Demystifying 64-bit development on Android™

Ramin Zaghi
Senior Applications Engineer
Partner Enablement Group, ARM

24 Sep 2015

Demystifying 64-bit development on Android™

Agenda

PART I: ARM® 64-bit Architecture in Android

PART II: Developing 64-bit Applications for Android

ARM

PART I: ARM® 64-bit Architecture in Android™

ARM

ARM 64-bit Architecture in Android™

Agenda

1. Why Should I Care?
2. What Has Changed?
3. ART: The New Virtual Machine
4. How Does It All Work?
5. What Difference Does it Make?
6. What Should I Do?

ARM

ARM 64-bit Architecture in Android™

Why Should I Care?

- It's going to be almost everywhere, and soon!
  - Already there are sub £100 phones with 64-bit cores
- 64-bit is *not* an automatic performance win, but:
  - Android only supports ARMv8 with 64-bit binaries (i.e. AArch64 and not AArch32)
  - Increased register number and register width
  - Improvements in the instruction set
  - Increased pointer size is mitigated in the design of the ART virtual machine
  - '64-bit only' is the increasing rule in the desktop space and mobile is likely to follow
- Switching to 64-bit mostly involves fixing bugs that you haven't noticed yet

ARM

ARM 64-bit Architecture in Android™

What Has Changed?

- There is no "64-bit-only" system but systems that support 64-bit as well as 32-bit
  - Also known as *Multilib* – the 64-bit ARMv8 AArch64, and 32-bit ARMv7 instruction sets
- The way your Android device launches an App is slightly different (see next slide)
- You get all the benefits as well as costs depending on what binaries/libraries you provide
- We think there is more long term scope for optimising for 64-bit
- Last but not least, there is a brand new and improved Virtual Machine (known as ART)

ARM 64-bit Architecture in Android™

ART: The New Virtual Machine

- Shorter garbage collection pause times
- Improved compilation techniques
- Suitable for both low-end and high-end Android devices (better on multi-core systems)
- Watch *"Google I/O 2014 - The ART runtime"* on YouTube
  https://www.youtube.com/watch?v=EBlTzQsUoOw

ARM

ARM 64-bit Architecture in Android™

ART: The New Virtual Machine

- ART supports 64-bit so Apps written only in Java™ benefit with no modification
- ART compiles your App's bytecode to native code at *install time*
  - This means shorter launch times
  - No Just-In-Time (JIT) compilation while your App is running
- 64-bit data types are now processed natively and with fewer instructions
- Optimised use of memory and cache
  - Java's object heap uses 32-bit object references (compressed pointers)

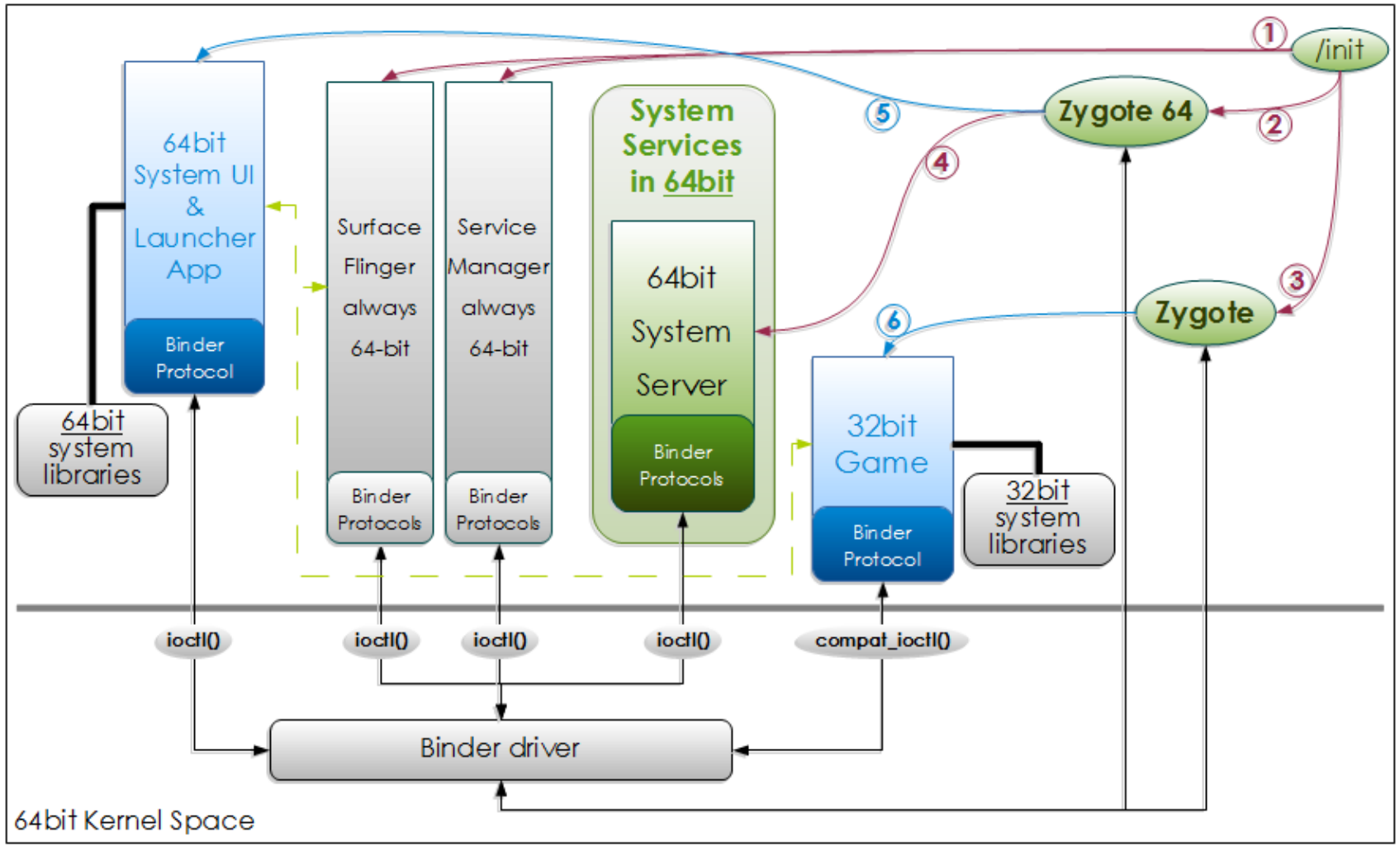
ARM 64-bit Architecture in Android™

How Does It All Work?

