Android Studio Project Site

Preview Releases
Downloads & Preview Channels
Recent Changes
New Build System

Resources & Feedback
Technical docs
Known Issues
Feedback
Filing Bugs
Tips

Open Source
Projects Overview
Build Overview
Contributing

Social Media
@AndroidStudio
+Android Studio

Technical docs > New Build System >

Experimental Plugin User Guide

Contents

Introduction

The new experimental plugin is based on Gradle's new component model mechanism, while allows significant reduction in configuration time. It also includes NDK integration for building JNI applications. This user guides provides details on how to use it and highlights the difference between the new plugin and the original plugin.

CAVEAT: Note that this plugin is a preview of the plugin for feedback on performance and NDK integration. The Gradle API for the new component model is not final, which means each plugin will only work with a specific version of Gradle.

Additionally, the DSL may also change.

Latest Version

Please check the bintray repository for the latest version.

Requirements

- Gradle (see section below for version required)
- Android NDK r10e (if you are using NDK)
- SDK with Build Tools at least version 19.0.0 and we aim to minimize the amount of changes needed for the migration process in the future. Some features may require a more recent version.

Gradle Requirements

Each version of the experimental plugin requires a specific version of Gradle. Here is a list of required Gradle version.

Plugin Version	Gradle Version
0.1.0	2.5
0.2.0	2.5
0.3.0-alpha3	2.6
0.4.0	2.8
0.6.0-alpha1	2.8
0.6.0-alpha5	2.10
0.7.0-alpha1	2.10
0.7.0	2.10
0.7.3	2.14.1

Migrating from Traditional Android Gradle Plugin

A typical Android Studio project may have a directory structure as follows. File that needs to be change is highlighted in red:
There are some significant changes in the DSL between the new plugin and the traditional one.

```
.
├── app/
│   ├── app.iml
│   ├── build.gradle
│   └── src/
├── build.gradle
├── gradle/
│   └── wrapper/
│       ├── gradle-wrapper.jar
│       └── gradle-wrapper.properties
├── gradle.properties
├── gradlew*
├── gradlew.bat
```

```
├── local.properties
├── MyApplication.iml
└── settings.gradle
```

./gradle/wrapper/gradle-wrapper.properties

- Each version of the new plugin supports a specific Gradle version. Consult Gradle Requirements for the Gradle version to use.

```
#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/
2.10-all.zip
```

./build.gradle

- Classpath for the plugin is com.android.tools.build:gradle-experimental instead of com.android.tools.build:gradle.

```
// 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:0.7.0-alpha4"

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

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

./app/build.gradle

There are significant changes to the DSL of the plugin. We understand that many of the changes are frustrating and seem unnecessary, and our goal is to remove some of these current changes to minimize the migration process from the traditional plugin in the future.

NOTE: There has been significant DSL improvements starting from version 0.6.0-alpha5 compare to previous versions. The example code here will not work for previous version. If you are using an older version of the plugin. Please refer to the user guide at https://sites.google.com/a/android.com/tools/tech-docs/new-build-system/gradle-experimental/0-4-0.

DSL Changes:

- Plugin name is `com.android.model.application` instead of `com.android.application`. Or use `apply plugin: "com.android.model.library"` if you want to create an Android aar library.
- Configuration is wrapped with the `model { }` block
- Adding elements to a Collection should be done using the add method.

Current DSL Limitations that will hopefully go away:

- List properties are only set with their direct types only, with no way to accept other types and adapting them. For instance:
  - You can set a property of type File using a String, but a property of type List<File> accepts only File objects.
- Creating a buildType or productFlavor requires calling the create method. Modifying an existing one such as the release and debug buildType can be done using the just the name.
- The DSL for modifying variants and their tasks is very, very limited right now.

```
apply plugin: "com.android.model.application"

model {
    android {
        compileSdkVersion 23
        buildToolsVersion "23.0.2"

        defaultConfig {

           applicationId "com.example.user.myapplication"
            minSdkVersion.apiLevel 15
            targetSdkVersion.apiLevel 22
            versionCode 1
            versionName "1.0"

            buildConfigFields {
                create() {
                    type "int"
                    name "VALUE"
                    value "1"
                }
            }
        }
        buildTypes {
         release {
             minifyEnabled false
             proguardFiles.add(file("proguard-
rules.pro"))
         }
     }
        productFlavors {
         create("flavor1") {
             applicationId "com.app"
         }
     }

     // Configures source set directory.
     sources {
         main {
             java {
                 source {
                     srcDir "src"
```

```
                }
            }
        }
    }
}

dependencies {
    compile fileTree(dir: "libs", include: ["*.jar"])
    compile "com.android.support:appcompat-v7:22.2.0"
}
```

Signing Config

You can refer to another model element using the $() syntax. To use this syntax, "-Dorg.gradle.model.dsl=true" has to be added as an argument to the Gradle command line for version below 2.10. This is useful for specifying signing configs. NOTE: android.signingConfigs currently must be outside of the `android {}` block.

