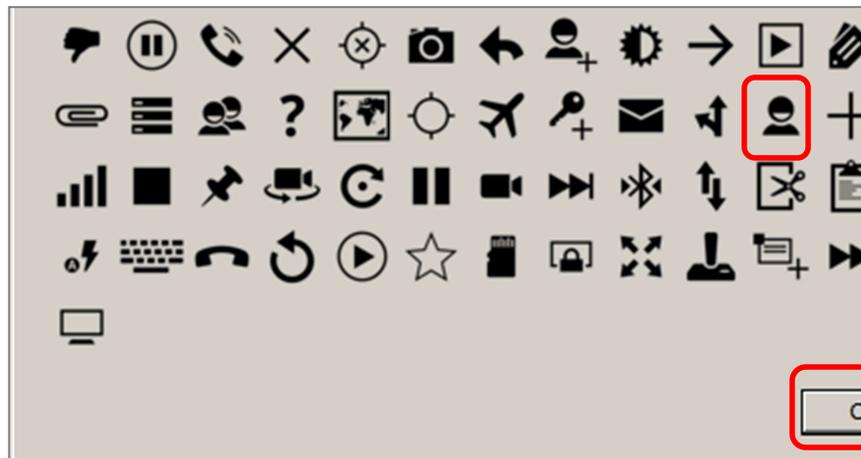


Eclipse: Using the 'New Android Application' Wizard

Example : HelloWorld App

On the form **Configure Launcher Icon** do the following:

11. Foreground > **Clipart** > **Choose**
12. Select an icon from the set of available images > **Close**
13. Pick a *Foreground Color*
14. Click **Next**

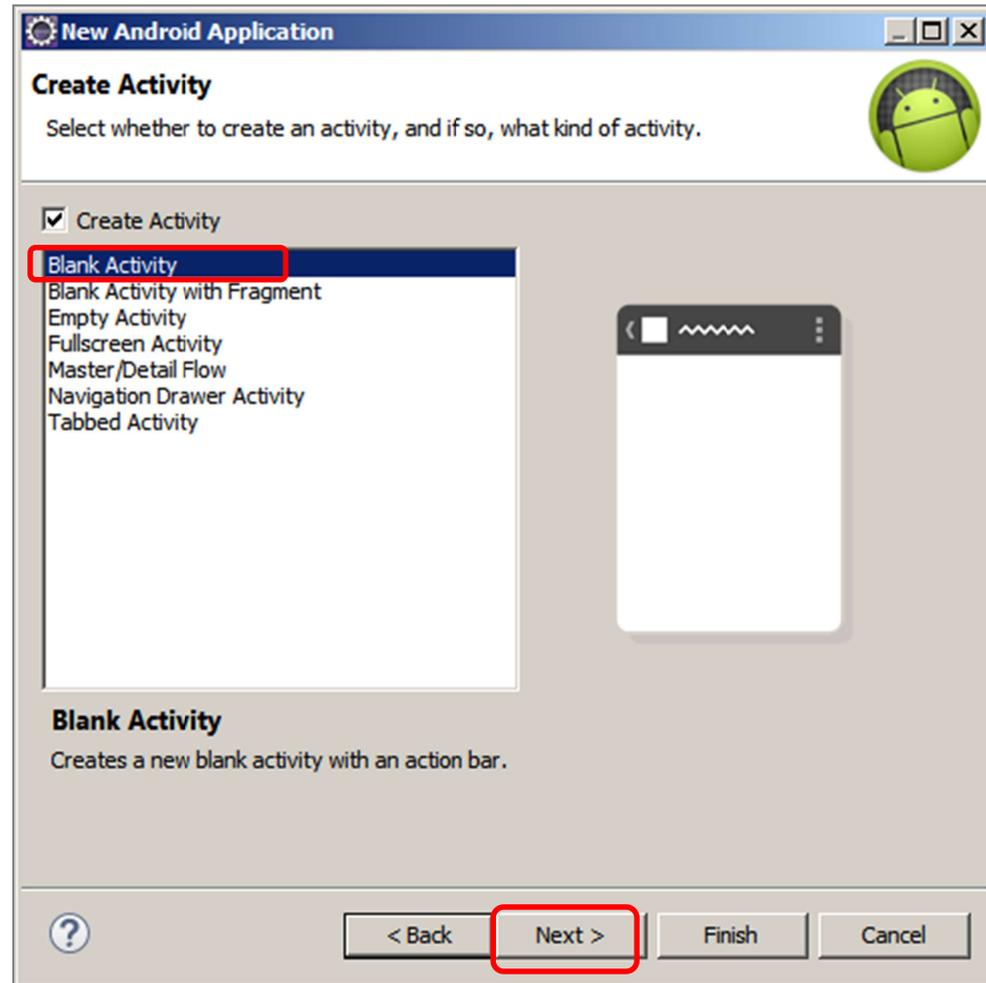


Eclipse: Using the 'New Android Application' Wizard

Example : HelloWorld App

The **Create Activity** form provides a number of basic templates from which your application could be constructed.

15. Select the **Blank Activity** template.
16. Click **Next**.



Eclipse: Using the 'New Android Application' Wizard

Example : HelloWorld App

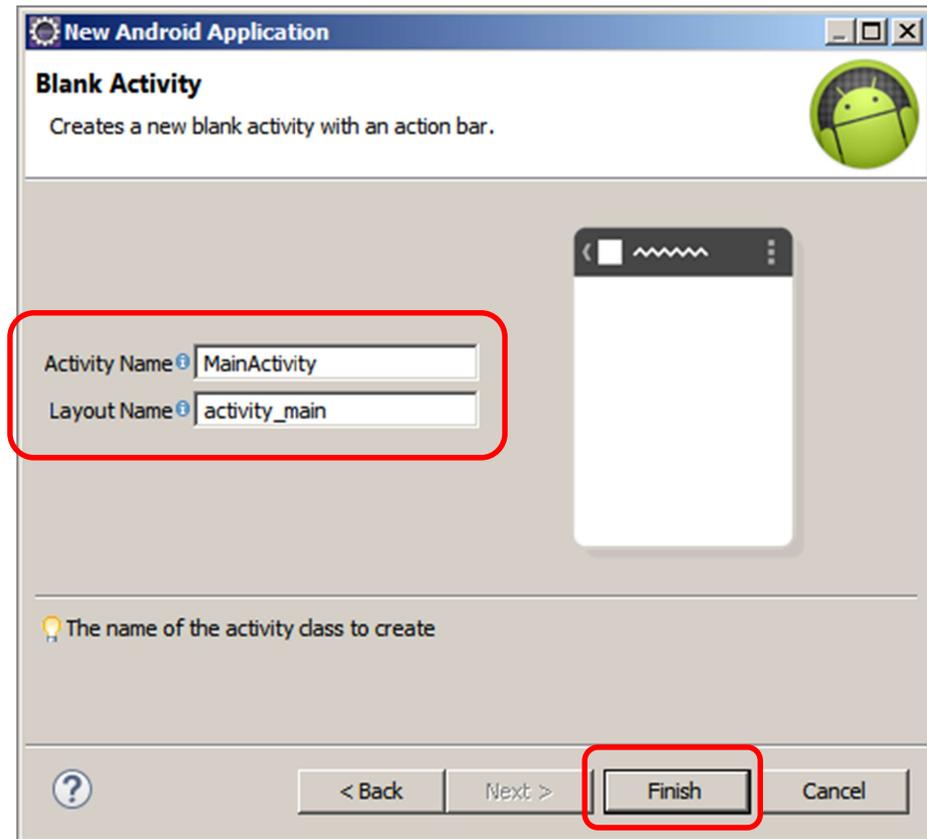
The **Blank Activity** form provides a way to name the main Activity and Layout name.

