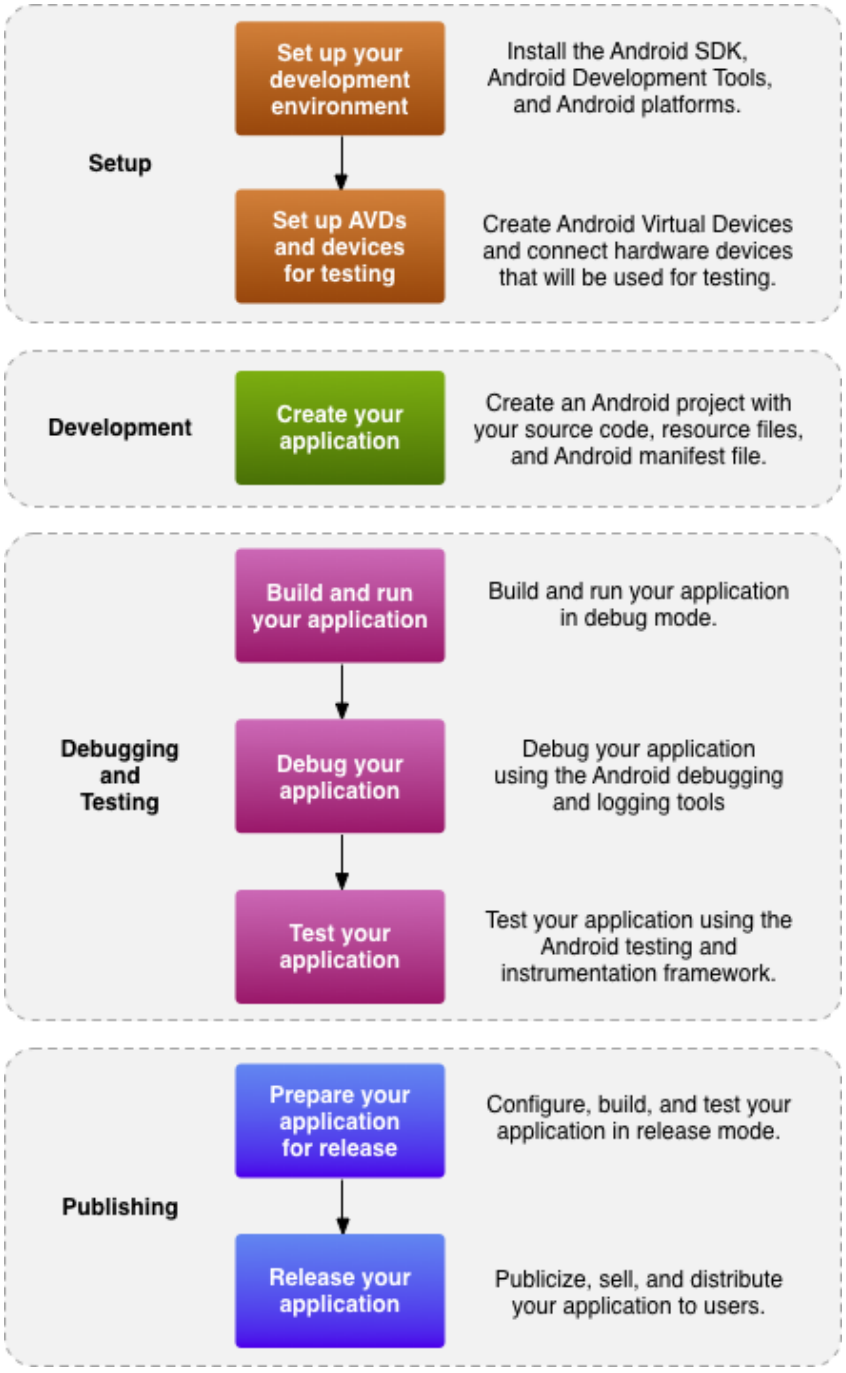
Android: Application Development

FIRST LOOK TO APPLICATION DEVELOPMENT

Work Flow


Setup
Set up your development environment
Install the Android SDK, Android Development Tools, and Android platforms.
Set up AVDs and devices for testing
Create Android Virtual Devices and connect hardware devices that will be used for testing.
Development
Create your application
Create an Android project with your source code, resource files, and Android manifest file.
Debugging and Testing
Build and run your application
Build and run your application in debug mode.
Debug your application
Debug your application using the Android debugging and logging tools
Test your application
Test your application using the Android testing and instrumentation framework.
Publishing
Prepare your application for release
Configure, build, and test your application in release mode.
Release your application
Publicize, sell, and distribute your application to users.

Application Development Requires

Application Development for Android requires:

