Android Development Migrating from Eclipse to Android Studio

(LAST MODIFIED: 11/7/15)

About Android Studio (AS)

This document is written to Android Studio version 1.4 as it is installed on Mac OS X (v.11). The Android Studio IDE is provided for free by android.developers.com and is based upon the IntelliJ IDEA for Java IDE by JetBrains.

AS IDE Feature Settings to Note on Mac OS X version

- Android Studio/Preferences.../Appearance & Behavior/Appearance UI Options group box, Theme: list box dropdown, select "Darcula"
- Android Studio/Preferences.../Editor/General Soft Wraps group box, for soft wraps in editor and console
- Android Studio/Preferences.../Editor/General Highlight of Caret Movement group box, Highlight match brace checkbox
- Android Studio/Preferences.../Editor/General Other group box, Strip trailing spaces on Save: list box dropdown, select "All" (get rid of all unnecessary white space)
- Android Studio/Preferences.../Editor/Appearance, check "Show line numbers"
- Android Studio/Preferences.../Editor/Code Style "Default Indent Options" group box, uncheck "Use tab character" (will indent only using space chars)
- Android Studio/Preferences.../Keymap/Keymaps: list box dropdown to "Visual Studio" or "Mac OS X 10.5+"
- Android Studio/Preferences.../Appearance & Behavior/System Settings/Updates, to the right of "Automatically check updates for" set list box dropdown to "Stable Channel"

Modern Android Source File Organization

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

Eclipse Project source code structure layout and IntelliJ Module's are the same. Android uses a new Module source code layout. See the above URL for details

The Unity3D Editor/Player 5.X way of integrating Android plugins.

http://forum.unity3d.com/threads/obsolete-providing-android-resources-in-assets-plugins-android-res-is-deprecated.315889/

Android Studio (AS) Development Platform Requirements (for Mac OS X, v.11)

1. Download the Latest AS IDE here

   https://developer.android.com/sdk/index.html

   The document is written to version 1.4. The best way to approach this complicated setup process, after installing AS, use the wizard to create a hello world Android app and keep the that project open.

2. Download and install the latest two Java SE (Standard Edition) JDK (not JRE, a custom JRE and platform specific JVM are bundled with AS) version from Java.Oracle.com - versions 1.7 and 1.8. For Mac OS X, if there was a previous version of JDK and AS installed, update AS configuration to use latest and greatest installed JDK. Go to file path Applications/"Android Studio.app". Right click to "Show Package Content." Inside "Contents" directory, edit "info.plist" file with a real text editor. In “info.plist", find the string "<key>JVMVersion</key>." Replace old version string with latest and greatest just installed, e.g. change "<string>1.6*,1.7+</string>" to “<string>1.7*,1.8*</string>"

3. To set which installed JDK to use, in AS go to “File”/“Project Structure…", “SDK Location” heading, "JDK location:" textbox browse button, browse to JDK. E.g. “/Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home”.

4. Android creates its own Java intermediate byte code, called DEX, and provides its own VM to execute DEX byte code, called the Dalvik virtual memory. However, starting with Android 4.4 KitKit, the translation of DEX to native code takes place at a different time. Changed from just-in-time (JIT) at runtime to ahead-of-time (AOT), that is, install time. The Dalvik VM is replaced by the Android RT (ART). The Java byte code compiler and runtime versions have to match the Java language features supported by the ART which trails the latest and greatest Java language features from Java.Oracle.com. In a future version of AS that better supports the Android NDK, to match Java source and target ART compatibility, go to “File”/“Project Structure…", select "app" under "Modules" heading, "Properties" tab, set both the "Source Compatibility" and "Target Compatibility" list box drop-downs to match. For now, to support development with AS and the “NDK Preview”, see step 2 of this section.

5. AS is bundled with the Android SDK, like the old Eclipse ADT Bundle. And like the Eclipse ADT bundle installation, the Android SDK must be installed outside of the AS “Application” installation folder. In AS, see configured Android SDK path setting: go to "File"/"Project Structure...", select the "SDK Location" heading, and use the browse button to set the file path in the "Android SDK location:" textbox.