17. Leave the default values shown in the form (Activity Name and Layout Name).

18. Click **Finish**.

At this point the wizard has completed all the steps required to make the app.

After a few seconds the Eclipse perspective shows the app's UI. The Java solution is shown in the PackageExplorer pane (see next pages)



Eclipse: Using the 'New Android Application' Wizard

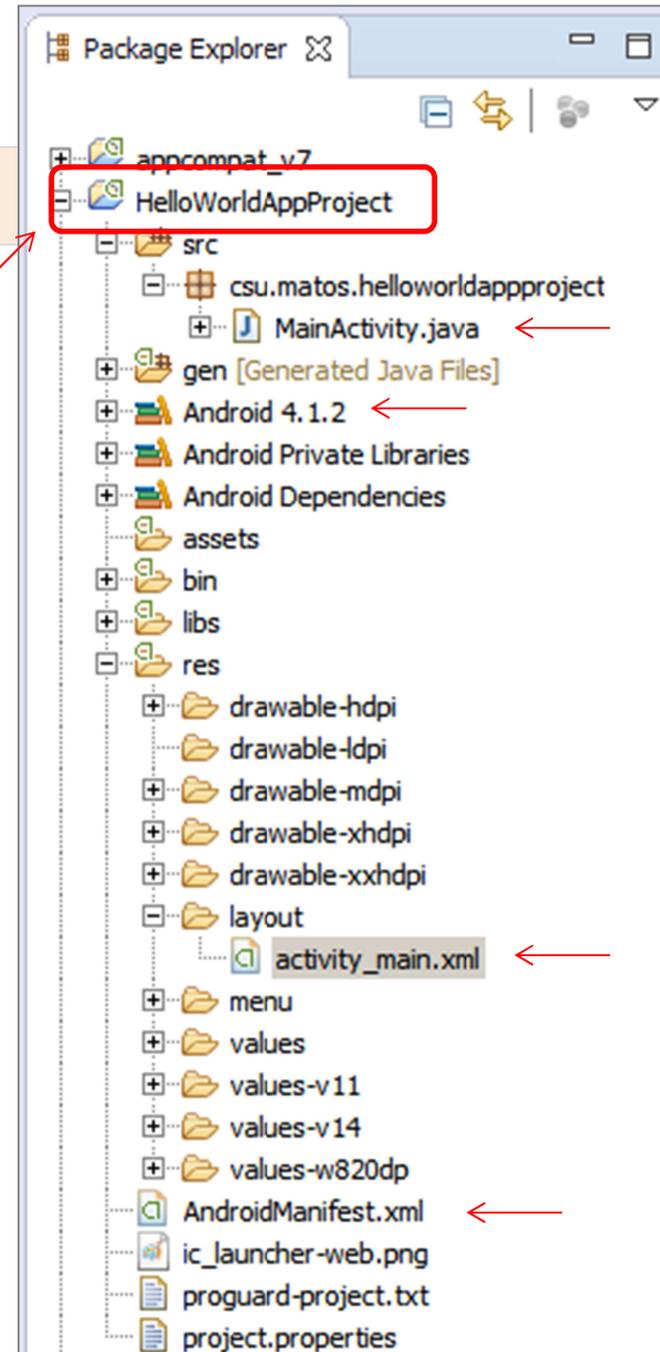
Example : HelloWorld App

File Structure

The folders and files shown on the figure are part of the newly created app.

Here we are using Eclipse's *Package Explorer* facility to navigate inside the folder holding the app.

To test the application, position the cursor on the code panel, and then click on the *Run* menu button. 



Eclipse: Using the 'New Android Application' Wizard

File Structure of a Typical Android App

- **src/** Includes your skeleton Activity Java file. All other Java files for your application go here.
- **<Android Version>/** (e.g., Android 4.1/) Includes the android.jar file that your application will build against.
- **gen/** This contains the Java files generated by ADT, such as your R.java file
- **assets/** This is empty. You can use it to store raw asset files.
- **res/** This folder holds application resources such as *drawable* files, UI *layout* files, *string* values, etc.
- **bin/** The bytecode (.apk) version of your app is stored here
- **AndroidManifest.xml** The Android Manifest for your project.
- **default.properties** This file contains project settings, such as the build target.

