Android NDK Native APIs

The Android NDK provides a set of native headers and shared library files that has gradually increased with successive releases of new Android API levels. This page explains these headers and files, and maps them to specific Android API levels.

Using native APIs

There are two basic steps to enable your app to use the libraries that the NDK provides:

1. Include in your code the headers associated with the libraries you wish to use.
2. Tell the build system that your native module needs to link against the libraries at load time.

- If you are using ndk-build: Add the native library to your LOCAL_LDLIBS variable in your Android.mkfile. For example, to link against /system/lib/libfoo.so, add the following line:

3. LOCAL_LDLIBS := -lfoo

To list multiple libraries, use a space as a delimiter. For more information about using theLOCAL_LDLIBS variable, see Android.mk.

- If you are using CMake: Follow the instructions in Add C and C++ Code to Your Project.

For all API levels, the build system automatically links the standard C and C++ libraries. You do not need to explicitly include them when setting LOCAL_LDLIBS.

The NDK often provides new headers and libraries for new Android releases. For a list of the native APIs introduced by Android release version, see Table 1 below. These files reside in your NDK installation root, under sysroot/usr/include.

The following table shows the correspondence between NDK-supported API levels and Android platform releases. For more information about Android API levels, see What is API Level?

Table 1. Summary of key native API support introduced by Android version.

Native API highlights

Android API level 3

C library

The standard C library headers are available through their usual names, such as <stdlib.h> and <stdio.h>.

Note that on Android, unlike Linux, there are no separate pthread and rt libraries. That functionality is included in libc. The math library remains separate in libm, but is automatically added by the compiler.

Dynamic linker library

You can access the Android dynamic linker's dlopen(3) and dlsym(3) functionality. You must also link against libdl. For example:

```
LOCAL_LDLIBS := -ldl
```

C++ library

C++17 support is available. For more information on C++ library support, see C++ Library Support.

Android-specific log support

<android/log.h> contains various functions that an app can use to send log messages to logcat from native code. For more information about these definitions, see the logging documentation.

Typically you should write your own wrapper macros to access this functionality. If you wish to log, link against liblog. For example:

```
LOCAL_LDLIBS := -llog
```

ZLib compression library

You can use the Zlib compression library by including zlib.h and zconf.h. You must also link your native module against /system/lib/libz.so by including the following line in your Android.mk file:

```
LOCAL_LDLIBS := -lz
```

Android API level 4

The NDK provides the following APIs for developing native code that runs on Android 1.6 system images and above.

OpenGL ES 1.x Library

The standard OpenGL ES headers <GLES/gl.h> and <GLES/glext.h> contain the declarations necessary for performing OpenGL ES 1.x rendering calls from native code.

To use these headers, link your native module to /system/lib/libGLESv1_CM.so by including the following line in your Android.mk file:

```
LOCAL_LDLIBS := -lGLESv1_CM
```

All Android-based devices support OpenGL ES 1.0, because Android provides an Open GL 1.0-capable software renderer that can be used on devices without GPUs.

Only Android devices that have the necessary GPU fully support OpenGL ES 1.1. An app can query the OpenGL ES version string and extension string to determine whether the current device supports the features it needs. For information on how to perform this query, see the description of glGetString() in the OpenGL specification.

Additionally, you must put a <uses-feature> tag in your manifest file to indicate the version of OpenGL ES that your application requires.

The EGL APIs are only available starting from API level 9. You can, however, use the VM to perform some of the operations that you would get from those APIS. These operations include surface creation and flipping. For an example of how to use GLSurfaceView, see Introducing GLSurfaceView.

The san-angeles sample application provides an example of how to perform these operations, rendering each frame in native code. This sample is a small Android port of the excellent San Angeles Observation demo program.

Android API level 5

The NDK provides the following APIs for developing native code that runs on Android 2.0 system images and above.

OpenGL ES 2.0 library:

The standard OpenGL ES 2.0 headers <GLES2/gl2.h> and <GLES2/gl2ext.h> contain the declarations needed for performing OpenGL ES 2.0 rendering calls from native code. These rendering calls provide the ability to use the GLSL language to define and use vertex and fragment shaders.

To use OpenGL ES 2.0, link your native module to /system/lib/libGLESv2.so by including the following line in your Android.mk file:

```
LOCAL_LDLIBS := -lGLESv2
```

Not all devices support OpenGL ES 2.0. An app can query the OpenGL ES version string and extension string to determine whether the current device supports the features it needs. For information on how to perform this query, see the description of glGetString() in the OpenGL specification.

Additionally, you must put a <uses-feature> tag in your manifest file to indicate which version of OpenGL ES your application requires. For more information about the OpenGL ES settings for <uses-feature>, see OpenGL ES.

The hello-gl2 sample application provies a basic example of how to use OpenGL ES 2.0 with the NDK.

The EGL APIs are only available starting from API level 9. You can, however, use the VM to perform some of the operations that you would get from those APIs. These operations include surface creation and flipping. For an example of how to use GLSurfaceView, see Introducing GLSurfaceView.

Android API level 8

The NDK provides the following APIs for developing native code that runs on Android 2.2 system images and above.jnigraphics

The jnigraphics library exposes a C-based interface that allows native code to reliably access the pixel buffers of Java bitmap objects. The workflow for using jnigraphics is as follows:

1. Use AndroidBitmap_getInfo() to retrieve information from JNI, such as width and height, about a given bitmap handle.
2. Use AndroidBitmap_lockPixels() to lock the pixel buffer and retrieve a pointer to it. Doing so ensures that the pixels do not move until the app calls AndroidBitmap_unlockPixels().
3. In native code, modify the pixel buffer as appropriate for its pixel format, width, and other characteristics.
4. Call AndroidBitmap_unlockPixels() to unlock the buffer.

To use jnigraphics, include the <bitmap.h> header in your source code, and link against jnigraphics by including the following line in your Android.mk file:

```
LOCAL_LDLIBS += -ljnigraphics
```

Additional details about this feature are in the comments of the <android/bitmap.h> file.

Android API level 9

The NDK provides the following APIs for developing native code that runs on Android 2.3 system images and above.

EGL

EGL provides a native platform interface for allocating and managing OpenGLES surfaces. For more information about its features, see EGL Native Platform Interface.

EGL allows you to perform the following operations from native code:

- List supported EGL configurations.
- Allocate and release OpenGLES surfaces.
- Swap or flip surfaces.

The following headers provide EGL functionality:

- <EGL/egl.h>: the main EGL API definitions.

- <EGL/eglext.h>: EGL extension-related definitions.

To link against the system's EGL library, add the following line to your Android.mk file:

```
LOCAL_LDLIBS += -Legl
```

OpenSL ES

Android native audio handling is based on the Khronos Group OpenSL ES 1.0.1 API.

The standard OpenSL ES headers <SLES/OpenSLES.h> and <SLES/OpenSLES_Platform.h> contain the declarations necessary for performing audio input and output from the native side of Android. The NDK distribution of the OpenSL ES also provides Android-specific extensions. For information about these extensions, see the comments in <SLES/OpenSLES_Android.h> and <SLES/OpenSLES_AndroidConfiguration.h>.

The system library libOpenSLES.so implements the public native audio functions. Link against it by adding the following line to your Android.mk file:

```
LOCAL_LDLIBS += -lOpenSLES
```