- All Apps are forked from a background Virtual Machine (VM) process called Zygote
- Multilib devices run two Zygotes (a 32-bit one and a 64-bit one) in parallel!
- When an application is launched on a 64-bit system
  - If it contains a supported 64-bit library, it runs as a 64-bit process against the loaded system
  - If it contains a supported 32-bit library, it is launched as a 32-bit process
  - Applications with no native code are launched using the default virtual machine (typically 64-bit)

ARM 64-bit Architecture in Android™

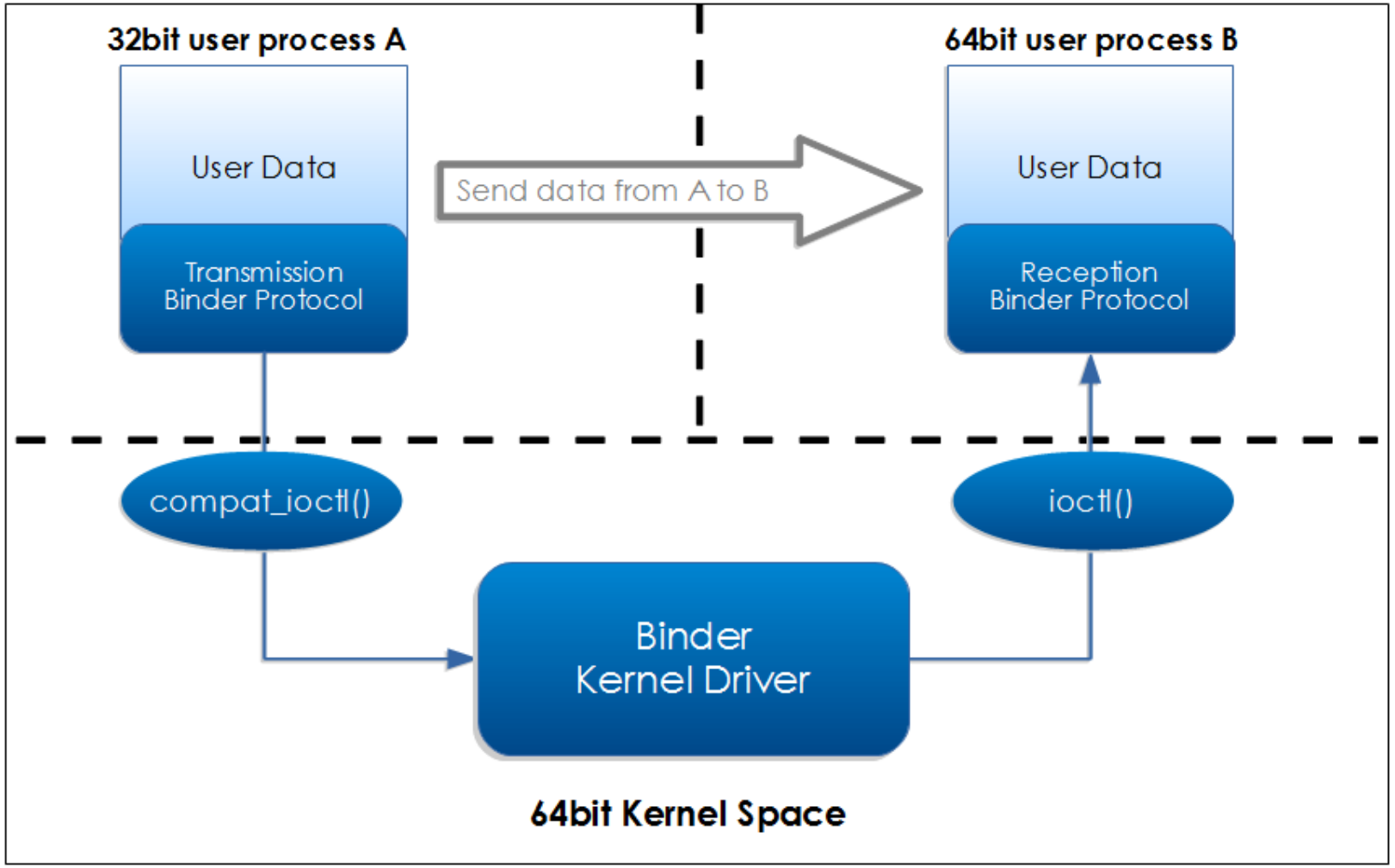
How Does It All Work?



ARM

ARM 64-bit Architecture in Android™

How Does It All Work?



ARM

ARM 64-bit Architecture in Android™

What Difference Does it Make?

- OS Images are larger & Take longer to build
  - Not really your problem if you are developing Apps…
- Increased binary size
  - But asset files (such as the graphics which make up most of an App) are unchanged
- Future-proofing done early rather than in a panic
- Huge chunks of Android built for AArch64 at least three or four years ago
- Lots of work done by Google (and a bit by ARM and others)

ARM

ARM 64-bit Architecture in Android™

What Should I Do?

- Have a pure Java™ application?
  - Just sit back and enjoy…
- Well written code will just work (most of the time)
  - You do pay attention to your types in C/C++ code, don't you?
- NEON™ code needs a little updating
  - Less if it uses intrinsics
- If you want to take full advantage of 64-bit
  - ARM has lots of training courses…

PART II: Developing 64-bit Applications for Android™

ARM

Developing 64-bit Applications for Android™

Agenda

1. Overview of developing Apps using Android Native Development Kit (NDK)
2. The old way: *Apache Ant*™ and *Android Development Tools (ADT)* for *Eclipse*
3. The new way: *Gradle*™ and *Android Studio*
4. Building as part of the *Android Open Source Project (AOSP)* source tree

"The NDK is a toolset that allows you to implement parts of your app using native-code languages such as C and C++. Typically, good use cases for the NDK are CPU-intensive applications such as game engines, signal processing, and physics simulation."