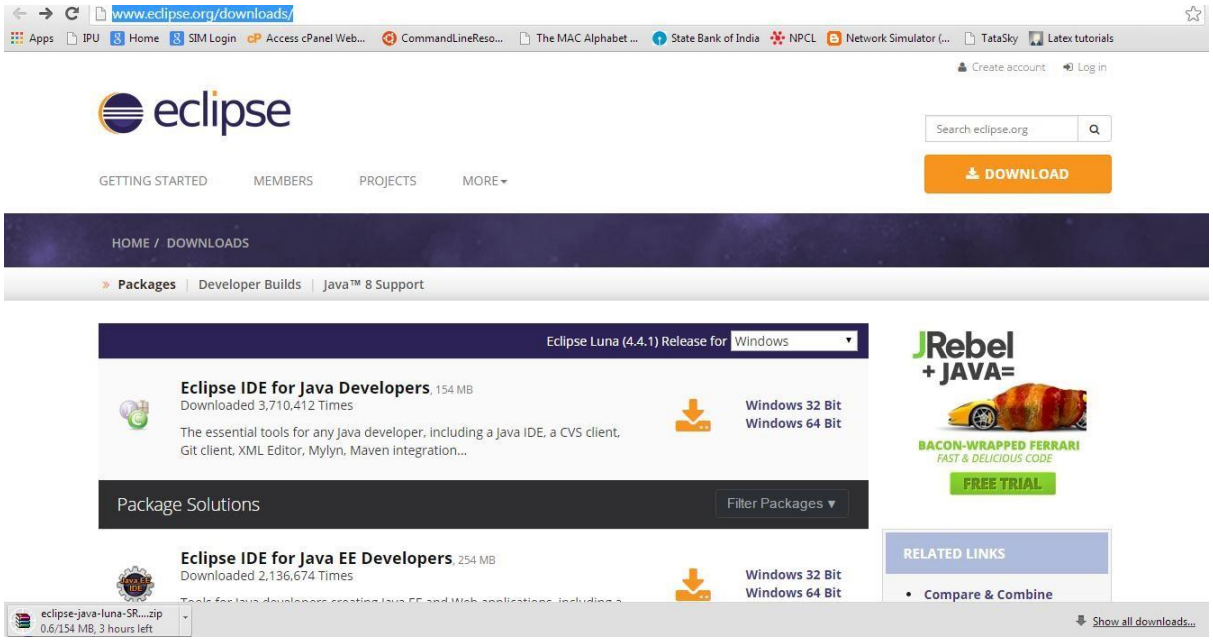
- SDK
- An Editor (Eclipse/Netbeans)
  - APT

  Or

- Android Studio (with basic sdk)
  - Android Studio is now the official IDE for Android

App Development: Using Eclipse

- **Download Eclipse ide from**
  - http://www.eclipse.org/downloads/
  - Jdk must be installed



Android Development Tools (ADT)

Android Development Tools (ADT) is a plugin for the Eclipse IDE that extends the capabilities of Eclipse to let you

- Quickly set up new Android projects,
- Create an application UI,
- Add packages based on the Android Framework API,
- Debug your applications using the Android SDK tools
- Export signed (or unsigned) .apk files in order to distribute your application.

Download the ADT Plugin

To add the ADT plugin to Eclipse:

1. Start Eclipse, then select Help > Install New Software.
2. Click Add, in the top-right corner.
3. In the Add Repository dialog that appears, enter "ADT Plugin" for the *Name* and the following URL for the *Location*:

https://dl-ssl.google.com/android/eclipse/

Note: The Android Developer Tools update site requires a secure connection. Make sure the update site URL you enter starts with HTTPS.

4. Click OK.
5. In the Available Software dialog, select the checkbox next to Developer Tools and clickNext.
6. In the next window, you'll see a list of the tools to be downloaded. ClickNext.
7. Read and accept the license agreements, then click Finish.
8. If you get a security warning saying that the authenticity or validity of the software can't be established, click OK.
9. When the installation completes, restart Eclipse.

Installing from an offline version

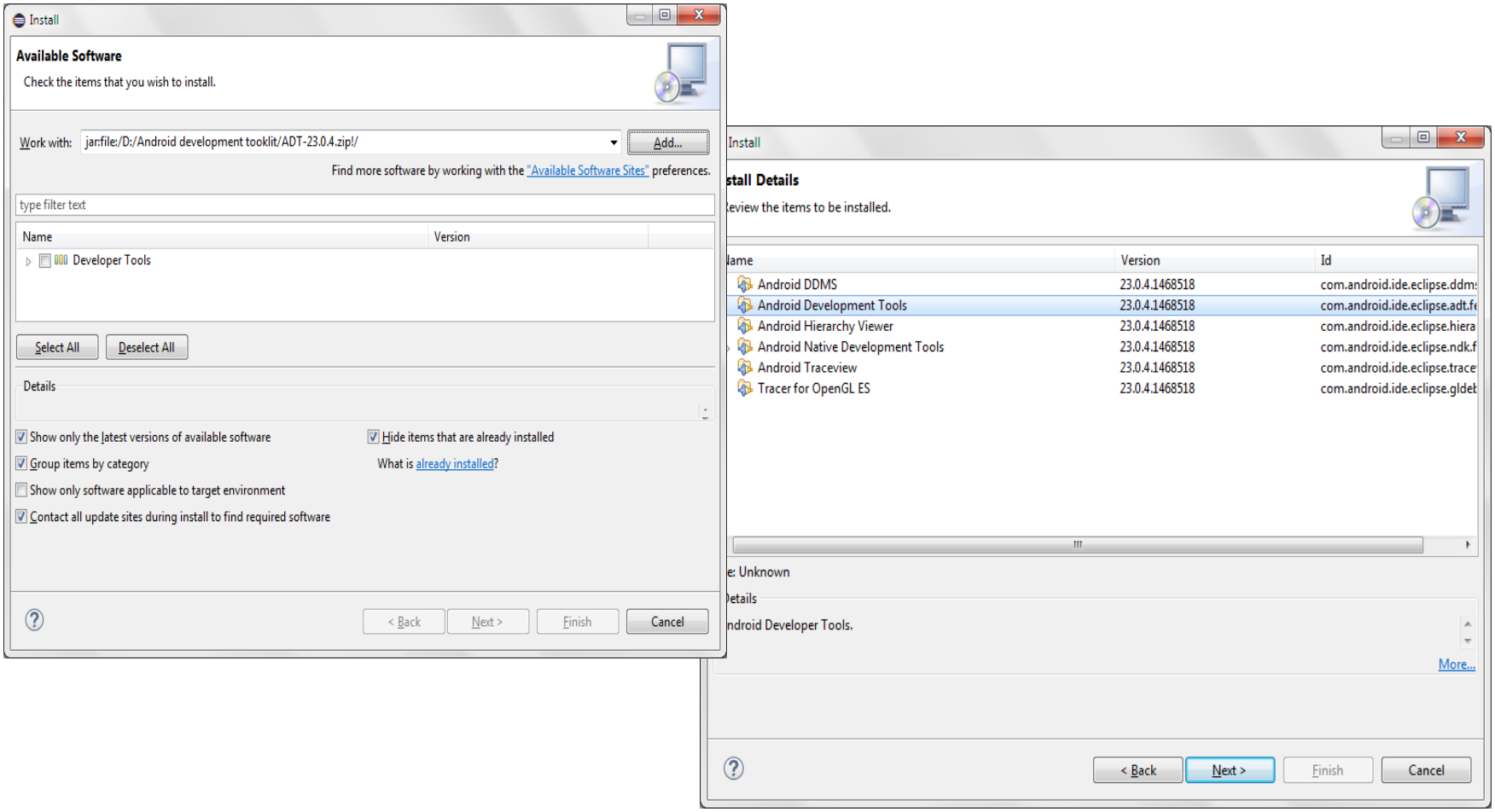
- If you are still unable to use Eclipse to download the ADT plugin as a remote update site, you can download the ADT zip file to your local machine and manually install it:

1. Download the ADT Plugin zip file (do not unpack it):

https://dl.google.com/android/ADT-23.0.4.zip

1. Start Eclipse, then select Help > Install New Software.
2. Click Add, in the top-right corner.
3. In the Add Repository dialog, click Archive.
4. Select the downloaded ADT-23.0.4.zip file and click OK.
5. Enter "ADT Plugin" for the name and click OK.
6. In the Available Software dialog, select the checkbox next to Developer Tools and click Next.
7. In the next window, you'll see a list of the tools to be downloaded. Click Next.
8. Read and accept the license agreements, then click Finish.
9. If you get a security warning saying that the authenticity or validity of the software can't be established, click OK.
10. When the installation completes, restart Eclipse.
11. To update your plugin once you've installed using the zip file, you will have to follow these steps again instead of the default update instructions.

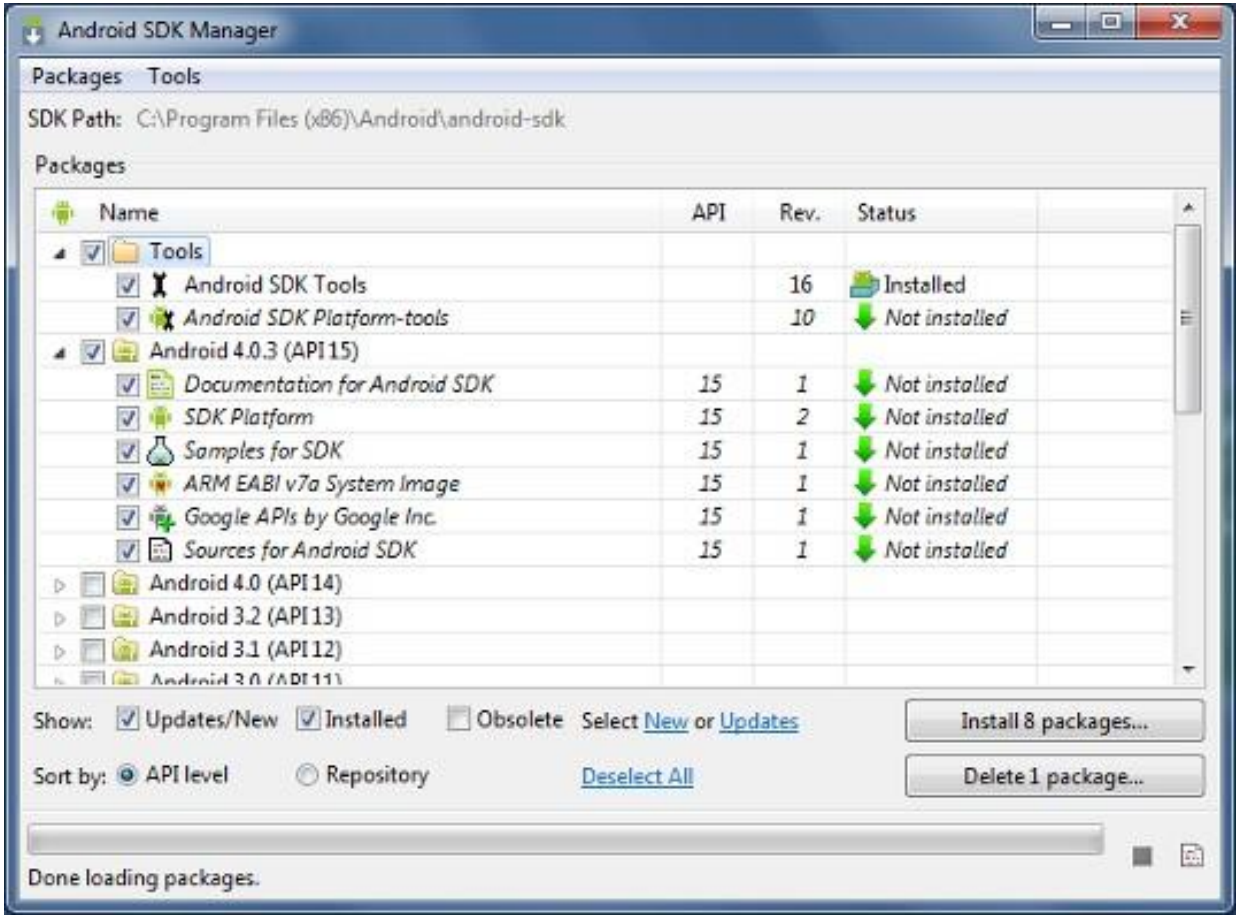
Installing ADT...



Android SDK Manager

- The Android SDK separates tools, platforms, and other components into packages you can download using the SDK Manager.
  - For example, when the SDK Tools are updated or a new version of the Android platform is released, you can use the SDK Manager to quickly download them to your environment.
- You can launch the SDK Manager in one of the following ways:
  - From Eclipse (with ADT), select Window > Android SDK Manager.
  - From Android Studio, select Tools > Android > SDK Manager.
  - On Windows, double-click the SDK Manager.exe file at the root of the Android SDK directory.