Android Emulator – Looking Under the Hood



Login into the Android OS shell

- Although it is *not* necessary, a developer may gain access to some of the innermost parts of the Android OS.
- For a UNIX-like experience you can log into the system by executing the emulator and issuing selected shell commands.

```
C:\windows\system32\cmd.exe - adb shell
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Program Files (x86)\Android\android-sdk\platform-tools>adb shell
# ls -l
ls -l
dr-x----- root    root           2012-03-10 00:01 config
drwxrwx--- system  cache         2012-03-10 10:33 cache
lrwxrwxrwx root    root           2012-03-10 00:01 sdcard -> /mnt/sdcard
drwxr-xr-x root    root           2012-03-10 00:01 acct
drwxrwxr-x root    system        2012-03-10 00:01 mnt
lrwxrwxrwx root    root           2012-03-10 00:01 vendor -> /system/vendor
lrwxrwxrwx root    root           2012-03-10 00:01 d -> /sys/kernel/debug
lrwxrwxrwx root    root           2012-03-10 00:01 etc -> /system/etc
-rw-r--r-- root    root           3764 1969-12-31 19:00 ueventd.rc
-rw-r--r-- root    root            0 1969-12-31 19:00 ueventd.goldfish.rc
drwxr-xr-x root    root           2011-02-03 18:01 system
drwxr-xr-x root    root           1969-12-31 19:00 sys
drwxr-xr-x root    root           1969-12-31 19:00/sbin
drwxr-xr-x root    root           1969-12-31 19:00/proc
-rwxr-xr-x root    root          13805 1969-12-31 19:00 init.rc
-rwxr-xr-x root    root           1677 1969-12-31 19:00 init.goldfish.rc
-rwxr-xr-x root    root          94168 1969-12-31 19:00 init
-rw-r--r-- root    root            118 1969-12-31 19:00 default.prop
drwxrwxr-x system  system        2012-03-09 23:02 data
drwx----- root    root           2010-01-27 19:59 root
drwxr-xr-x root    root           2012-03-10 00:02 dev
# df
df
Filesystem      Size  Used  Free  Blksize
/dev            125M  32K  125M  4096
/mnt/asec       125M   0K  125M  4096
/mnt/obb        125M   0K  125M  4096
/system         96M   96M   0K   4096
/data           64M   32M   31M  4096
/cache          64M    1M   62M  4096
/mnt/sdcard     1019M 164M  855M  2048
/mnt/secure/asec 1019M 164M  855M  2048
# cd sdcard
cd sdcard
# ls -l
ls -l
d---rwxr-x system  sdcard_rw     2012-03-09 23:03 LOST.DIR
d---rwxr-x system  sdcard_rw     2012-03-10 19:59 DCIM
---rwxr-x system  sdcard_rw     5239976 2012-03-09 23:10 Amarcord.mp3
d---rwxr-x system  sdcard_rw     2012-03-09 23:11 Android
---rwxr-x system  sdcard_rw     263230 2012-03-09 23:29 Bea-Strada-Volterra-12X17.jpg
---rwxr-x system  sdcard_rw     314676 2012-03-09 23:29 Bea-Uic-Arno-Firenze.jpg
```

Android Emulator – Looking Under the Hood



Login into the Android OS shell

STEPS

1. Use the Eclipse **AVD Manager** to start one of your AVDs (say Gingerbread23)
2. At the DOS command prompt level run the Android Debug Bridge (**adb**) application

adb shell

```
C:\windows\system32\cmd.exe - adb shell
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Program Files (x86)\Android\android-sdk\platform-tools>adb shell
# ls -l
ls -l
dr-x----- root    root          2012-03-10 00:01 config
drwxrwx--- system  cache        2012-03-10 10:33 cache
lrwxrwxrwx root    root          2012-03-10 00:01 sdcard -> /mnt/sdcard
drwxr-xr-x root    root          2012-03-10 00:01 acct
drwxrwxr-x root    system       2012-03-10 00:01 mnt
lrwxrwxrwx root    root          2012-03-10 00:01 vendor -> /system/vendor
lrwxrwxrwx root    root          2012-03-10 00:01 d -> /sys/kernel/debug
lrwxrwxrwx root    root          2012-03-10 00:01 etc -> /system/etc
-rw-r--r-- root    root          3764 1969-12-31 19:00 ueventd.rc
-rw-r--r-- root    root           0 1969-12-31 19:00 ueventd.goldfish.rc
drwxr-xr-x root    root          2011-02-03 18:01 system
drwxr-xr-x root    root          1969-12-31 19:00 sys
drwxr-xr-x root    root          1969-12-31 19:00/sbin
dr-xr-xr-x root    root          1969-12-31 19:00/proc
-rwxr-x--- root    root         13805 1969-12-31 19:00 init.rc
-rwxr-x--- root    root          1677 1969-12-31 19:00 init.goldfish.rc
-rwxr-x--- root    root         94168 1969-12-31 19:00 init
-rw-r--r-- root    root           118 1969-12-31 19:00 default.prop
drwxrwx-x system  system       2012-03-09 23:02 data
drwx----- root    root          2010-01-27 19:59 root
drwxr-xr-x root    root          2012-03-10 00:02 dev
# df
df
Filesystem      Size  Used  Free  Blksize
/dev            125M  32K  125M  4096
/mnt/asec       125M   0K  125M  4096
/mnt/obb        125M   0K  125M  4096
/system         96M   96M   0K   4096
/data           64M   32M   31M  4096
/cache          64M    1M   62M  4096
/mnt/sdcard     1019M 164M  855M  2048
/mnt/secure/asec 1019M 164M  855M  2048
# cd sdcard
cd sdcard
# ls -l
ls -l
d---rwxr-x system  sdcard_rw    2012-03-09 23:03 LOST.DIR
d---rwxr-x system  sdcard_rw    2012-03-10 19:59 DCIM
---rwxr-x system  sdcard_rw    2012-03-09 23:10 Amarcord.mp3
d---rwxr-x system  sdcard_rw    2012-03-09 23:11 Android
---rwxr-x system  sdcard_rw    2012-03-09 23:29 Bea-Strada-Volterra-12X17.jpg
---rwxr-x system  sdcard_rw    2012-03-09 23:29 Bea-Uic-Arno-Firenze.jpg
```

adb is a tool located in the directory:

C:\Your-SDK-Folder\Android\android-sdk\platform-tools

Android Emulator – Looking Under the Hood

Login into the Android OS shell

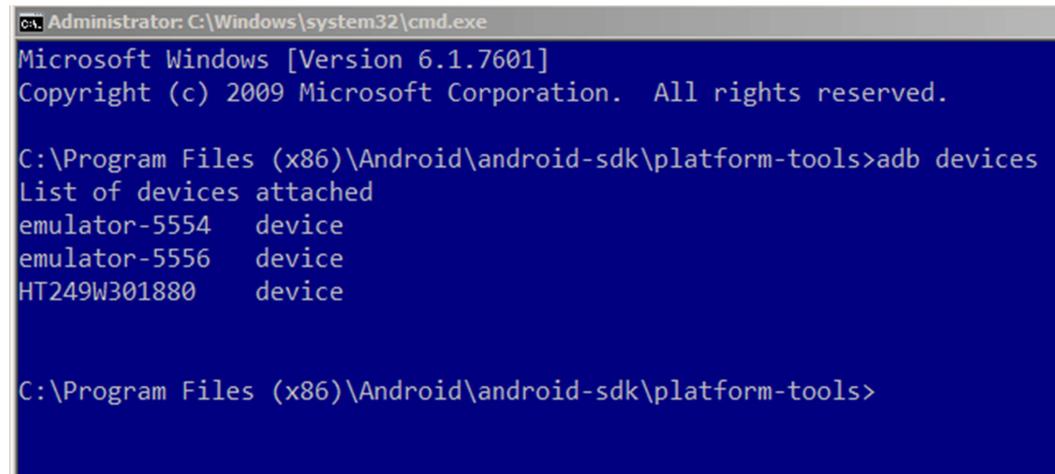
If more than one emulator is running (or your phone is physically connected to the computer using the USB cable) you need to identify the target.