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

Developing 64-bit Applications for Android™

Overview of developing Apps using Android Native Development Kit (NDK)

1. Write Java™ code that links to native code through the JNI standard
   e.g. `public static native void SomeFunction();`

2. Compile your native code into shared object files
   e.g. *`libMyNative.so`*

3. Place them under one of the architecture specific directories under "*libs*"
   e.g. *`libs/arm64-v8a/libMyNative.so`*

4. Build your App's final package (i.e. the .Apk that can be installed)

Directory structure for a typical Android™ App project

http://malideveloper.arm.com

ARM

Developer Center - Mali D

malideveloper.arm.com

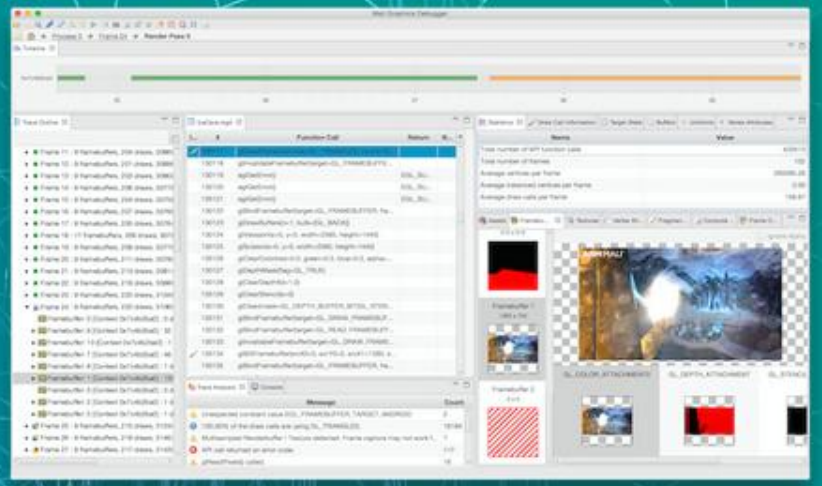
English | 简体中文

Everything you need to develop for Graphics and Compute

Mali DEVELOPER CENTER

Resources Documentation Demos Partners News & Events ARM Mali

ARM



Get Mali Graphics Debugger v3.1

Support for Android 6.0 (Marshmallow)

Developer Center - Mali D

malideveloper.arm.com

English | 简体中文

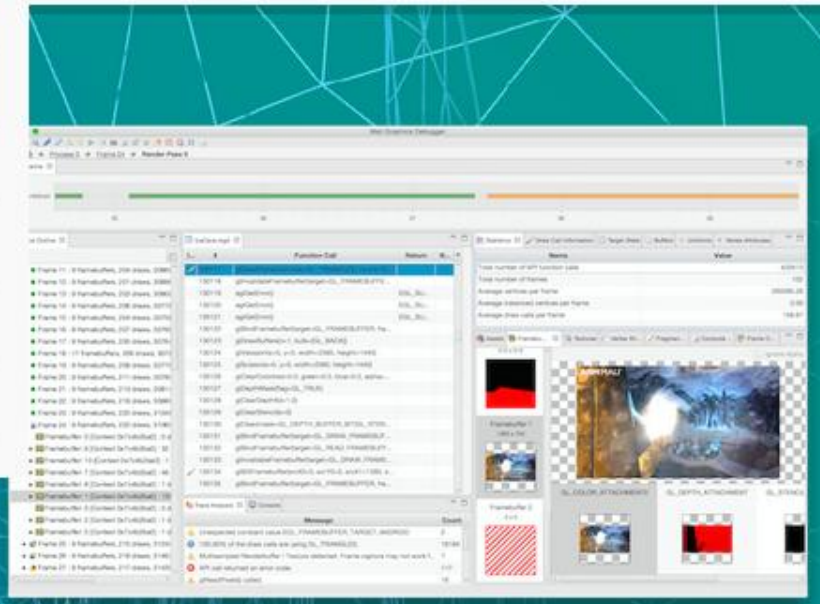
Everything you need to develop for Graphics and Compute

Mali DEVELOPER CENTER

Resources | Documentation | Demos | Partners | News & Events | ARM Mali

ARM

- SDKs
- Drivers
- Tools
- Sample Code
- Development Platforms



Get Mali Graphics Debugger v3.1

Support for Android 6.0 (Marshmallow)

SDKS

Mali VR SDK for Android

Mali OpenGL ES SDK for Linux

Mali OpenGL ES SDK for Android

Mali OpenCL SDK

DOCUMENTATION

× What's new in v2.0

- Added Particle Flow Simulation with Compute Shaders.
- Added Advanced Shading Techniques with Pixel Local Storage.
- Added Occlusion Culling with Hierarchical-Z.
- Added Introduction to compute shaders.
- Added Instanced Tessellation.
- Added Occlusion Query tutorial.
- Added Min Max Blending tutorial.
- Added Integer Logic tutorial.
- Added ETC2 Texture tutorial.
- Added Boids tutorial.
- Added Shadow Mapping tutorial.
- Added Instancing tutorial.
- Added Projected Lights tutorial.