Updating SDK



The Android SDK Manager shows the SDK packages that are available, already installed, or for which an update is available.

Managing Virtual Devices AVD

- An Android Virtual Device (AVD) is an emulator configuration that lets you model an actual device by defining hardware and software options to be emulated by the Android Emulator.
- The easiest way to create an AVD is to use the graphical AVD Manager, which you launch from Eclipse by clicking**Window > AVD Manager**.
- You can also start the AVD Manager from the command line by calling the android tool with theavd options, from the **<sdk>/tools/** directory.

Android Virtual Device (AVD)

An AVD consists of:

- **A hardware profile:** Defines the hardware features of the virtual device.
  - For example, you can define whether the device has a camera, whether it uses a physical QWERTY keyboard or a dialing pad, how much memory it has, and so on.
- **A mapping to a system image:** You can define what version of the Android platform will run on the virtual device.
  - You can choose a version of the standard Android platform or the system image packaged with an SDK add-on.
- **Other options**: You can specify the emulator skin you want to use with the AVD, which lets you control the screen dimensions, appearance, and so on.
  - You can also specify the emulated SD card to use with the AVD.
  - A dedicated storage area on your development machine: the device's user data (installed applications, settings, and so on) and emulated SD card are stored in this area.
- You can create as many AVDs as you need, based on the types of device you want to model.

Note: To thoroughly test your application, you should create an AVD for each general device configuration (for example, different screen sizes and platform versions) with which your application is compatible and test your application on each one.

Android Virtual Device (AVD)...

Keep these points in mind when you are selecting a system image target for your AVD:

- **The API Level of the target is important**, because your application will not be able to run on a system image whose API Level is less than that required by your application, as specified in the **minSdkVersion** attribute of the application's manifest file.
- You should create at least one AVD that uses a target whose API Level is greater than that required by your application, because it allows you to test the **forward-compatibility** of your application. Forward-compatibility testing ensures that, when users who have downloaded your application receive a system update, your application will continue to function normally.
- If your application declares a **uses-library** element in its manifest file, the application can only run on a system image in which that external library is present. If you want to run your application on an emulator, create an AVD that includes the required library.
  - Usually, you must create such an AVD using an Add-on component for the AVD's platform (for example, the Google APIs Add-on contains the Google Maps library).

Android Studio: Set Up Your Environment

- You need to:
  - Download Android Studio.
    - from http://developer.android.com/sdk/index.html
  - Download the latest SDK tools and platforms using the SDK Manager.
    - From http://developer.android.com/tools/help/sdk-manager.html

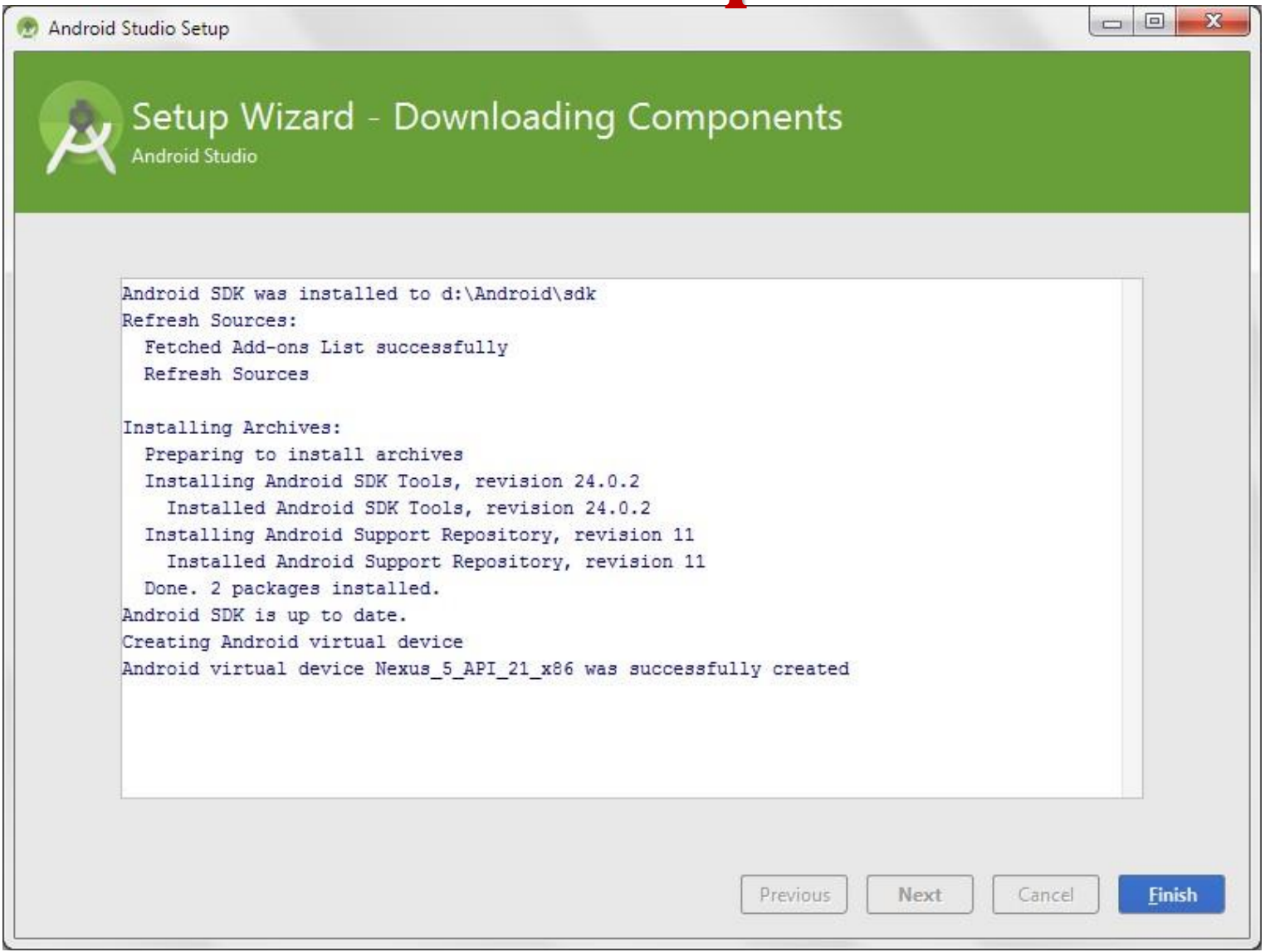
If you have been using Eclipse with ADT, be aware that Android Studio is now the official IDE for Android, so you should migrate to Android Studio to receive all the latest IDE updates.