Follow the next steps:

1. Get a list of attached devices

adb devices

List of devices attached
emulator-5554 device
emulator-5556 device
HT845GZ45737 device



```
Administrator: C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Program Files (x86)\Android\android-sdk\platform-tools>adb devices
List of devices attached
emulator-5554    device
emulator-5556    device
HT249W301880    device

C:\Program Files (x86)\Android\android-sdk\platform-tools>
```

2. Run the **adb** application as follows:

adb -s emulator-5554 shell

Remember, the **adb** tool is located at **C:\Program Files (x86)\Android\android-sdk\platform-tools**

Android Emulator – Looking Under the Hood

Login into the Android OS shell

Android accepts a number of Linux shell commands including the useful set below

```
ls ..... show directory (alphabetical order)
mkdir ..... make a directory
rmdir ..... remove directory
rm -r ..... to delete folders with files
rm ..... remove files
mv ..... moving and renaming files
cat ..... displaying short files
cd ..... change current directory
pwd ..... find out what directory you are in
df ..... shows available disk space
chmod ..... changes permissions on a file
date ..... display date
exit ..... terminate session
```

There is no copy (**cp**) command in Android, but you could use **cat** instead.

For instance:

```
# cat data/app/theInstalledApp.apk > cache/theInstalledApp.apk
```

Android Emulator – Looking Under the Hood

Hacking: Moving an app from a Rooted Phone to the Emulator

If you want to transfer an app that is currently installed in your rooted developer's phone to the emulator, follow the next steps:

1. Run command shell: > **adb devices** (find out your hardware's id, say **HT096P800176**)
2. Pull the file from the device to your computer's file system. Enter the command **adb -s HT096P800176 pull data/app/theInstalledApp.apk c:\theInstalledApp.apk**
3. Disconnect your Android phone
4. Run an instance of the Emulator
5. Now install the app on the emulator using the command **adb -s emulator-5554 install c:\theInstalledApp.apk**
adb -s emulator-5554 uninstall data/app/theInstalledApp.apk ← *to uninstall*

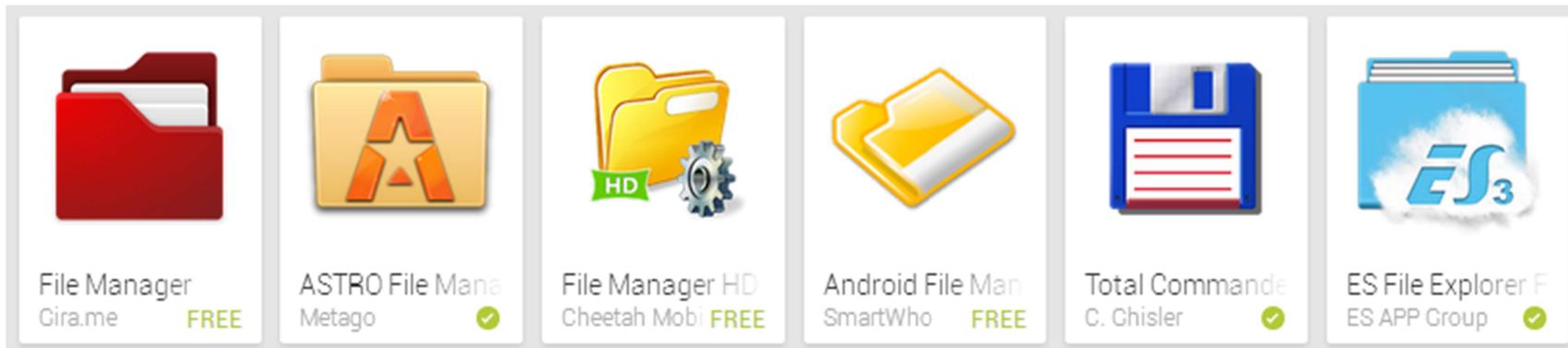
You should see a message indicating the size of the installed package, and finally: *Success.*

Android Emulator – Looking Under the Hood

Simpler than Hacking: Install a File Manager for Android

Visit **Google Play Store** and choose a user-friendly file manager app from the various (usually very good) options available.

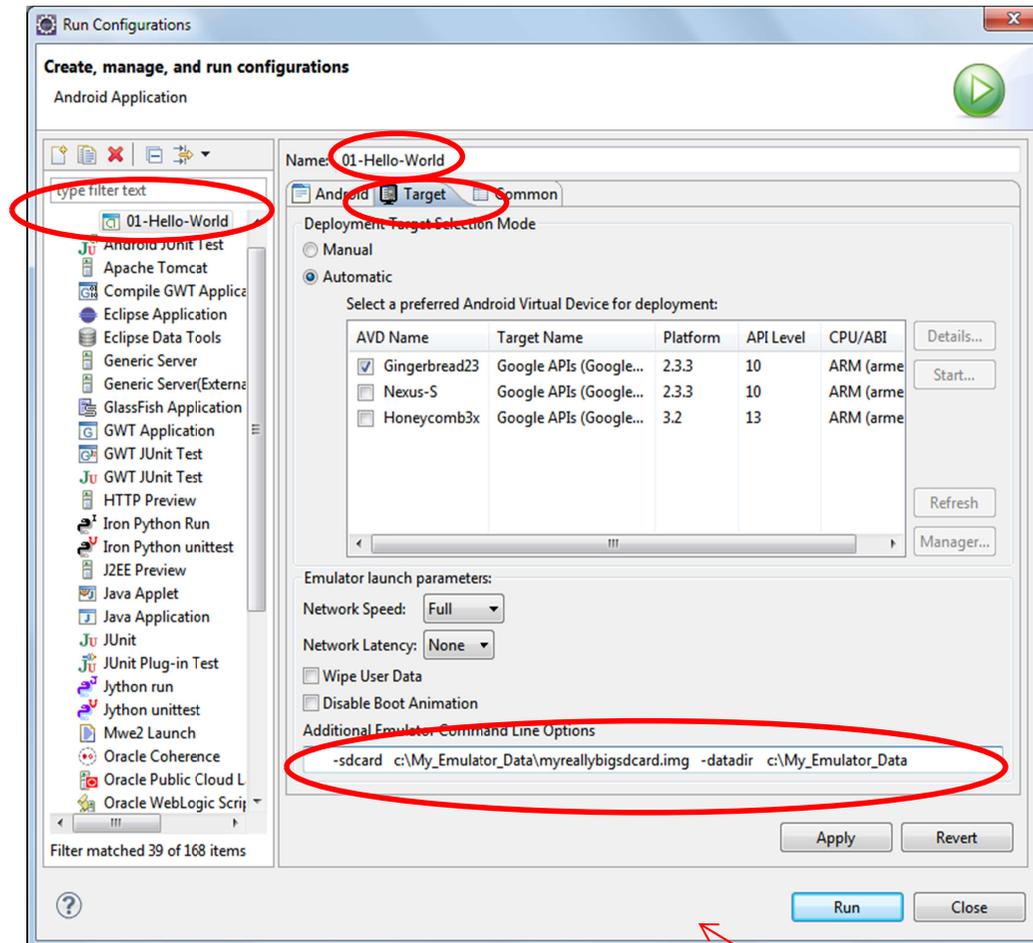
A file manager app allows you to easily administer the folders and files in the system's flash memory and SD card of your Android device (or emulator).