A backlog of all older version features is available here

+ Changes in v1.6

+ Changes in v1.5

OpenGL ES 2.0 and Open GL ES 3.x SDK for Android

HTML Documentation included with installer package

Related Documentation

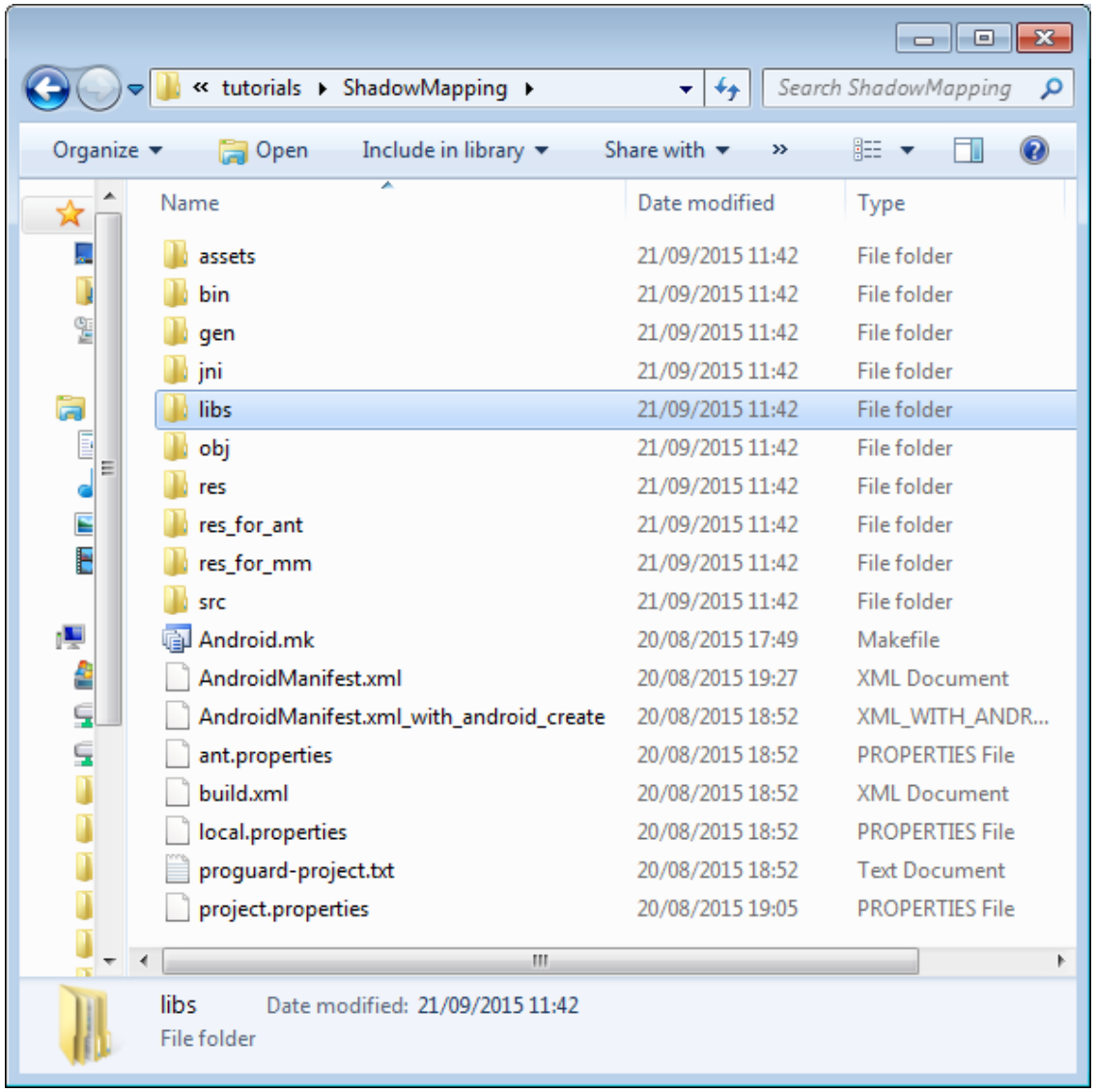
OPTIMIZATION OF EMBEDDED 3D GRAPHICS APPLICATIONS

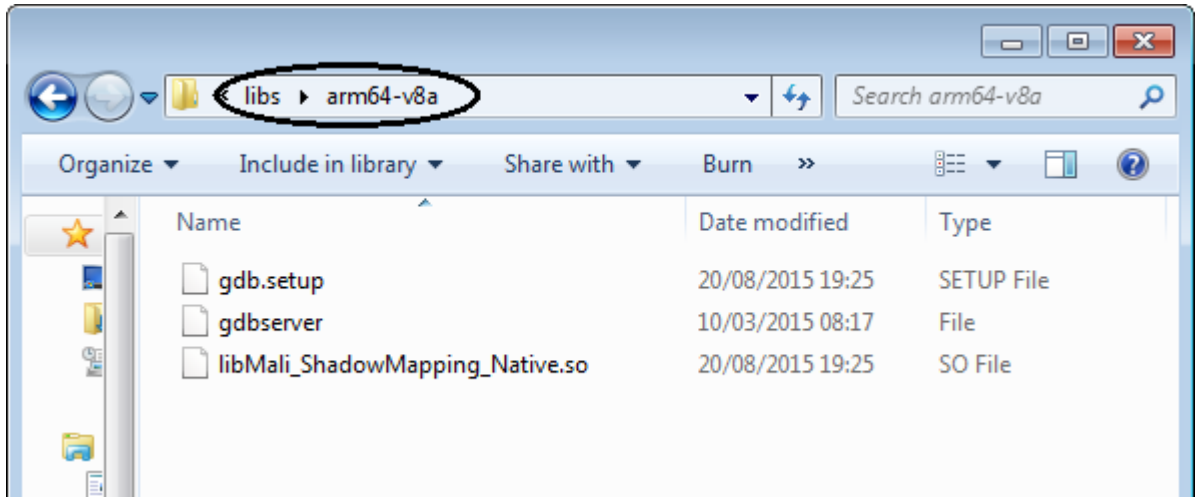
MALI GPU DEVELOPER TOOLS OVERVIEW SLIDES