6. Android NDK allows application developers to build performance-critical portions of their apps by use of native (C/C++) code. Developing apps using the Android NDK is not seamlessly supported as of AS version 1.4. Therefore, setting up AS to use the “NDK Preview” is a bit cumbersome. Start by downloading and installing the NDK using the AS IDE. In the IDE, go to "File"/"Project Structure...", click on the "SDK Location" panel. Under the "Android NDK location:" heading, click the "Download Android NDK" hyperlink. This downloads and installs the very large NDK file set to under the Android SDK root directory in the sub-directory of “ndk-bundle.” At the time of this document draft, the NDK version downloaded by AS version 1.4 IDE is “r10e” - the 64-bit version for Mac OS X.

   To be safe, create and expose the following environment variables that have as values the full path to the “NDK Preview” file set directory; full path to the root ”ndk-bundle” directory:

   ANDROID_NDK_ROOT
   NDK_ROOT
   NDK

7. The “NDK Preview” setup requires a non-default AS build system configuration. The build system used by AS is Gradle. See the section “The Android Plugin for Gradle and the Gradle Build System” for details about the Gradle-based AS build system. The NDK Preview requires Gradle version 2.5 and the

Experimental Android Plugin for Gradle must be installed. Gradle 2.5 requires Android SDK Build Tools version of 19.0.0 or greater. In the AS IDE, open the "Project" window and the "Android" view. Open the "Gradle Scripts" twirl, double click and open the file: "gradle-wrapper.properties." In the file, change the zip file name embedded in the value of "distributionUrl," to "gradle-2.5-all.zip." Then click “File"/"Synchronize". The AS IDE should download and install Gradle 2.5. If AS doesn't automatically download and re-synchronize to the new gradle version. Or a new version of AS deletes the installation, use the "distributionURL" to manually download the zip file and copy the unzipped folder to the file system path of "/Applications/Android Studio.app/Contents/gradle/“. Remember, to see the file system elements in Finder under “Android Studio.app”, right-click “/Applications/Android Studio.app” and select “Show Package Contents”.

After Gradle 2.5 is installed, double click and open the top-level, project “build.gradle" file, located in the Android view under the "Gradle Scripts" twirl. In the "dependencies" block, change the "classpath" member to have the value of 'com.android.tools.build:gradle-experimental:0.2.1'

The GitHub "artoolkit/ARToolKit5-dev" Repository

ARToolKit SDK (ARTK) Documentation

https://github.com/artoolkit/artoolkit-docs

ARTK Download, Installation and Setup

1. Go to ARToolKit.org and download the ARToolKit for Android developed on Mac OS X (or Windows 8.1). Copy ARToolKit SDK root directory and the entire file set to a development directory
2. Build the ARTK shared object libraries, file name extension ".so" (".DLL" for Windows), and static libraries, file name extension ".a" (.LIB for Windows), using the development system's compiler. Using a command-line shell window, change to the "android" directory under the ARToolKit SDK development root directory. Execute the script, build.sh (build.bat for Windows). Look to see that all builds without error.

Eclipse versus IntelliJ and Android Studio (AS)

Nomenclature

Eclipse "Workspace" A collection of related group of projects versus IntelliJ "Project" A collection of related groups of modules.

Eclipse "Project" is the same as IntelliJ "Module" - both can have custom JDK level configured, i.e. not Eclipse "Workspace"/IntelliJ "Project" wide configuration.

Build Systems

- Eclipse uses Ant
- Android Studio uses the Gradle build system that is written with Java. Gradle’s programming language is called Groovy. Android Studio uses a domain specific language (DSL), a dialect of Groovy. See the section, “The Android Plugin for Gradle and the Gradle Build System” for details.