System Requirements

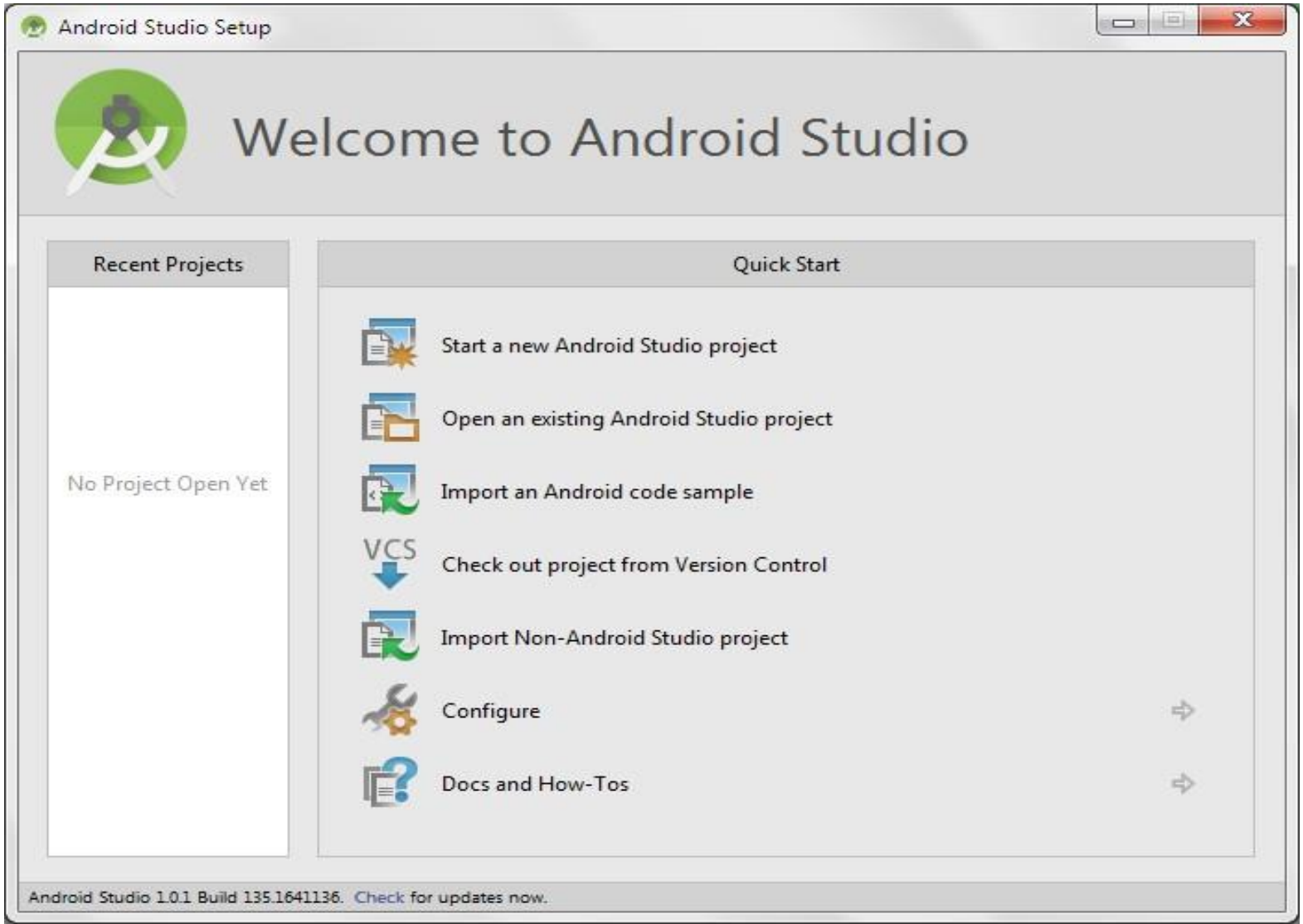
Windows

- Microsoft® Windows® 8/7/Vista/2003 (32 or 64-bit)
- 2 GB RAM minimum, **4 GB RAM** recommended
- 400 MB hard disk space + **at least 1 G** for Android SDK, emulator system images, and caches
- 1280 x 800 minimum screen resolution
- Java Development Kit **(JDK) 7**
- Optional for accelerated emulator: Intel® processor with support for Intel® VT-x, Intel® EM64T (Intel® 64), and Execute Disable (XD) Bit functionality

Downloading Additional SDK Components



Startup Screen (ver:1.0.1)