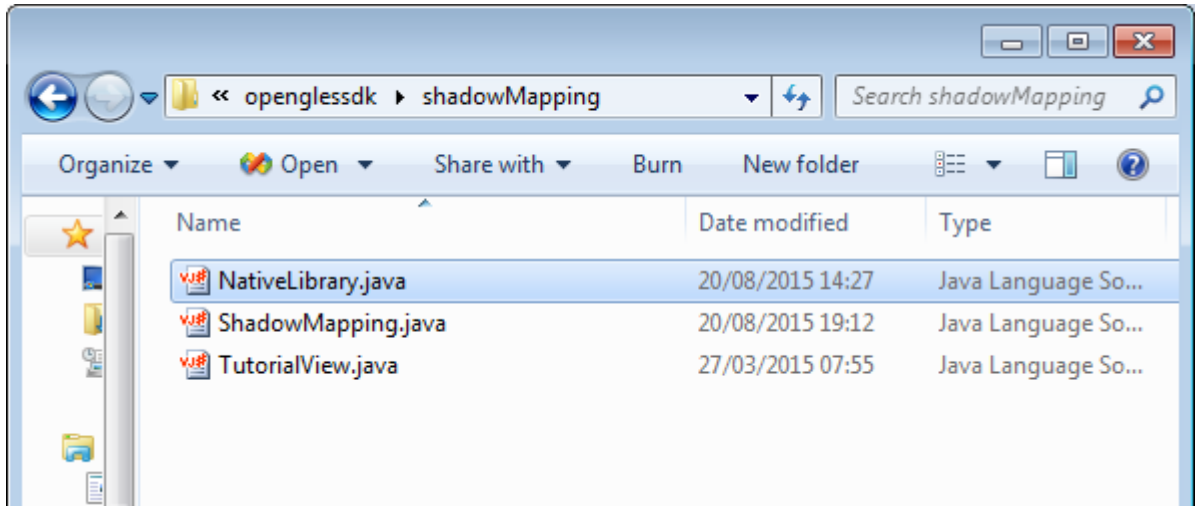
Support

Please submit your questions and issues to the Mali Developer Forums

Directory structure for a typical Android™ App project







ARM

Write Java™ code that links to native code through the JNI standard

ARM

Developing 64-bit Applications for Android™

Write Java™ code that links to native code through the JNI standard

```
package com.arm.malideveloper.openglessdk.shadowMapping;

public class NativeLibrary
{
    static
    {
        System.loadLibrary("Mali_ShadowMapping_Native");
    }
    public static native void init(int width, int height);
    public static native void uninit();
    public static native void step();
}
```

Developing 64-bit Applications for Android™

Write Java™ code that links to native code through the JNI standard

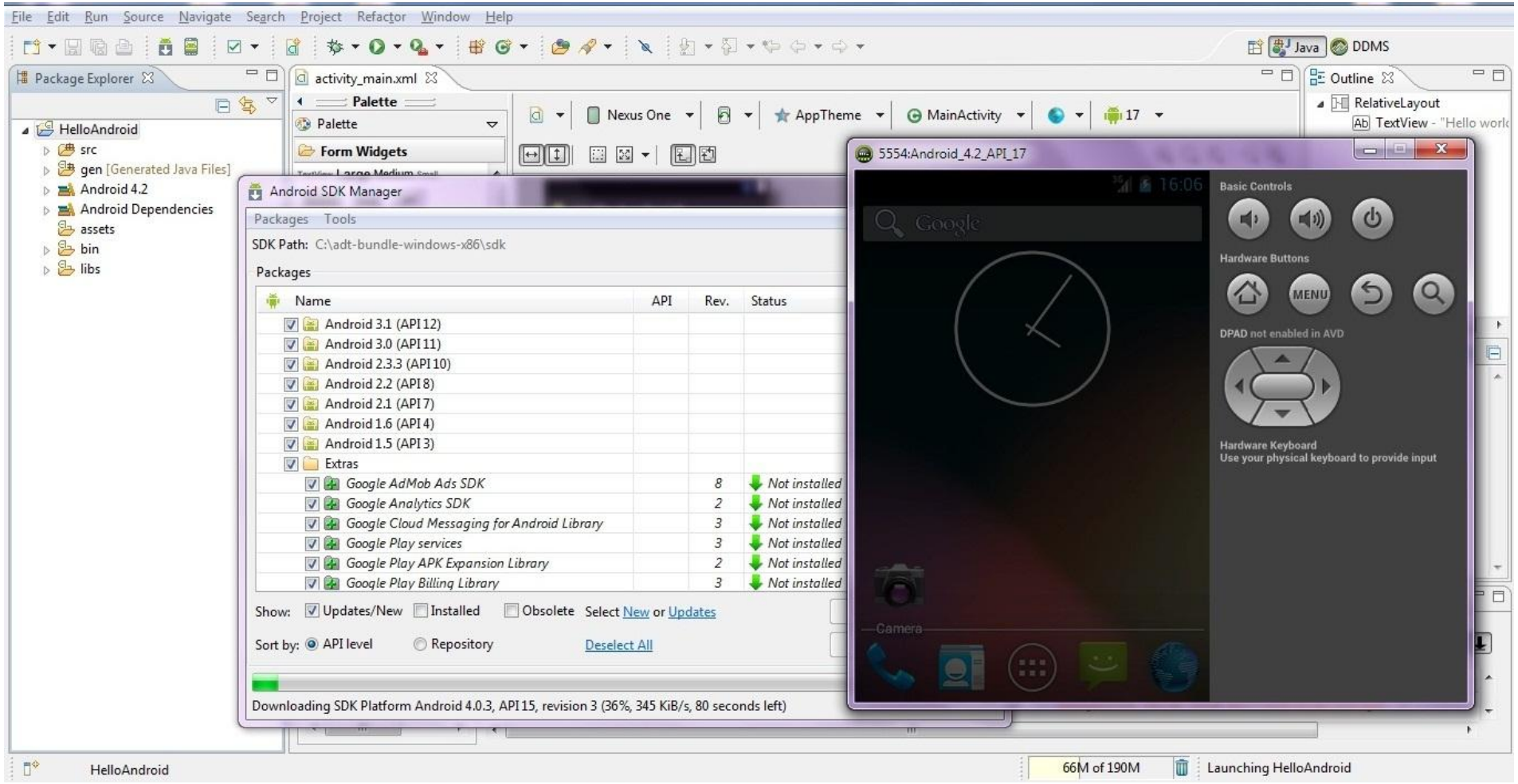
```
void uninit()
{
    /* Delete all created GL objects. */
    deleteObjects();
    /* Deallocate memory. */
    deallocateMemory();
}
extern "C"
{
    JNIEXPORT void JNICALL Java_com_arm_malideveloper_openglessdk_shadowMapping_NativeLibrary_init
        (JNIEnv * env, jobject obj, jint width, jint height);
    JNIEXPORT void JNICALL Java_com_arm_malideveloper_openglessdk_shadowMapping_NativeLibrary_step  (JNIEnv * env, jobject obj);
    JNIEXPORT void JNICALL Java_com_arm_malideveloper_openglessdk_shadowMapping_NativeLibrary_uninit(JNIEnv * env, jobject obj);
};

JNIEXPORT void JNICALL Java_com_arm_malideveloper_openglessdk_shadowMapping_NativeLibrary_init(
        JNIEnv * env, jobject obj, jint width, jint height)
{
    setupGraphics(width, height);
}
JNIEXPORT void JNICALL Java_com_arm_malideveloper_openglessdk_shadowMapping_NativeLibrary_uninit(
        JNIEnv * env, jobject obj)
{
    uninit();
}
JNIEXPORT void JNICALL Java_com_arm_malideveloper_openglessdk_shadowMapping_NativeLibrary_step(
        JNIEnv * env, jobject obj)
{
    renderFrame();
}
```