A sample of *File-Management* apps seen at <https://play.google.com> on Aug-27th -2014

Android Emulator – Looking Under the Hood

Using an alternate SD card & userData Image



From the Eclipse menu create a new launch configuration:

Run >

Run Configurations >

New icon



On the **Target** panel:

1. Select existing AVD (Gingerbread in this example)
2. Enter additional Command Line Options (see caption below)
3. Click on **Apply** > **Run**

Additional Emulator Command Line Options:

-sdcard c:\My_Emulator_Data\myreallybigsdcard.img **-datadir** c:\My_Emulator_Data

Android Emulator – Simulate Texting

Sending Text Messages from your Window's PC to the Emulator

1. Start the emulator.
2. Open a new DOS command shell and type :
c:> adb devices
this way you get to know the emulator's numeric port id (usually **5554**, **5556**, and so on)
3. Initiate a Telnet session with the sender at localhost, port 5556 identifies an active (receiving) Android emulator. Type the command:
c:> telnet localhost 5554
4. After receiving the telnet prompt, you can send a text message to the emulator on port 5554 (no quotes needed for the message)
sms send <Sender's phone number> <text message>

Windows7 – temporarily install Telnet Client by using a command line

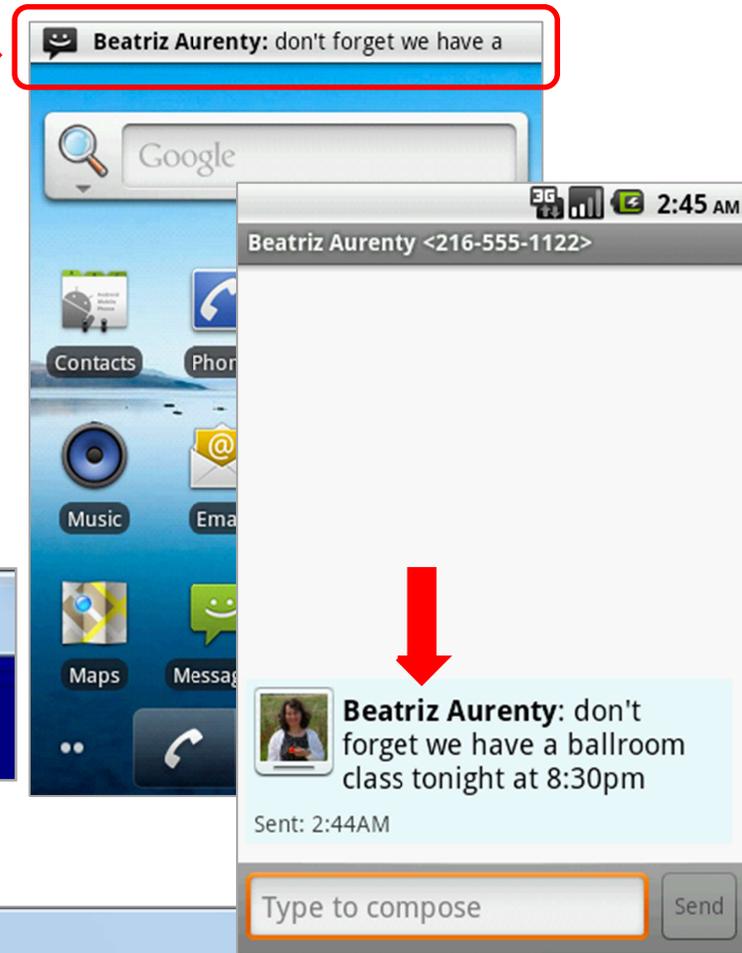
1. Click **Start** button, type **cmd** in the 'search programs and files' box, and then press **ENTER**.
2. Type the following command: **pkgmgr /iu:"TelnetClient"**

Android Emulator – Simulate Texting

Sending a text Message (SMS) from your PC to the Emulator

```
C:\windows\system32\cmd.exe  
C:\Users\1002125>telnet localhost 5554
```

```
C:\> Telnet localhost  
Android Console: type 'help' for a list of commands  
OK  
sms send 5551122 don't forget we have a ballroom class tonight at 8:30pm  
OK  
_
```



Android Emulator – Simulate Phone Calls

Making a Phone Call from your PC to the Emulator

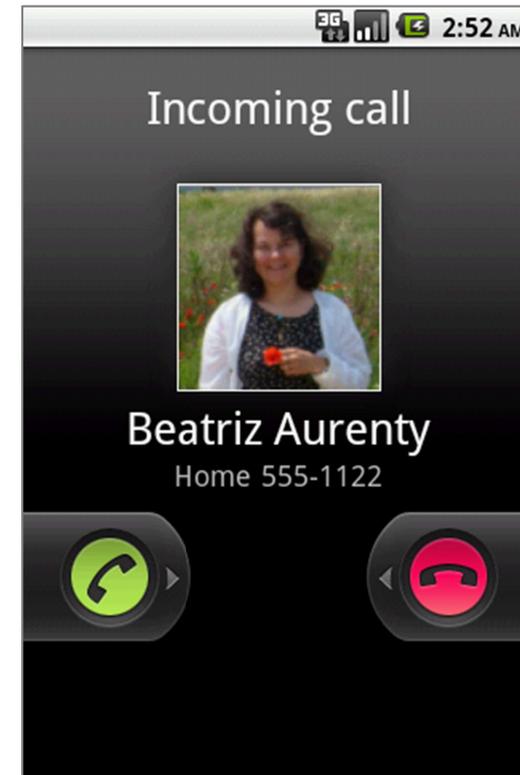
1. Start the emulator.
2. Open a new shell and type :
adb devices
to know the emulator's numeric port id (usually **5554**, **5556**, and so on)
3. Connect to the console using telnet command like:
telnet localhost 5554 (5554 is the 'phone number' to be called)
4. After receiving the telnet prompt you can place a call (voice) with the command
gsm call <caller's phone number>