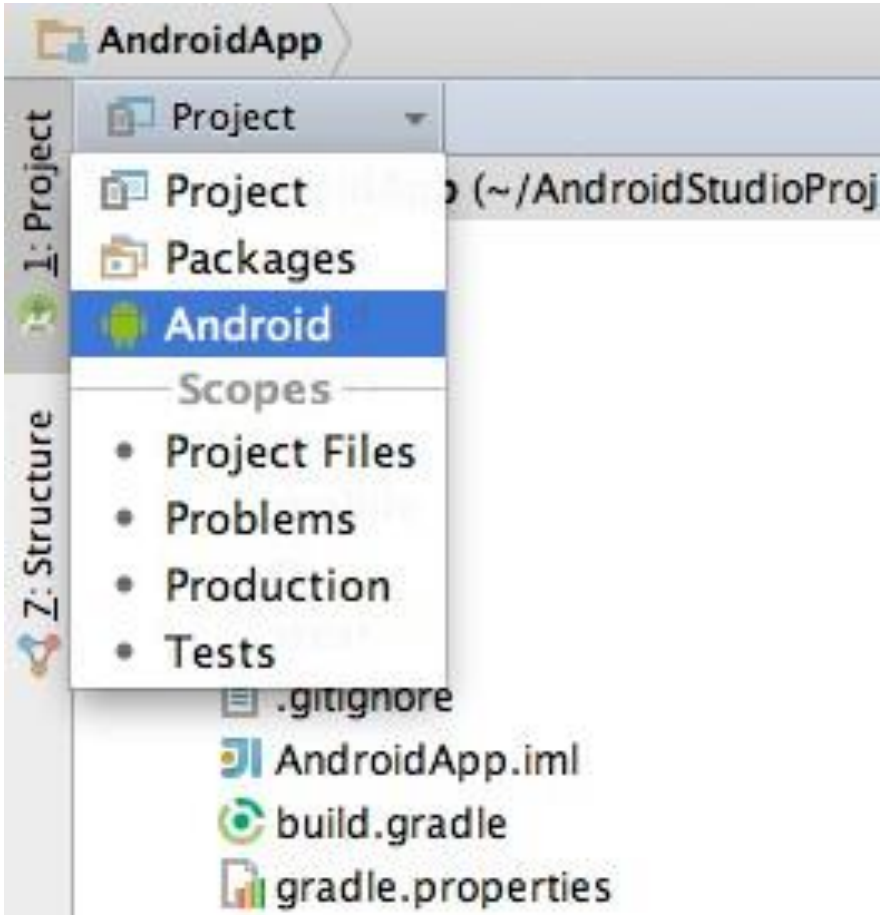


Android Studio: Why?

- Built on IntelliJ IDEA Community Edition, the popular Java IDE by JetBrains.
- Flexible Gradle-based build system.
- Build variants and multiple APK generation.
- Expanded template support for Google Services and various device types.
- Rich layout editor with support for theme editing.
- Lint tools to catch performance, usability, version compatibility, and other problems.
- ProGuard and app-signing capabilities.
- Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine.
- For futher details (Android Studio Overview)

https://developer.android.com/tools/studio/index.html

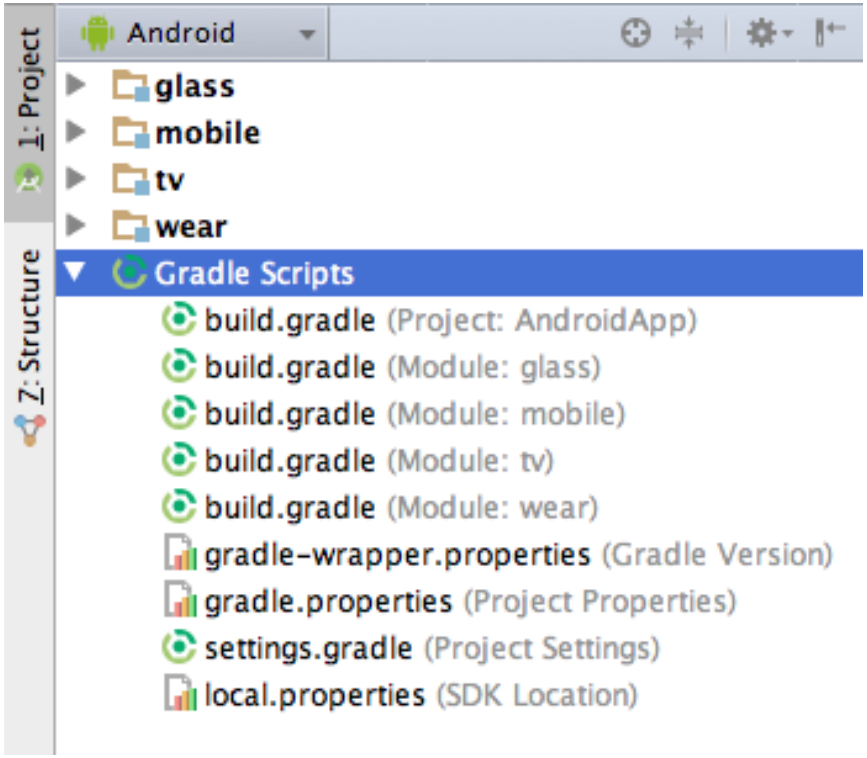
Project and File Structure



***Android* Project View**

- By default, Android Studio displays your profile files in the *Android* project view. This view shows a flattened version of your project's structure that provides quick access to the key source files of Android projects and helps you work with the new Gradle-based build system. The Android project view:
- Groups the build files for all modules at the top level of the project hierarchy.
- Shows the most important source directories at the top level of the module hierarchy.
- Groups all the manifest files for each module.
- Shows resource files from all Gradle source sets.
- Groups resource files for different locales, orientations, and screen types in a single group per resource type.

Project and File Structure



- The *Android* project view shows all the build files at the top level of the project hierarchy under **Gradle Scripts**. Each project module appears as a folder at the top level of the project hierarchy and contains these three elements at the top level:
- **java/** - Source files for the module.
- **manifests/** - Manifest files for the module.
- **res/** - Resource files for the module.