Workflow to Migrate the Example ARTK Eclipse Projects from the Eclipse IDE to the AS IDE

Migration Pre-steps using the Eclipse IDE

1. Open the Eclipse IDE, open the root directory of the download ARToolKit SDK file set, that is setup as your ARToolKit development directory, as the Eclipse workspace (an Eclipse workspace is just a directory). Import an existing project found in the “EclipseProjects” directory of the ARTK SDK development root directory: use "File"/"Import", under the "General" twirl, select "Existing Projects into Workspace," and click "Next >". With "Select root directory:" radio button selected, click the "Browse..." button. Browse to an ARTK SDK development root directory and, under “EclipseProjects” directory, select the Eclipse project root directory to import, click "Open". Uncheck "Copy projects into workspace," and click "Finish".
2. Since each example ARTK Eclipse Projects depends on the "EclipseProject" project of "ARBaseLib", the "ARBaseLib" project needs to be imported into the Eclipse workspace as well. As long as the app and the "ARBaselib" library projects are in the same workspace, the app will build and run.
3. Set the path to the NDK repository (NDK file set root directory). Go to "Eclipse"/"Preference...", open the "Android" twirl, select "NDK". Click the "Browse..." button and browse and select root directory of the NDK repository. If the NDK was downloaded and installed by way of Android Studio, the NDK root directory will be under the Android SDK with the root folder name of “ndk-bundle”. See the section “Android Studio (AS) Development Platform Requirements (for Mac OS X, v.11),” item #6, for the location of the NDK repository.
4. Set the project build target (Android API level). Right click on the project in the Package Explorer, select "Properties". In left side margin, click and select "Android." Under the "Project Build Target" section, the "Target Name" heading, select the target API level. If the desired target is not displayed, it must be loaded though the Android SDK Manager. The level selected here needs to match the compileSdkVersion and targetSdkVersion properties will later be targeted and configured by the AS IDE.
5. Build the projects. Click "Project"/"Clean..." then click "Project"/"Build All". This creates: "armeabi", "armeabi-v7a", "mips", "x86" shared object and library variants of, respectively, "lib*.so", "lib*.a" dependencies.

Migrating a Eclipse workspace-project to an AS project-module

Eclipse ADT supports project exports that setup to use the Gradle build system but this export method keeps the Eclipse Android project source code structure layout. This exports generates a folder named “gradle” in the AS project root directory; and inside the “gradle” folder, three files named “build.gradle”, one file named “gradlew", and one file called “gradlew.bat”.

However, as of AS version 0.4.x, the above Eclipse ADT project export functionality is no longer needed and not recommended. The recommended migration function is to use the AS IDE import functionality. The AS IDE can directly import Eclipse Android projects and modernize the source code structure layout. First ensure the Eclipse project being imported does not contain the above Eclipse generated gradle directory and files.

Steps:

1. Setup the ARToolKit SDK, install and configure the AS development environment requirements and prepare the Eclipse project being migrated to AS. Respectively, see the "ARTK Download, Installation and Setup," "Android Studio (AS) Development Platform Requirements (Mac OS X)" and "Migration Pre-steps using the Eclipse IDE" sections.
2. Pre-AS 1.0: Open AS and from welcome screen, select "Import Project..." under the "Quick Start" group box.
   Post-AS 1.0: Open AS and from welcome screen, select "Import Non-Android Studio project" under the

"Quick Start" group box.
Post-AS 1.4: Open AS and from welcome screen, select "Import project (Eclipse ADT, Gradle, etc.)" under the "Quick Start" group box.

a) In the next dialog box, which may be mislabeled as "Select build.gradle or settings.gradle" in old versions of AS, select the Eclipse project base/root directory under the ARToolKit development directory of "EclipseProjects" you wish to import from and click "OK".

b) In the next dialog box, browse to the new destination directory, under the ARToolKit development directory, under "AndroidStudioProjects" directory, create a new project folder (in AS called a module) with the filename that matches the Eclipse project being migrated from. Remember the Eclipse workspace and the AS project are the same in concept.

c) In the next dialog box, leave all options checked. Leaving "Create Gradle-style (camelCase) modules names" checked using the gradle project and module naming convention. Names the imported module as "app."

d) An "import-summary.txt" file is generated - read to see what worked and what didn't in import process.

e) Click "File"/"Save All" and then "Android Studio"/"Quit Android Studio"

3. Edit the project build.gradle file, the module build.gradle files and the gradle-wrapper.properties file to support the experimental Android Plugin for Gradle, version "0.2.1". Close the Android Studio project and quit the AS IDE. Then use the file system and a "real" text editor to do the following edits outside of the AS IDE.