Android Emulator – Simulate Phone Calls

**Example:
Making a Phone from your PC to the
Emulator**

```
C:\windows\system32\cmd.exe  
C:\Users\1002125>telnet localhost 5554
```

```
Telnet localhost  
Android Console: type 'help' for a list of commands  
OK  
gsm call 5551122  
OK
```



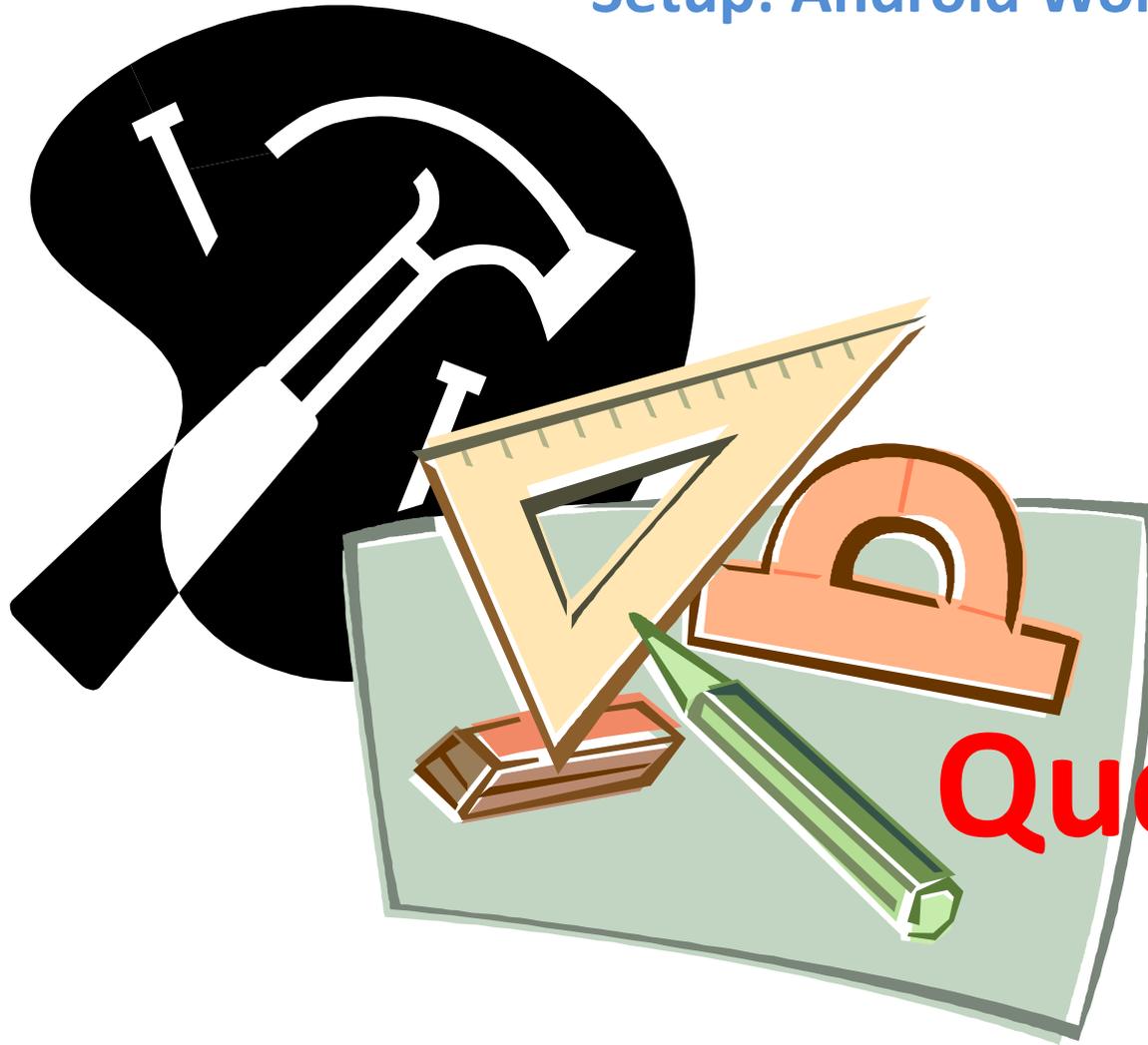


Using: Android Device Monitor

It is *much simpler* to test telephony operations (SMS/Voice) as well as GPS services using the controls included in the IDE (both AS and Eclipse)

- 1. Telephony Status** - change the state of the phone's Voice and Data plans (home, roaming, searching, etc.), and simulate different kinds of network Speed and Latency (GPRS, EDGE, UTMS, etc.).
- 2. Telephony Actions** - perform simulated phone calls and SMS messages to the emulator.
- 3. Location Controls** - send mock location data to the emulator so that you can perform location-aware operations requiring GPS assistance.
 - Manually send individual longitude/latitude coordinates to the device. Click **Manual**, select the coordinate format, fill in the fields and click **Send**.
 - Use a **GPX file** describing a route for playback to the device.
 - Use a **KML** file to place multiple *placemark points* on a map