```
apply plugin: "com.android.model.application"

model {
    android {
        compileSdkVersion 23
        buildToolsVersion "23.0.2"
        buildTypes {
            release {
                signingConfig =
$("android.signingConfigs.myConfig")
            }
        }
    }
    android.signingConfigs {
        create("myConfig") {
            storeFile "/path/to/debug.keystore"
            storePassword "android"
            keyAlias "androiddebugkey"
            keyPassword "android"
            storeType "jks"
        }
    }
}
```

Ndk Integration

The experimental plugin comes with NDK integration for creating native applications. To use the NDK integration:

- Use the SDK Manager inside Studio to download the NDK.
- Set ndk.dir in local.properties or the ANDROID_NDK_HOME environment variable to the directory containing the NDK.
- Add an `android.ndk` block to the model in build.gradle.

The build.gradle of a simple NDK application may look like this:

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

model {
    android {
        compileSdkVersion 23
        buildToolsVersion "23.0.2"
        ndk {
            moduleName "native"
        }
    }
}
```

*Note that the moduleName is **required**. It determines the name of the resulting native library.

Source Set

By default, it will look in src/main/jni for C/C++ file. Configure android.sources to change the source directory.

```
model {
    android {
        compileSdkVersion 23
        buildToolsVersion "23.0.2"
        ndk {
            moduleName "native"
        }
        sources {
            main {
                jni {
                    source {
                        srcDir "src"
                    }
                }
            }
        }
    }
}
```

The JNI source set may contain both C and C++ files. All files in the sub-directories are included. Files with extension '.c' is considered as C files, whereas C++ files has may have any of the following extensions: '.C', '.CPP', 'c++', '.cc, '.cp', '.cpp', '.cxx'. Files may be excluded with the `exclude` method, whereas `include` is ignored:

```
model {
    android.sources {
        main {
            jni {
                source {
                    include "someFile.txt"  // This is
ignored.
                    exclude "**/excludeThisFile.c"
                }
            }
        }
    }
}
```

Other Build Options

Various build options can be set within the `android.ndk { }` block. For example,

```
model {
    android {
        compileSdkVersion 23
        buildToolsVersion "23.0.2"
        ndk {
            // All configurations that can be changed in
android.ndk.
            moduleName "native"
            toolchain "clang"
            toolchainVersion "3.5"
            // Note that CFlags has a capital C, which is
inconsistent with
            // the naming convention of other properties.
 This is a
            // technical limitation that will be resolved
            CFlags.add("-DCUSTOM_DEFINE")
            cppFlags.add("-DCUSTOM_DEFINE")
            ldFlags.add("-L/custom/lib/path")
            ldLibs.add("log")
            stl "stlport_static"
        }
        buildTypes {
            release {
                ndk {
                    debuggable true
                }
            }
        }
        productFlavors {
            create("arm") {
                ndk {
                    // You can customize the NDK
configurations for each
                    // productFlavors and buildTypes.
                    abiFilters.add("armeabi-v7a")
                }
            }
            create("fat") {
                // If ndk.abiFilters is not configured,
the application
                // compile and package all suppported
ABI.
            }
        }
    }

    // You can modify the NDK configuration for each
variant.
    components.android {
        binaries.afterEach { binary ->
            binary.mergedNdkConfig.cppFlags.add(
                    "-DVARIANT=\"" + binary.name + "\"")
        }
```

```
    }
}
```

ABI Specific Configurations

You can configure the settings for a specific ABI.

```
model {
    android {
        compileSdkVersion 23
        buildToolsVersion "23.0.2"
        ndk {
            moduleName "native"
        }
        abis {
            create("x86") {
                platformVersion "android-19" //
platformVersion option is available from 0.8.0.
                CFlags.add("-DX86")
            }
        }
    }
}
```

Known Limitations

- No support for using a NDK modules like cpu_features
- No support for integrating external build systems.

Samples

Additional samples can be found at https://github.com/googlesamples/android-ndk.

Multiple NDK Projects

Plugin 0.4.0 added the preliminary support for NDK dependencies and the ability to create just a native library. Please be aware this is a preview of the direction we are going and the implementation is not complete. Note that while it is possible to compile the native project for Gradle, editing and debugging support in Android Studio is not yet implemented.

Standalone NDK Plugin

In gradle-experimental:0.4.0, a new plugin is created to allow creation of just the native library without creating an Android application or library. The DSL is similar to the application/library plugin. The following example build.gradle can create a libhello.so from sources in "src/main/jni"

```
apply plugin: "com.android.model.native"

model {
    android {
        compileSdkVersion 23
        ndk {
            moduleName "hello"
        }
    }
}
```

Known Issues

- Editing support for the standalone plugin is not yet implemented in Android Studio
- Modifying a source file in the library does not automatically cause the application to re-link the new library when building an application.

NDK Dependencies

The syntax for specifying dependency follows the style of Gradle's future dependency system. You can set a dependency on an Android project or a specific file.