a) In the Android Studio project, for every project module that was imported from Eclipse, open the modules' AndroidManifest.xml file and delete the XML "uses-sdk" element that looks something like this:

```
<uses-sdk android:targetSdkVersion="18" android:minSdkVersion="8"/>
```

The target, compile and minimum Android SDK usage versions are configured in the modules' build.gradle file by Android Studio.

b) See the "Sample Top-Level, Project 'build.gradle', Module App and Library 'build.gradle' and gradle-wrapper.properties Files" section to see how the files have to be edited and changed.

c) It may be necessary to re-target to a different Android platform. This can be done from within AS or by editing the build.gradle file directly using a text editor while the AS project is closed. To do the edits from within AS, go to project window and changing it to the project view (as opposed to the Android view), open "app" folder and double click to open the "build.gradle" file - the "app" module, gradle build configuration file, "build.gradle." Change the "compileSdkVersion," under "android" block, and the "targetSdkVersion," under "android/defaultConfig" block, to the same and desired API Level (usually the latest and greatest API level). The selected and configured API level must be installed by the Android SDK Manager. You may want to bump up the "minSdkVersion," under "android/defaultConfig" block, as well.

d) It may be necessary to re-target to a different "Android Support Library" and "Android Support Repository" package version when the Android Studio IDE and Android SDK don't agree on the version. The support libraries package contains libraries supporting a specific range of android platforms and set of features. Note: The "Android Support Repository" is a requirement of developing apps with Android Studio. These packages are downloaded and maintained by the Android SDK Manager, under the "Extras" folder/section.

e) The following edits can be done within or outside of AS. In the module "build.gradle" files, see the lines beginning with "compile" in a "dependencies" block. Find the lines that contain the value "'com.android.support:" + "support-v4:" or "appcompat-v7:", or both, or "design:" followed by version number formatted as: "xx.x.x"'. "com.android.support," "support-v4," "appcompat-v7," "design" and the version number "xx.x.x" strings translates to file system paths under the ASDK directory location. Run the ASDK Manager, open the "Extras" section and look to the right of the "Android Support Library"

item. Copy the "xx.x.x" version string next to "Installed." If the version string is different between the AS IDE and THE ASDKM, paste the ASDKM version string to replace the version string in the module “build.gradle” file. Save changes and click "Try Again" in the module “build.gradle” window. Reload the project if prompted.

4. Optional. If you wish to create a separate Android Studio module for a dependency, this step is necessary. To add dependencies modules to the AS project and set the app module to resolve to the dependency module, see the section "The Dependencies Types of Libraries, Files or Modules and the AS Module".
5. In the app module’s “build.gradle” file source, add "dependencies" block for module dependencies, like aRBaseLib, and the “Android Support Library” package, if required.
6. After editing the project “build.gradle” file, the module build.gradle files, the gradle-wrapper.properties file and after reopening the Android project, the following AS IDE gradle sync error may appear: "Error: Please fix the project's Gradle settings." If so, click the "Fix Gradle wrapper and re-import project Gradle settings" hyperlink. This takes you to a Project-level settings dialog box. Click and browse for the new "Gradle home:" path. Set the path to "/Applications/Android Studio.app/Contents/gradle/gradle-2.5." Sometime, using the AS IDE, the gradle-wrapper.properties, "distributionUrl" field requires the zip file component in that URL to be reset back to "gradle-2.5-all.zip". Re-Sync Gradle.
7. If there remains an error having to do with one or more missing dependency libraries, this is most likely due to the dependency declaration not being configured for module's “build.gradle” file. Use the AS IDE to update the configuration of the module's build.gradle file by right clicking the module in the "Android" or "Project" view, select "Open Module Settings". In the left side, under the "Module" heading, select the module missing the dependency declaration, click the "Dependencies" tab and click the "'+" button on the bottom left of the window. Depending on the type of missing dependency, select "Library," "File" or "Module" and then select the dependency. The module's build.gradle "dependencies" block is updated with the dependency declaration. For further information, see the section "The Dependencies Types of Libraries, Files or Modules and the AS Module".