ARM

Compiling the native code and building the final package

Developing 64-bit Applications for Android™

The old way: *Apache Ant*™ and *Android Development Tools (ADT)* for *Eclipse*



ARM

Developing 64-bit Applications for Android™

The old way: *Apache Ant*™ and *Android Development Tools (ADT)* for *Eclipse*

- Commands to build your native code using NDK

```
cd jni/

<path-to-your-ndk-root>/ndk-build
```

Developing 64-bit Applications for Android™

The old way: *Apache Ant*™ and *Android Development Tools (ADT)* for *Eclipse*

- Don't forget that you do need correct *Android.mk* and *Application.mk*

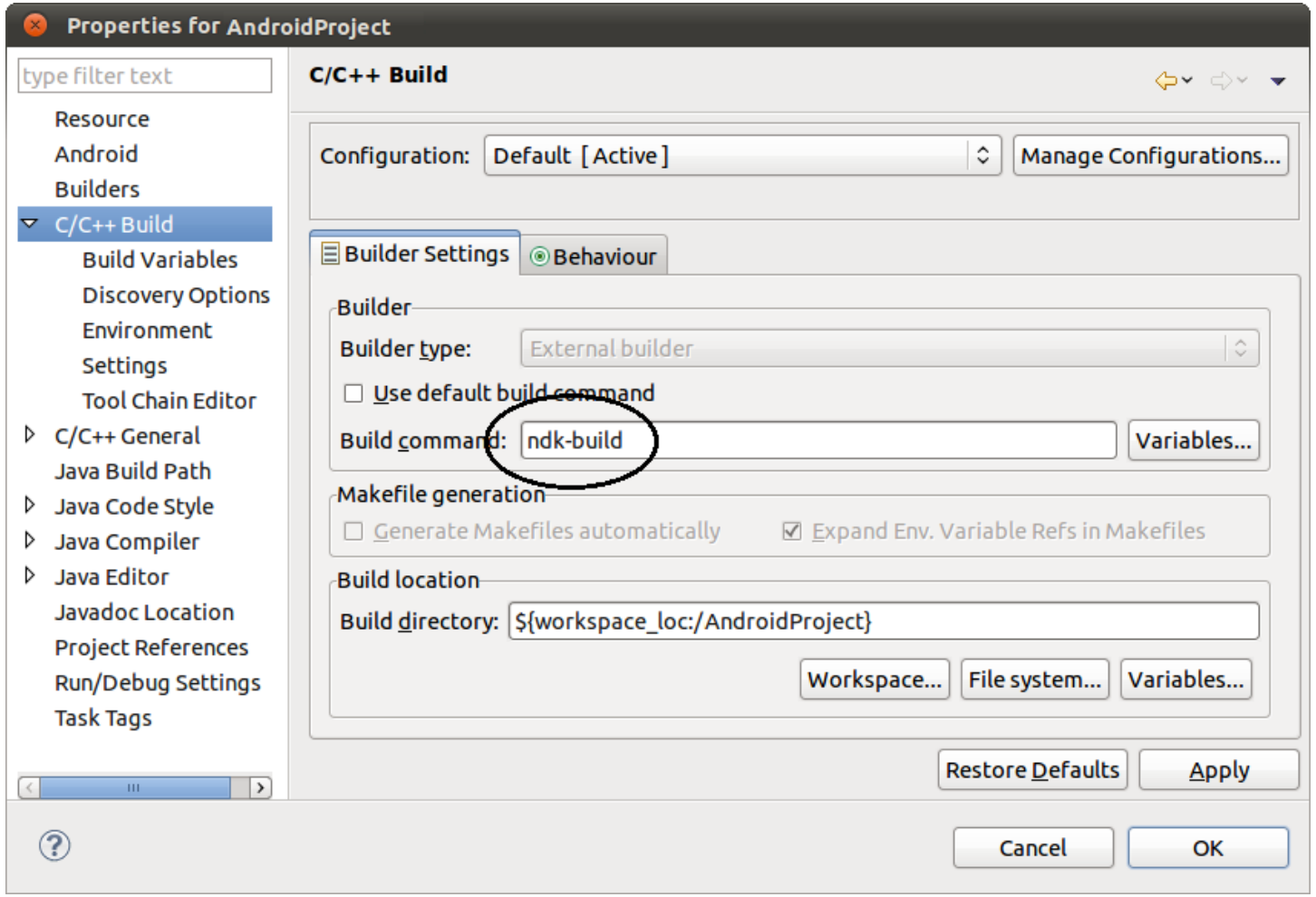
```
# Android.mk

LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE_TAGS := samples
LOCAL_SRC_FILES := Native.cpp.........
LOCAL_MODULE := libMali_ShadowMapping_Native
LOCAL_STATIC_LIBRARIES :=
LOCAL_C_INCLUDES += $(JNI_H_INCLUDE)
LOCAL_LDLIBS := -llog -lGLESv3
LOCAL_CFLAGS +=
LOCAL_DEX_PREOPT := false
include $(BUILD_SHARED_LIBRARY)
```

```
# Application.mk

APP_ABI := arm64-v8a

# Older API levels might miss some libraries
# such as "libGLESv3.so"
APP_PLATFORM := android-21
```