Lesson 2: Setup: Android Workbench & Emulator

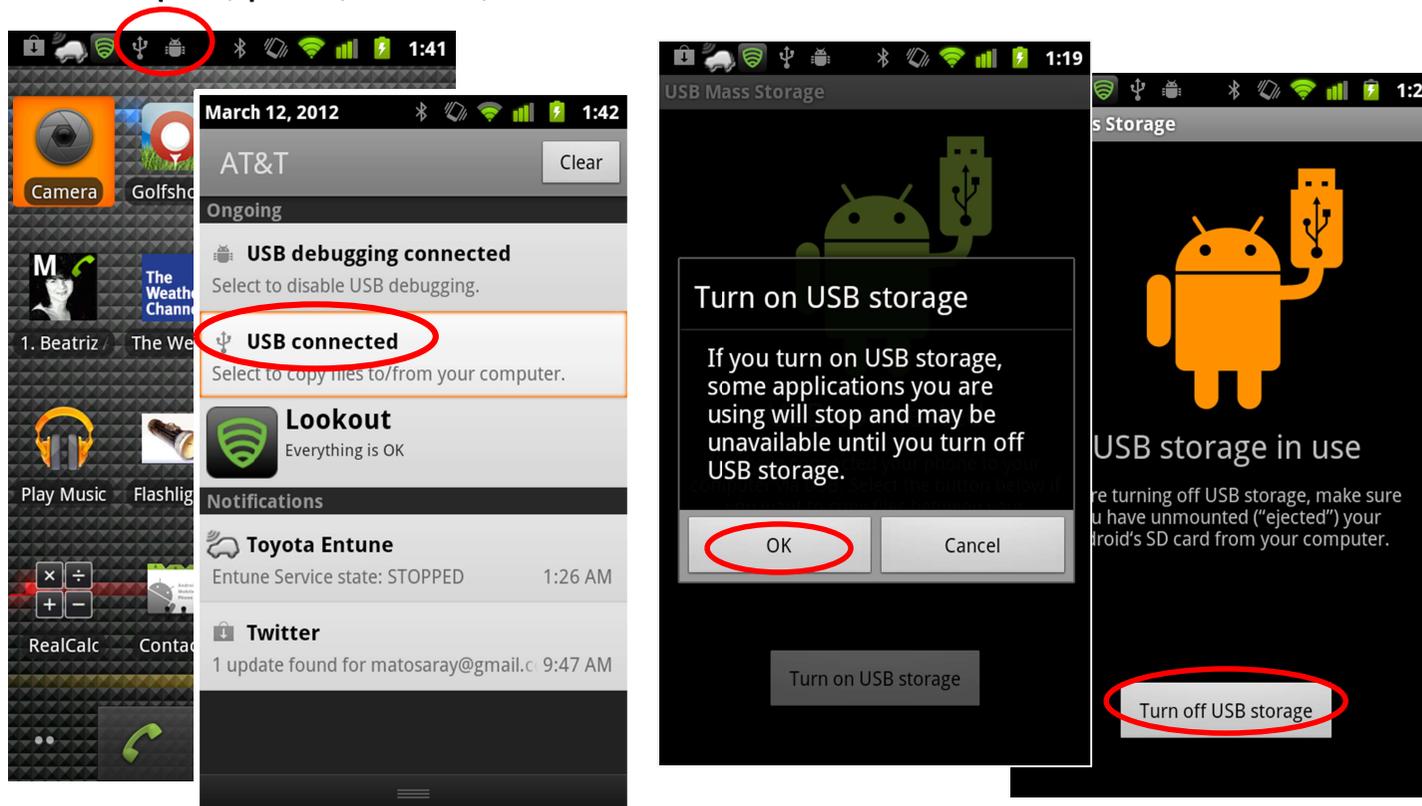


Questions ?

Appendix 1 - Using a Hardware Device

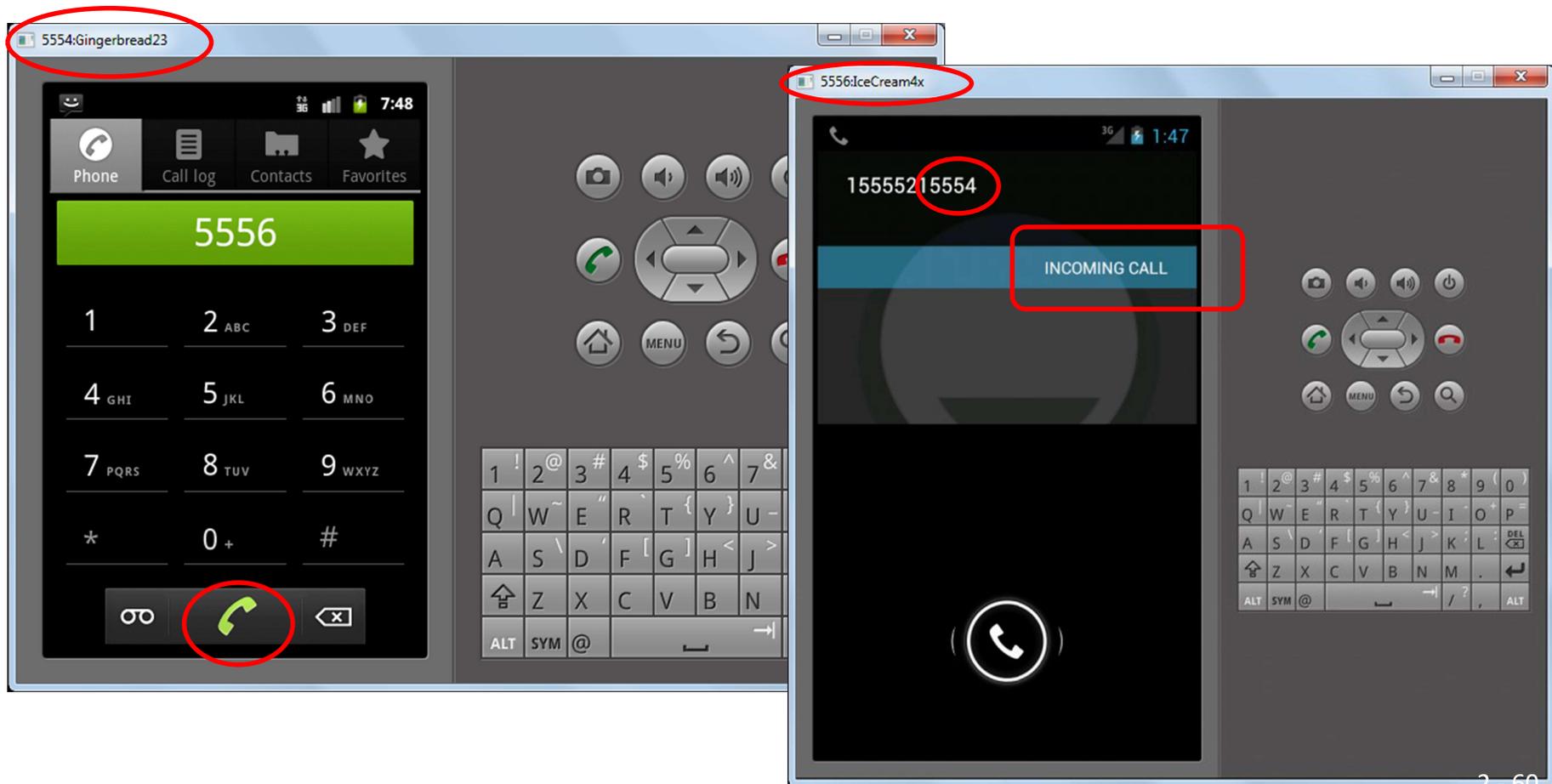
Connecting your Physical Device to the Computer

1. Make sure the USB driver has been installed in your PC (click  SDK Manager > Extras > check box [*Google USB driver package*] to install)
2. Use a mini-USB cable to connect the device to your computer.
3. Expand the Notification bar. Click on [*USB connected*] option.
4. Click on [*Turn on USB storage*] to mount the device.
5. Now you could now use the Eclipse-ADT-File Explorer and your Window's Explorer tool to pull/push/delete/rename files to the device.