The Dependencies Types of Libraries, Files or Modules and the AS Module

"Library" and "File" Dependencies (as external 3rd party binaries or source)

Are references to classes, archives, libraries and resources, not imported and are built through a reference string used by the build system that references a repository. Usually a software library known and recognized to and by the build system. Adding a remote library dependency through Maven Central (remote repository of libraries). In the project level build.gradle file, see jcenter() which hosts remote repository dependencies. jcenter() is a RepositoryHandler interface method returning a org.gradle.api.artifacts.repositories.MavenArtifactRepository instance, a URL to https://bintray.com, a software distribution service. Allows you to download libraries, on demand.

Adding a library (local or remote Maven-like repository) dependency using a reference string

File/Project Structure..., under the Modules heading, select the "app" module, select Dependencies tab, click + button and select 1 Library dependency. Select the library dependency from the Maven Central list (search and select "gson" as an example) and click OK, OK. In the App module build.gradle file "compile 'com.google.code.gson:gson:2.3.1'" is added.

Adding (and deleting) a single dependency file, internal to app module or as an external library module

a) Internal to app module: To add a *.jar file, a *.jar file to just be used by the app module, in project window and view, open up the "libs" folder under the app module folder. Go to file system a and copy *.jar file (probably in the download directory). In the AS IDE, paste over the "libs" folder - this

places a copy in the "libs" directory in the file system. The *.jar file shows up in the AS IDE under the app module's "libs" folder. Right click on the *.jar file and go to and select "Add As Library..." in the context menu. Select the correct module name, and click OK. Gradle syncs. This adds a "compile files('libs/[blah-blah].jar')" line in the "dependencies" block in the module's "build.gradle" file. To do the same thing, use "File"/"Project Structure...", under the "Modules" heading, select the module's panel, and click the module's "Dependencies" tab. Then click the "+" button in the lower left of the module's panel, for a jar file, select "File dependency" and, in the browse dialog box, select the jar file. (To delete a dependency, in the "Dependencies" tab, select the dependency and use "-" button to clear the dependency configuration in the build.gradle file. The dependency object still needs to be deleted from "libs" file system directory - in project view, right click delete on *.jar file).

b) External library module: Making the *.jar file a standalone module, within the AS project, and then adding the new module as a dependency of the app module. Go to "File"/"New"/"New Module...", select "Import .JAR/.AAR Package" (imports an existing JAR or AAR package file as a new project module), click Next. (Note: to create a source library module, go to "File"/"New"/"New Module..." and select "Android Library".) In the next dialog box, use the File name: browse button to browse and select the *.jar file. Accept inserted "Subproject (module) name:" value. Click the Finish button. Go to "File"/"Project Structure..." dialog box. Notice there is a new module under the" Modules" heading, in addition to the "[app]" module. Select the "[app]" module and then "Dependencies" tab. Click the "+" button, select "3 Module dependency", select the just created module dependency by it's "Subproject" name and click OK twice. Gradle syncs. In the "[app"] module's build.gradle file "compile project('[new Subproject name]')" is added. More *.jar files are easily added to the dependency module that the app module will be able to access and use.

"Module" Dependencies as Source Code

Is a set of class files stored in a local archive file or directory, all imported and packaged, added to project under "app/libs", explicitly. In the module build.gradle file, explicit libraries are added with this gradle compile command:

"compile files('libs/foo-1.4.5.jar')," or "compile fileTree(dir: 'libs', include: ['*.jar'])" within the "dependencies" block.

Two ways to add a dependency as source code, internal to the app module or externally as an module library

a) Internal to app module: "File"/"New"/"New Module...", select "Android Library." Name your library, probably something other than "lib" and set Compile with, Target SDK and Minimum required SDK to match the app module which has the dependency. Change the package name to not the app's package name but named to be a standalone library that many apps can use. Uncheck the "Create Activity" option (generally not needed for library code) and click "Next" and "Finish". Right click the new module in AS project view, select "New"/"Java Class." Name the Java Class source and add the source in the newly created .java file. Now in the app that has a dependency on the new library module, in the project view right click on the app module, select "Open Module Settings," under the Modules heading on the left, click the module to have the dependency, then click the "Dependencies" tab in the right window. Click the '+' button on the bottom left and select "3 Module Dependency" and select the library module to have the app depend on. Add an "import" statement to all source using the classes defined by the library module. The app build.gradle file is dependencies block is updated with a "compile project(':[library module name]')".