ARM

Developing 64-bit Applications for Android™

The old way: *Apache Ant*™ and *Android Development Tools (ADT)* for *Eclipse*



ARM

Developing 64-bit Applications for Android™

The old way: *Apache Ant*™ and *Android Development Tools (ADT)* for *Eclipse*

- Commands to prepare your project for building the final package using *Apache Ant* *(if you are not using Android ADT from within Eclipse)*

```
android update sdk -u

android list targets

android create project --target current --name Mali_ShadowMapping
--path . --activity ShadowMapping  --package
com.arm.malideveloper.openglessdk.shadowMapping
```

Developing 64-bit Applications for Android™

The old way: *Apache Ant*™ and *Android Development Tools (ADT)* for *Eclipse*

- The command to build the final package using *Apache Ant*
  (if you are not using Android ADT from within Eclipse)

```
ant debug
```

Developing 64-bit Applications for Android™

The new way: *Gradle*™ and *Android Studio*

- Android Studio is the future
- It takes a familiar approach
- Android SDK now includes Android Studio:
  https://developer.android.com/sdk/index.html

ARM

Android Studio Overview

Android Studio is the official IDE for Android application development, based on IntelliJ IDEA. On top of the capabilities you expect from IntelliJ, Android Studio offers:

- Flexible Gradle-based build system
- Build variants and multiple `apk` file generation
- Code templates to help you build common app features
- Rich layout editor with support for drag and drop theme editing
- `lint` tools to catch performance, usability, version compatibility, and other problems

Developing 64-bit Applications for Android™

The new way: *Gradle*™ and *Android Studio*

- NDK support is experimental
- It is only supported in the preview versions of Android Studio
  http://tools.android.com/tech-docs/android-ndk-preview

Android Tools Project Site

Android NDK Preview

We've just released another update to Android Studio 1.3's canary channel, Release Candidate 1.

This build contains a big new feature: Android NDK support, with support for editing, running and debugging C and C++ code! **Note** however that NDK su **preview** quality, and will remain in preview status for the upcoming final 1.3 release. There are many significant limitations (described below), but we hop will be useful for many Android NDK developers even in its current state!

Limitations

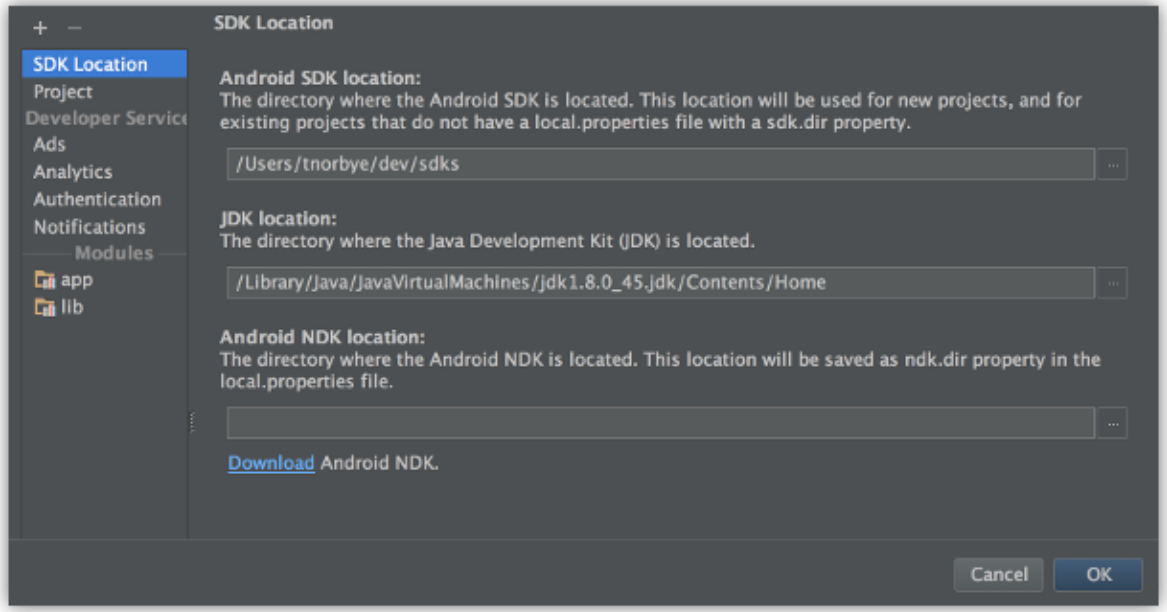
First, the Android NDK support **only** works with the new experimental Gradle plugin for Android (which in turn requires Gradle 2.5, released yesterday.)

While the new gradle plugin delivers some major performance improvements (and Android NDK build support), note that it also requires changes to the bu way your build is described in your build.gradle files.) Not only will you need to modify your build.gradle files (a process we hope to automate before the p from experimental to stable), but we anticipate making some additional incompatible changes along the way.

Second, note also that Android Studio has not yet been updated to fully handle the experimental plugin. This means that for example the Project Structur various quick fixes which automatically update the build data, do not work correctly. You'll need to edit your build.gradle files manually to configure your p example, the various templates which update the build files (such as New Module), have not yet been updated.

Historically, the Android NDK was distributed separately from the Android SDK, as a giant .zip file download, and installed in a separate location from where the Android SDK resides.

In this release we've tried to unify this; the NDK should now be installed under the SDK home in "ndk-bundle". We've updated the IDE built-in SDK Manager to let you install the NDK in the right place, and the Project Structure Dialog's SDK Location panel (where you can edit the locations), now also offers a quick link to install it directly; just click on the Download hyperlink shown below:



Note that the NDK is a **large** download (~1 GB!) so unless you have a really great network connection, prepare to be patient...

Speaking of large downloads, note also that the IDE update patches from previous 1.3 builds to this one are really large: not only does this build contain all the new code to support Android NDK development, but it also contains LLDB binaries (~200 MB uncompressed). This is just a temporary situation; we'll move these into the NDK.

How to get started

Since the Android NDK support depends on an experimental plugin, with an incompatible build file syntax which we know will change over the next few months, we have not added a simple way to create Android NDK projects. Instead, to explore the Android NDK support, either

1. Follow the Experimental Gradle Plugin document which describes how to modify your project to use the current version of the experimental plugin, or
2. Check out some of the existing Android NDK samples which have been updated to work with the new Gradle plugin.