Appendix 2 – Emulator-to-Emulator Interaction

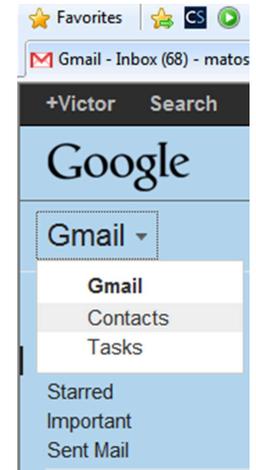
1. Run **two** instances of the emulator (typical IDs are: 5554, 5556, ...)
2. Dial (or send SMS) from one of them (say 5554) to the other (5556)
3. Press the Green/Red call buttons to accept/terminate the call
4. Try sending SMS (use numbers 5554 and 5556)



Appendix 3 – Sync your Contacts

How to Transfer Your Google Contacts into the Emulator

1. Go to your **Gmail account** using a web browser, click on **Gmail** > **Contacts** on the left sidebar.
2. Select all the contacts you want on your emulator/phone. Then click on **More** > **Export** and select **vCard** format. Download the “**contacts.vcf**” file to your PC.
3. Push the **contacts.vcf** file from the PC to the emulator’s **SD card**.
4. Open the emulator’s **Contacts** app hit **Menu** > **Import**.
5. Choose the option *Import from SD card*.



Source visited on July 2009, link:

<http://stackoverflow.com/questions/1114052/importing-gmail-contacts-on-android-emulator>

Appendix 4

Shortcuts: Android-Studio IDE

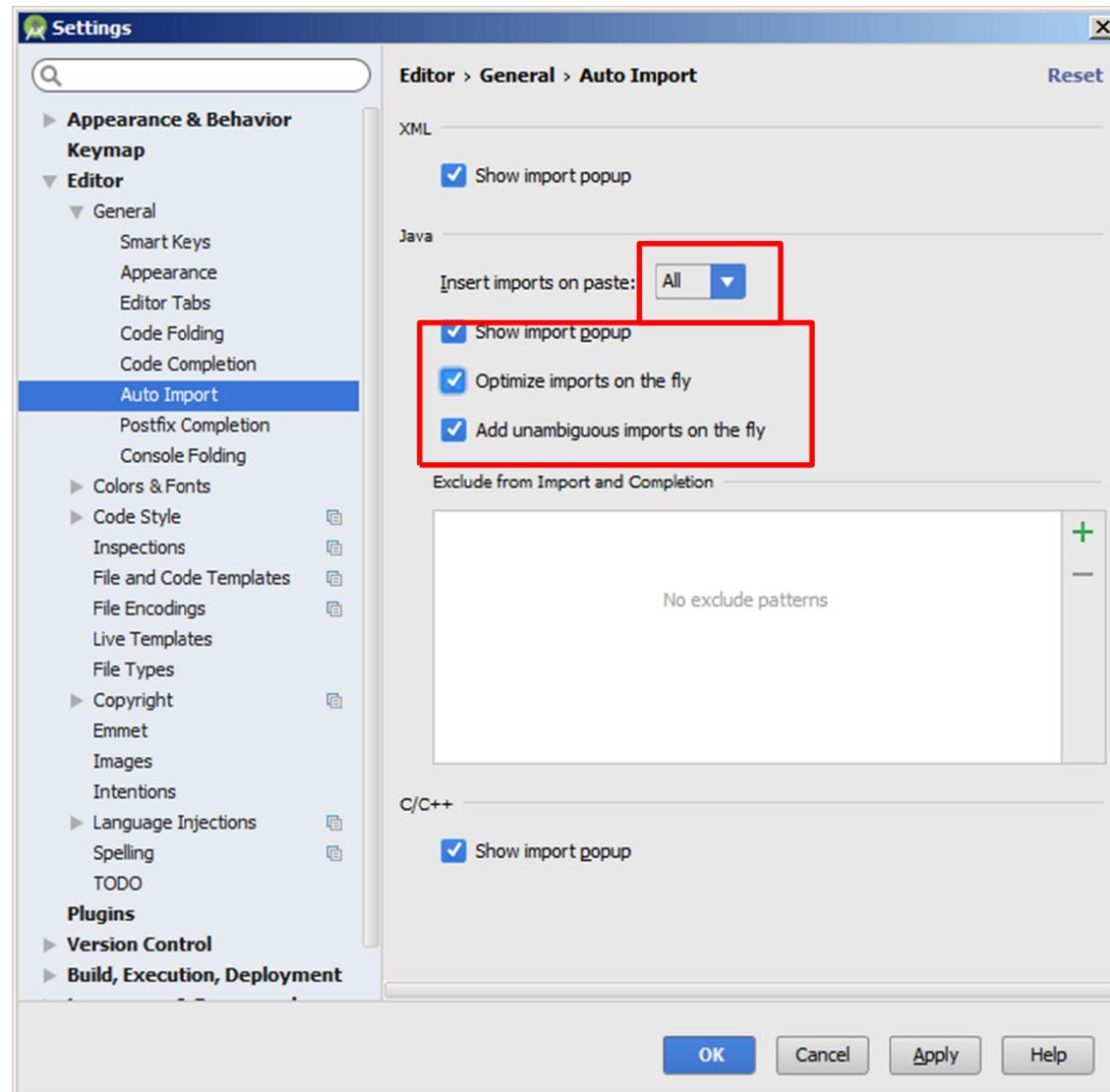
Eclipse developers are used to typing

Ctrl + Shift + O

To **Organize ALL imports**.

To automatically accomplish the same effect, modify your Android Studio Workbench as indicated on the figure to the right.

File > Settings > Editor > General > Auto Import



Appendix 4

Shortcuts: Android-Studio IDE

Operation	Android Studio Shortcut
Reformat code	CTRL + ALT + L
Optimize imports	CTRL + ALT + O
Code Completion	CTRL + SPACE
Issue quick fix	ALT + ENTER
Surround code block	CTRL + ALT + T
Line Comment or Uncomment	CTRL + /
Block Comment or Uncomment	CTRL + SHIFT + /
Close Active Tab	CTRL + F4
Build and run	SHIFT + F10
Build	CTRL + F9
All Options	Ctrl + Shift + A