b) External library module (e.g. in a Eclipse project): "File"/"New"/"New Module...", select "Android Library." Name your library, probably something other than "lib" and set Compile with, Target SDK and Minimum required SDK to match the app module which has the dependency. Use the same module name (project name in the case of Eclipse) and package name used by the previous dev IDE. Uncheck the "Create Activity" option (generally not needed for library code) and click "Next"

and "Finish". Open the AndroidManifest.xml file in the just added library module and delete the <application/> element if it exist (since the module is supposed to be a library). using the file system, copy the java class source file to the new bottom-most package directory for the library module - the AS IDE should see and refresh the module to see and show the new source file. Update the dependent app module to depend on the new module library dependency. Right click the dependent app module in the project view, select "Open Module Settings," under the Modules heading on the left, click the module to have the dependency, then click the "Dependencies" tab in the right window. Click the '+' button on the bottom left and select "3 Module Dependency" and select the library module to have the app depend on. Add an "import" statement to all source using the classes defined by the library module. The app build.gradle file is dependencies block is updated with a "compile project(':[library module name]')".

The Android Plugin for Gradle and the Gradle Build System

The build system used by AS is Gradle. Gradle is integrated with AS through a plugin, called the Android Plugin for Gradle, and other general build tools are provided both by Android Studio IDE (AS IDE) and by the Android SDK manager (ASDKM). Therefore the build tool configuration between the AS IDE and ASDKM can get out of synch. After a “Android SDK Build-tools” update through the ASDKM, you may have to let the AS IDE know about the newer Android Plugin for Gradle version (opposed to the integrated Gradle build system version). This is done by updating the AS IDE top-level, project level, "build.gradle" file (while the integrated Gradle build system version is defined in the gradle-wrapper.properties file). To understand what Android plugin for Gradle version is supported by a specific “Android SDK Build-tools” version, consult the developer.android.com website.

Gradle and Android Studio Nomenclature Difference

- AS Module is equivalent to a gradle project scoping
- AS Project and the root level Android project directory is equivalent to all gradle projects scoping

The Difference between the Android Plugin for Gradle version and the Gradle Build System Version

To see Android plugin for Gradle version:

a) Project Windows, Project View, project root folder, open the top-level , project "build.gradle" file. In the file see the "buildscript" block, "dependencies" sub-block, the "classpath" property value-part, e.g. 'com.android.tools.build:gradle:1.3.0'. Hence, the Android plugin for Gradle version is "1.3.0".

b) “File”/“Project Structure…"/"Project" pane, "Android Plugin Version" textbox, “1.3.0”.

To see the integrated Gradle build system version:

a) Open project window, project view, project root folder, "gradle" folder, "wrapper" folder, open "gradle-wrapper.properties". In the file, see "distributionURL" property value-part, e.g. "https://services.gradle.org/distributions/gradle-2.4-all.zip". Hence, the gradle version is “2.4”.

b) "File"/"Project Structure..."/"Project" pane, "Gradle version" textbox, “2.4”.

Known AS and Gradle Issues

1. The Gradle build system version number, e.g. ”x.y,” “x.y.z" or "x.y.z+" and the Android Plugin for Gradle (contained in Android SDK Build Tools) version number, e.g. ”u.v.w” or “u.v+", are different.

   Within the project level “build.gradle”, the "top-level build” file, in the “buildscript" block and then the “dependencies” sub-block, contains a “classpath" variable with the value of 'com.android.tools.build:gradle:u.v.w.' This is the Android plugin for gradle version number. The plugin comes bundled with the gradle build system version, “x.y”, the plugin uses. Sometimes this version string will be specifying the gradle build system version formatted like this "x.y" when it should strictly be specifying a specific Android plugin for gradle version number, "u.v.w" or newer version numbers, “u.v.+." Insure that the “classpath” version string is defined in reference to the Android plugin for gradle version.