For example, let say you have a subproject in "lib" using the standalone NDK plugin:

lib/build.gradle:

```
apply plugin: "com.android.model.native"

model {
    android {
        compileSdkVersion 23
     ndk {
         moduleName "hello"
     }
     sources {
         main {
             jni {
                 exportedHeaders {
                     srcDir "src/main/headers"
                 }
             }
         }
     }
    }
}
```

Any projects with a JNI dependency will include the directories specified in the exportedHeaders.. You can add dependency on the lib project from your application for your JNI code:

app/build.gradle:

```
apply plugin: "com.android.model.application"

model {
    android {
```

```
        compileSdkVersion 23
        buildToolsVersion "23.0.2"
     sources {
         main {
             jni {
                 dependencies {
                     project ":lib1"
                 }
             }
         }
     }
    }
}
```

You can specify a build type and/or product flavor of your target project. Otherwise, the plugin will try to find the same build types and product flavor as your application. You can also specify the linkage type if you would like the native library to be linked statically. E.g.

```
model {
    android.sources {
        main {
            jni {
                dependencies {
                    project ":lib1" buildType "debug"
productFlavor "flavor1" linkage "static"
                }
            }
        }
    }
}
```

To declare a dependency on a file, create a prebuilt library and add dependency on the library. E.g.,

```
model {
    repositories {
        libs(PrebuiltLibraries) {
            prebuilt {
                headers.srcDir "path/to/headers"
                binaries.withType(SharedLibraryBinary) {
                    sharedLibraryFile =
file("lib/${targetPlatform.getName()}/prebuilt.so")
                }
            }
        }
    }
    android.sources {
        main {
            jniLibs {
                dependencies {
                    library "prebuilt"
                }
            }
        }
    }
}
```

翻译

You can add native dependency to either 'jniLibs' or 'jni' source set. When dependency is added to "jniLibs" the native library will be package into the application/library, but it will not be used for compiling the JNI code. E.g.

```
model {
```

```
    android.sources {
        main {
            jniLibs {
                dependencies {
                    library "prebuilt"
                }
            }
        }
    }
}
```

DSL Change

The plugin is still in experimental stage. DSL will change throughout the development of the plugin. This section documents the changes that occurs between different versions to help with migration.

0.6.0-alpha1 -> 0.6.0-alpha5

- Plugin now requires Gradle 2.10, which brings significant improvements to DSL
- Configurations can now be nested. E.g. you can write

```
android {
    buildTypes {
        ...
    }
}
```

instead of:

```
android.buildTypes {
    ...
}
```

翻译

- File type now accepts a string, but String cannot be added to List<File> at the moment.
- -Dorg.gradle.model=true is now the default. This allows references to other model, but the model being referred to must be in a separate block.
- The equal sign '=' is no longer required for most properties.

0.4.x -> 0.6.0-alpha1

- The DSL for specifying dependencies on a specific library files have changed to follow Gradle's native dependency DSL. (see https://github.com/gradle/gradle/blob/master/subprojects/docs/src/samples binaries/prebuilt/build.gradle)

```
model {
    android.sources {
        main {
            jniLibs {
                dependencies {
```

```
                    library
file("lib/x86/prebuilt.so") abi "x86"
                    library file("lib/armeabi-
v7a/prebuilt.so") abi "armeabi-v7a"
                    library
file("lib/mips/prebuilt.so") abi "mips"
                }
            }
        }
    }
}
```

is replaced by:

```
model {
    repositories {
        prebuilt(PrebuiltLibraries) {
            binaries.withType(SharedLibraryBinary)
{
                sharedLibraryFile =
file("lib/${targetPlatform.getName()}/prebuilt.so")
            }
        }
    }
    android.sources {
        main {
            jniLibs {
                dependencies {
                    library "prebuilt"
                }
            }
        }
    }
}
```

翻译

0.2.x -> 0.4.0

- += no longer works for collections. Adding items to the list can be done with the 'add' or 'addAll' method. e.g. `CFlags += "-DCUSTOM_DEFINE"` can be replaced with `CFlags.add("-DCUSTOM_DEFINE")`

0.1.x -> 0.2.x

- jniDebuggable is removed from build type configuration and moved to the ndk block. e.g.

```
release {
    jniDebuggable = true
}
```

becomes

```
release {
    ndk.with {
```

```
            debuggable = true
        }
    }
```

Change Log

0.6.0-alpha3

- DSL for specifying prebuilt libraries dependency has been changed.
- Updated to Gradle 2.8.
- Fixed various issues with native library dependency resolution.

0.4.0

- Fixed issue with using jni code in experimental library plugin.
- Allow platform version to be set separately from compileSdkVersion.
- Allow ABI specific configurations in a variant that contains multiple ABI.
- Added support for dependencies on NDK plugin and shared object/static library files.
- A preview version of an standalone plugin for compiling just native code is now available. It can be use to build application with Gradle, but support in Android Studio is not yet implemented.

Subpages (2): Experimental Plugin User Guide (Version 0.4.0) Migrate to Stable Gradle for NDK Support using CMake and ndk-build