To check out the samples, invoke VCS > Checkout from Version Control > Git, and in the resulting dialog point to https://github.com/googlesamples/android-ndk.git . Pick a location to check out the samples into. **NOTE:** At the end of the import the IDE will ask you whether you want to create a project from the checked out sources. **Answer no**. The android-ndk git project contains a large number of separate projects. Instead, once the import is done, go to File > Open..., and navigate to (for example) the Teapot folder and select the file `build.gradle` in that folder. The project will now be opened and synced. This will take a while the first time as the new Gradle

Android Tools Project Site

Download

You can download the official bits from http://developer.android.com/sdk/index.html.
For Android Studio preview channels and download links, see the Android Studio Downloads page.

If you want to build it yourself, follow the instructions in the Build Overview document. Recent user-visible changes to the trunk are listed in the Rece

Subpages (6): ADT 14 Preview ADT 17 Preview ADT 20 Preview ADT 21 Preview Android Studio Downloads Studio 0.2.12

adt-issue-82393-v4.zip (14755k)	Tor Norbye, Dec 15, 2014, 7:56 AM
adt-issue-82393.zip (14744k)	Tor Norbye, Dec 12, 2014, 5:16 PM
sdk-repo-darwin-platform-tools-2219242.zip (2674k)	Siva Velusamy, Sep 3, 2015, 5:46 PM
sdk-repo-linux-platform-tools-2219198.zip (2798k)	Siva Velusamy, Sep 3, 2015, 5:47 PM

Android Tools Project Site

Projects Overview
Screenshots
Release Status
Roadmap
Download
Preview Channel
Recent Changes
Technical docs
New Build System

Known Issues
Tips

Build Overview
Contributing
Feedback

Download >

Android Studio Downloads

We offer preview channels for Android Studio:

- Canary
- Dev
- Beta
- Stable

For more information on how to get these updates, see the Preview Channel page.

Subpages (5): Android Studio Beta Channel Android Studio Builds Android Studio Canary Channel Android Studio Dev Chann

Comments

You do not have permission to add comments.

Android Tools Project Site

Android Studio Canary Channel

The Canary Channel for Android Studio delivers the bleeding edge updates on a roughly weekly b

For more information, see

- Latest Build: 1.4 RC 1 (September 18th, 2015)
- 1.4 RC 1 (September 18th, 2015)
- 1.4 Beta 4 (September 14th, 2015)
- 1.4 Beta 3 (September 9th, 2015)
- 1.4 Beta 2 (August 31st, 2015)
- 1.4 Beta (August 28th, 2015)

Android Tools Project Site

Latest Android Studio Canary Build: 1.4 RC 1

September 18th, 2015: For information on what's new in 1.4 RC 1, see the release announcement. For additional information about An

Installation

The release is available in the canary channel, so you can switch to that channel (if necessary) and check for updates via Help > Che patch rather than download a full IDE image. Note however that if you do this, you will replace your existing 1.3 installation, which you separate settings and cache directories.

Windows: https://dl.google.com/dl/android/studio/ide-zips/1.4.0.7/android-studio-ide-141.2262011-windows.zip (353 MB)
Mac: https://dl.google.com/dl/android/studio/ide-zips/1.4.0.7/android-studio-ide-141.2262011-mac.zip (354 MB)
Linux: https://dl.google.com/dl/android/studio/ide-zips/1.4.0.7/android-studio-ide-141.2262011-linux.zip (360 MB)

SHA-1 Checksums:

```
abc3144bae19459a87fe42056e13dc9508e08266 android-studio-ide-141.2262011-windows.zip
8cd5a56139abe2661bd1315502449014e40d0c59 android-studio-ide-141.2262011-mac.zip
```

Android Tools Project Site

Recent Changes >

Android Studio 1.4 RC 1 Available

posted Sep 18, 2015, 9:35 AM by Tor Norbye **[updated Sep 18, 2015, 10:57 AM]**

We've just released Android Studio 1.4 RC 1 to the canary channel.

We've also promoted 1.4 Beta 4 to the beta channel, since that build has been in testing for a few days now and there are no big problem. (However, if you're on Linux and using directly to RC 1 where there are important fixes for that configuration.)

RC 1 contains bug fixes in a number areas:

- Fixes for high-density displays on Linux (and in general, we have more information on how to configure high density displays for Windows and Linux in this document)
- Many more fixes to the templates (activities, etc), visual refresh, and removed misc warnings
- Vector icons displayed properly in the editor margin
- Layout editor bug with duplicate id's in RelativeLayouts
- Auto completion of upper-cased permission names in manifest files
- Inferring nullity in multi-module project
- Lint warns about missing platform-tools 23.0.1 (prevents properly API checking Android M calls) or using buildTools pre-23.0.1 (to prevent crashes on some devices for R
- Fix build errors with vector image generation on OSX when using Java 6 to run the IDE and Java 7 to run the IDE ("Could not find class: apple.awt.CGraphicsEnvironmer
- Fix git module configuration bug
- Misc theme editor bugs
- Misc crash reporter bugs

Installation

If you are already using a 1.4 Preview/Beta build, you can use the builtin Update mechanism to upgrade directly to 1.4 RC 1.

Developing 64-bit Applications for Android™

The new way: *Gradle*™ and *Android Studio*

- Download and installed Android Studio
- Make sure you follow the guidelines or the release notes for using the NDK
- Import your current project files into Android Studio
- Modify Gradle scripts to suit your project and your particular version of Android Studio

DEMO

Developing 64-bit Applications for Android™

Building as part of the *Android Open Source Project (AOSP)* source tree

- Useful for Systems Engineers or anyone working on the Android Open Source Project
- Drop your sample code into the AOSP source tree
  e.g. `/work/android/android-5.1.1_r1/development/samples`
- With *Android.mk* and *Application.mk* in your App's project and *jni/* directories do
  `mm -j8`
- This will build a "system" App which will only support the default architecture

ARM

Questions?

ARM

Thank You