2. Sometimes when AS version 1.0+ imports or opens an Eclipse project or an old AS project, after a gradle build, the following error result:

```
Error:(16, 0) Gradle DSL method not found: 'android()'

Possible causes:

   The project 'Teapot' may be using a version of the Android Gradle plug-in that does
   not contain the method (e.g. 'testCompile' was added in 1.1.0).

      Fix plugin version and sync project.
```

```
The project 'Teapot' may be using a version of Gradle that does not contain the
method.
   Open Gradle wrapper file.
The build file may be missing a Gradle plugin.
   Apply Gradle plugin
```

To fix this, remove the following from the top-level, project “build.gradle” file:

```
android {
   buildToolsVersion'23.0.1'
}
   dependencies {
}
```

3. Run-time exception for a modern Java compiler feature not support by the Android runtime:

```
:app:preDexDebug
UNEXPECTED TOP-LEVEL EXCEPTION:
Error:com.android.dx.cf.iface.ParseException: bad class file magic (cafebabe) or
version (0034.0000)
   at com.android.dx.cf.direct.DirectClassFile.parse0(DirectClassFile.java:472)
   at com.android.dx.cf.direct.DirectClassFile.parse(DirectClassFile.java:406)
   at
   com.android.dx.cf.direct.DirectClassFile.parseToInterfacesIfNecessary(DirectClassFi
   le.java:388)
   at com.android.dx.cf.direct.DirectClassFile.getMagic(DirectClassFile.java:251)
   at com.android.dx.command.dexer.Main.parseClass(Main.java:764)
   at com.android.dx.command.dexer.Main.access$1500(Main.java:85)
   at com.android.dx.command.dexer.Main$ClassParserTask.call(Main.java:1684)
   at com.android.dx.command.dexer.Main.processClass(Main.java:749)
   ... 12 more
1 error; aborting
Error:Execution failed for task ':app:preDexDebug'.
> com.android.ide.common.process.ProcessException:
org.gradle.process.internal.ExecException: Process 'command '/Library/Java/
JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home/bin/java'' finished with non-zero
exit value 1
```

To Fix this, go to "File"/"Project Structure…", click "SDK Location" in right side column, under "JDK location:" header, click "..." browse button. Change the file system path:

“/Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home”

to

“/Library/Java/JavaVirtualMachines/jdk1.7.0_79.jdk/Contents/Home”

This changes the JDK that Android Studio uses.

Sample Top-Level, Project "build.gradle", Module App and Library "build.gradle" and "gradle-wrapper.properties" Files

Top-Level, Project “build.gradle” File

Important change to notice is the experimental Android Plugin for Gradle: "gradle-experimental:0.2.1"

// Top-level build file where you can add configuration options common to all sub-projects/modules.

```
buildscript {
    repositories {
        jcenter()
    }

    dependencies {
        classpath 'com.android.tools.build:gradle-experimental:0.2.1'

        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        jcenter()
    }
}
```

Module App “build.gradle” File

Notice the "apply plugin" value: 'com.android.model.application' - this different between app and library modules. Notice the “ndk” section (ndk.with) below to integrate C++ code through the Java JNI. Notice the use of the '=' assignment character - it must be used for the experiment Android plugin for Gradle and the associated Groovy DSL. The possible gradle error that can result when the assignment character is omitted: "Error:Cause: org.gradle.api.internal.ExtensibleDynamicObject."

```
apply plugin: 'com.android.model.application'
model {
    android {
        compileSdkVersion = 23
        buildToolsVersion = "23.0.1"

        defaultConfig.with {
            applicationId = "org.artoolkit.ar.samples.ARSimpleNative"
            minSdkVersion.apiLevel = 22
            targetSdkVersion.apiLevel = 22
            versionCode = 1     //Integer type incremented by 1 for every release, major or minor, to Google store
            versionName = "1.0" //Real fully qualified major and minor release description

            buildConfigFields.with { //Defines fields in the generated Java BuildConfig class, in this case, for
                create() {           //default config, that can be accessed by Java code
                    type = "int"     //e.g. "if (1 == BuildConfig.VALUE) { /*do something*/}".
                    name = "VALUE"   //See: [app or lib]/build/generated/source/buildConfig/[package path]/
                    value = "1"      //     BuildConfig.java
                }
            }

            ndk.with {
                moduleName = “nftSimple"
            }
        }
    }

    android.buildTypes {
        release {
            minifyEnabled = false
```

```
            proguardFiles += file('proguard-rules.pro')
        }
    }

    android.productFlavors {
    }

    android.sources {
    }

    android.compileOptions {
        sourceCompatibilityJavaVersion.VERSION_1_7
        targetCompatibilityJavaVersion.VERSION_1_7
    }
}

dependencies {
    compile project(':aRBaseLib')
    //compile 'com.android.support:support-v4:23.0.1'
    //compile 'com.android.support:appcompat-v7:23.0.1' //Only required when the target device API level is greater than
}                                                       //the compile and target of the app being deployed to the device
```