Project Components

- **src – your source code**
- **gen – auto-generated code (usually just R.java)**
- **Included libraries**
- **Resources**
  - Drawables (like .png images)
  - Layouts
  - Values (like strings)
- **Manifest file**

R Class

- **Auto-generated: you shouldn't edit it**
- **Contains IDs of the project resources**
- **Enforces good software engineering**
- **Use findViewById and Resources object to get access to the resources**
  - Ex. Button b = (Button)findViewById(R.id.button1)
  - Ex. getResources().getString(R.string.*hello*));

Layouts (1)

- **Eclipse has a great UI creator/Android Studio has much better**
  - Generates the XML for you
- **Composed of *View* objects**
- **Can be specified for portrait and landscape mode**
  - Use same file name, so can make completely different UIs for the orientations without modifying any code

Strings

- **In res/values**
  - strings.xml
- **Application wide available strings**
- **Promotes good software engineering**
- **UI components made in the UI editor should have text defined in strings.xml**

- **Strings are just one kind of 'Value' there are many others**

Manifest File (1)

- **Contains characteristics about your application**
- **When have more than one Activity in app, NEED to specify it in manifest file**
  - Go to graphical view of the manifest file
  - Add an Activity in the bottom right
  - Browse for the name of the activity
- **Need to specify Services and other components too**
- **Also important to define permissions and external libraries, like Google Maps API**

USB Debugging

- **Should be enabled on phone to use developer features**
- **In the main apps screen select Settings -> Applications -> Development -> USB debugging (it needs to be checked)**

Android Debug Bridge(ADB)

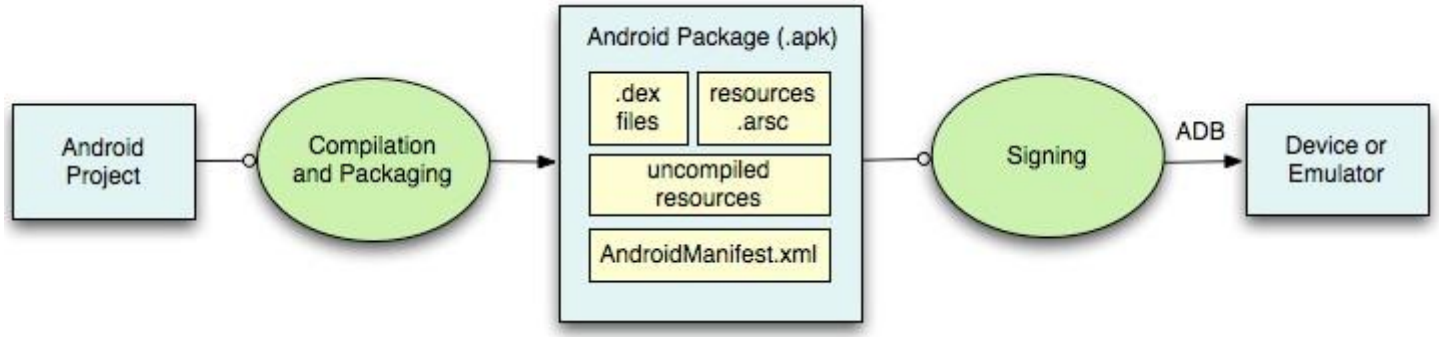
- **Used for a wide variety of developer tasks**
  - Read from the log file
  - Show what android devices are available
  - Install android applications (.apk files)
- **In the 'platform-tools' directory of the main android sdk directory**
  - Recommend putting this directory and the 'tools' directory on the system path
- **adb.exe**

Debugging

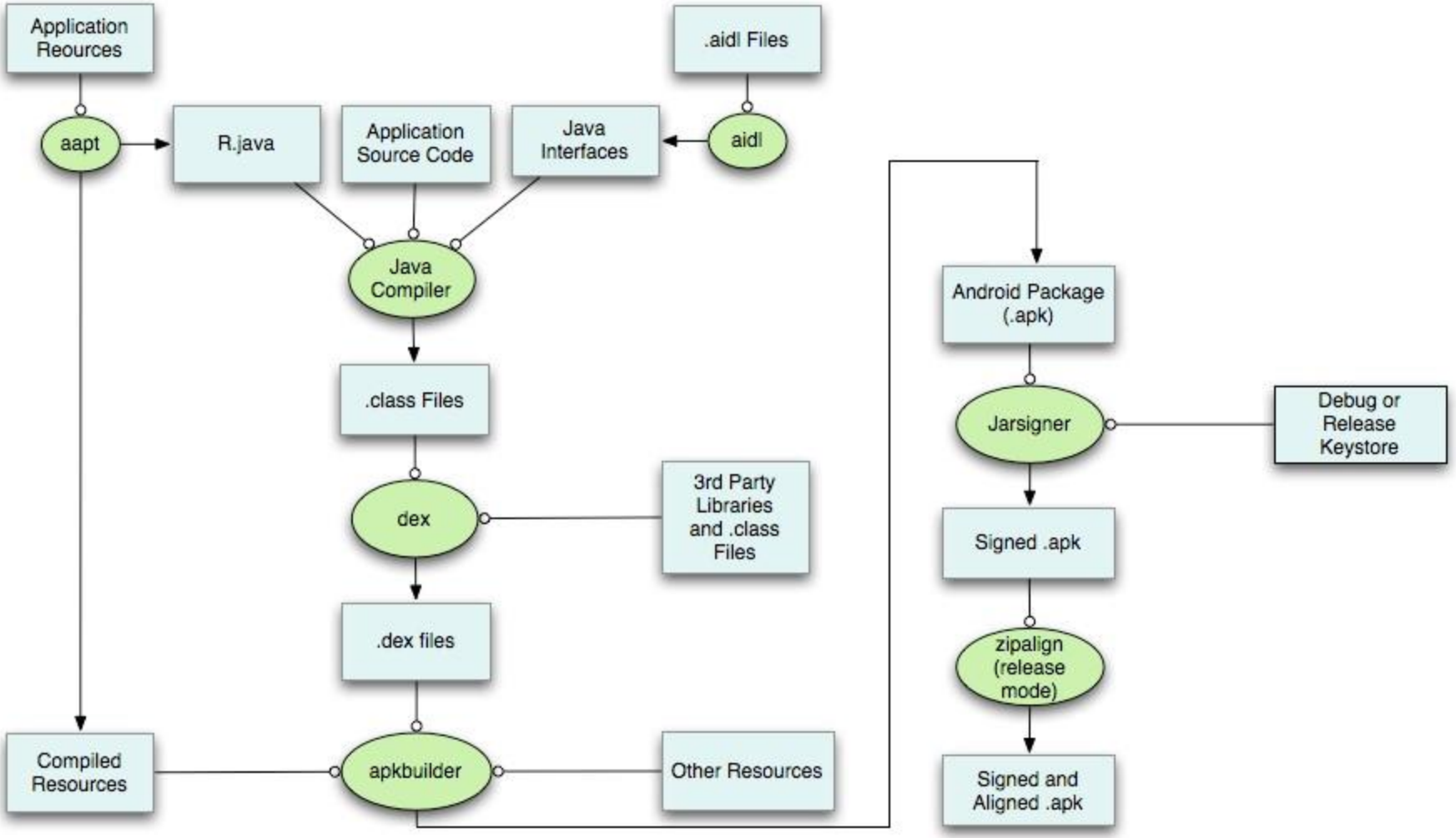
- **Instead of using traditional System.out.println, use the Log class**
  - Imported with android.util.Log
  - Multiple types of output (debug, warning, error, …)
  - Log.d(<tag>,<string>)
- **Can be read using logcat.**
  - Print out the whole log, which auto-updates
    - adb logcat
  - Erase log
    - adb logcat –c
  - Filter output via tags
    - adb logcat <tag>:<msg type> *:S
    - can have multiple <tag>:<msg type> filters
    - <msg type> corresponds to debug, warning, error, etc.
    - If use Log.d(), then <msg type> = D
- **Reference**
  - http://developer.android.com/guide/developing/debugging/debugging-log.html

Building and Running

- During the build process, your Android projects are compiled and packaged into an **.apk** file, the container for your application binary.
- It contains all of the information necessary to run your application on a device or emulator, such as
  - compiled .dex files (.class files converted to Dalvik byte code)
  - a binary version of the AndroidManifest.xml file
  - compiled resources (resources.arsc)
  - uncompiled resource files for your application.



Detailed Build Process



The general process for a typical build is outlined below:

- The **Android Asset Packaging Tool (aapt)** takes your application resource files, such as the AndroidManifest.xml file and the XML files for your Activities, and compiles them. An R.java is also produced so you can reference your resources from your Java code.
- The aidl tool converts any .aidl interfaces that you have into Java interfaces.
- All of your Java code, including the R.java and .aidl files, are compiled by the Java compiler and .class files are output.
- The dex tool converts the .class files to Dalvik byte code. Any 3rd party libraries and .class files that you have included in your project are also converted into .dex files so that they can be packaged into the final .apk file.
- All non-compiled resources (such as images), compiled resources, and the .dex files are sent to the apkbuilder tool to be packaged into an .apk file.
- Once the .apk is built, it must be signed with either a debug or release key before it can be installed to a device.
- Finally, if the application is being signed in release mode, you must align the .apk with the zipalign tool. Aligning the final .apk decreases memory usage when the application is running on a device.

Android SDK Tools

- **android**
  - Lets you manage AVDs, projects, and the installed components of the SDK.
- **Dalvik Debug Monitor Server (ddms)**
  - Lets you debug Android applications.
- **dmtracedump**
  - Generates graphical call-stack diagrams from trace log files. The tool uses the Graphviz Dot utility to create the graphical output, so you need to install Graphviz before running dmtracedump. For more information on using dmtracedump, see Profiling with Traceview and dmtracedump
- **Android Emulator (emulator)**
  - A QEMU-based device-emulation tool that you can use to design, debug, and test your applications in an actual Android run-time environment.
- **layoutopt**
  - Lets you quickly analyze your application's layouts in order to optimize them for efficiency.

Android SDK Tools...

- **mksdcard**
  - Helps you create a disk image that you can use with the emulator, to simulate the presence of an external storage card (such as an SD card).
- **Monkey**
  - Runs on your emulator or device and generates pseudo-random streams of user events such as clicks, touches, or gestures, as well as a number of system-level events. You can use the Monkey to stress-test applications that you are developing, in a random yet repeatable manner.
- **monkeyrunner**
  - Provides an API for writing programs that control an Android device or emulator from outside of Android code.
- **ProGuard**
  - Shrinks, optimizes, and obfuscates your code by removing unused code and renaming classes, fields, and methods with semantically obscure names.

Android SDK Tools...

- **Systrace**
  - Lets you analyze the execution of your application in the context of system processes, to help diagnose display and performance issues.
- **sqlite3**
  - Lets you access the SQLite data files created and used by Android applications.
- **traceview**
  - Provides a graphical viewer for execution logs saved by your application.
- **zipalign**
  - Optimizes .apk files by ensuring that all uncompressed data starts with a particular alignment relative to the start of the file. This should always be used to align .apk files after they have been signed

Platform Tools

- The platform tools are typically updated every time you install a new SDK platform.
- Each update of the platform tools is backward compatible with older platforms. Usually, you directly use only one of the platform tools—the Android Debug Bridge (adb).
- Android Debug Bridge is a versatile tool that lets you manage the state of an emulator instance or Android-powered device.
- You can also use it to install an Android application (.apk) file on a device.
- The other platform tools, such as **aidl, aapt, dexdump, and dx**, are typically called by the Android build tools or Android Development Tools (ADT), so you rarely need to invoke these tools directly. As a general rule, you should rely on the build tools or the ADT plugin to call them as needed.