Module Library “build.gradle” File

Notice the "apply plugin" value: 'com.android.model.library' - this different between app and library modules. Notice the use of the '=' assignment character - it must be used for the experiment Android plugin for Gradle and the associated Groovy DSL. The possible gradle error that can result when the assignment character is omitted: "Error:Cause: org.gradle.api.internal.ExtensibleDynamicObject."

```
apply plugin: 'com.android.model.library'

model {
    android {
        compileSdkVersion = 23
        buildToolsVersion = "23.0.1"

        defaultConfig.with {
            minSdkVersion.apiLevel = 22
            targetSdkVersion.apiLevel = 22
            versionCode = 1     //Integer type incremented by 1 for every major or minor release to Google store
            versionName = "1.0" //Real fully qualified major and minor release description

            buildConfigFields.with { //Defines fields in the generated Java BuildConfig class, in this case, for
                create() {           //default config, that can be accessed by Java code
                    type = "int"     //e.g. "if (1 == BuildConfig.VALUE) { /*do something*/}".
                    name = "VALUE"   //See: [app or lib]/build/generated/source/buildConfig/[package path]/
                    value = "1"      //     BuildConfig.java
                }
            }
        }
    }

    android.buildTypes {
        release {
            minifyEnabled = false
            proguardFiles += file('proguard-rules.txt')
        }
    }

    android.productFlavors {
    }

    android.sources {
    }

    android.compileOptions {
        sourceCompatibilityJavaVersion.VERSION_1_7
        targetCompatibilityJavaVersion.VERSION_1_7
    }
}

dependencies {
    compile 'com.android.support:support-v4:23.0.1'
    //compile 'com.android.support:appcompat-v7:23.0.1' //Only required when the target device API level is greater than
}                                                       //the compile and target of the app being deployed to the device
```

[AS project directory]/gradle/wrapper/gradle-wrapper.properties File

To open to edit the “gradle-wrapper.properties” file, in the AS IDE, open the project window and then the project view. Open the “gradle” twirl, followed by opening the “wrapper” twirl and double click “gradle-wrapper.properties” file. See the “distributionUrl” property. Insure that the gradle version used for development with the “NDK Preview” is “2.5”. Look to see that the Gradle zip file is named “gradle-2.5-all.zip”. (Not to be confused with the experimental Android Plugin for Gradle, version “0.2.1.”)

```
#Wed Apr 10 15:27:10 PDT 2013
distributionBase=GRADLE_USER_HOME
distributionPath=wrapper/dists
zipStoreBase=GRADLE_USER_HOME
zipStorePath=wrapper/dists
distributionUrl=https\://services.gradle.org/distributions/gradle-2.5-all.zip
```

Android Studio and ARToolKit SDK Example Apps

ARToolKit SDK example apps and source code is now provided as Android Studio projects. The Android Studio integrated development environment is now the officially supported development tool for Android developers by developer.android.com and Google.

To work with the ARToolKit SDK example app Android Studio projects, clone or fork “artoolkit/artoolkit5” from github.com. Once downloaded, in your repository, change directory to “[the repository root]/AndroidStudioProjects.” Startup Android Studio and select to open the root app project directory under “AndroidStudioProjects”. Note that an Android Studio project is simply a directory that contains one or more Android Studio app or library modules. It’s customary for the project name to match the name of a single app type module contained within the project.

Once the project is open, Android Studio should automatically use the integrated gradle build system to build the app. At that point, that app can be deployed and executed on an Android device, just click Run and select the target real or virtual device. Remember, webcam capture is not possible on virtual devices.