2017 Android of WMNTAA

Native Development Kit (NDK)

NDK

- Overview
- Set up
- Write Native Functions

Overview

- The Native Development Kit (NDK) is a set of tools that allows you to use C and C++ code with Android, and provides platform libraries you can use to manage native activities and access physical device components, such as sensors and touch input.
- The NDK can be useful for cases in which you need to do one or more of the following:
  - Squeeze extra performance out of a device to achieve low latency or run computationally intensive applications, such as games or physics simulations.
  - Reuse your own or other developers' C or C++ libraries.

Set up

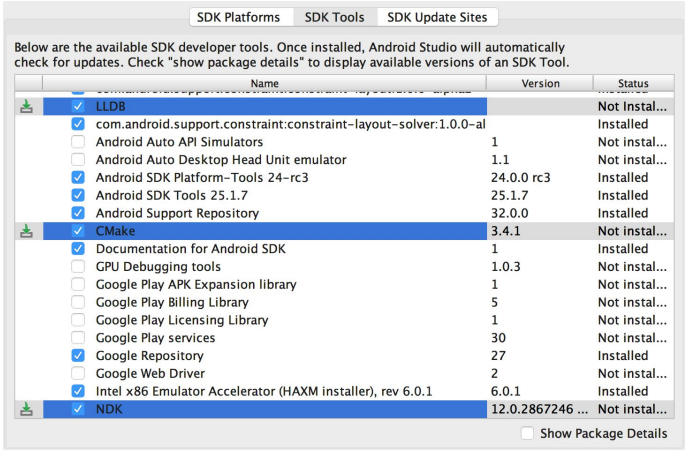
- Download the NDK and Tools
- Hello World
- Add C/C++ Code to an Existing Project

Download the NDK and Tools

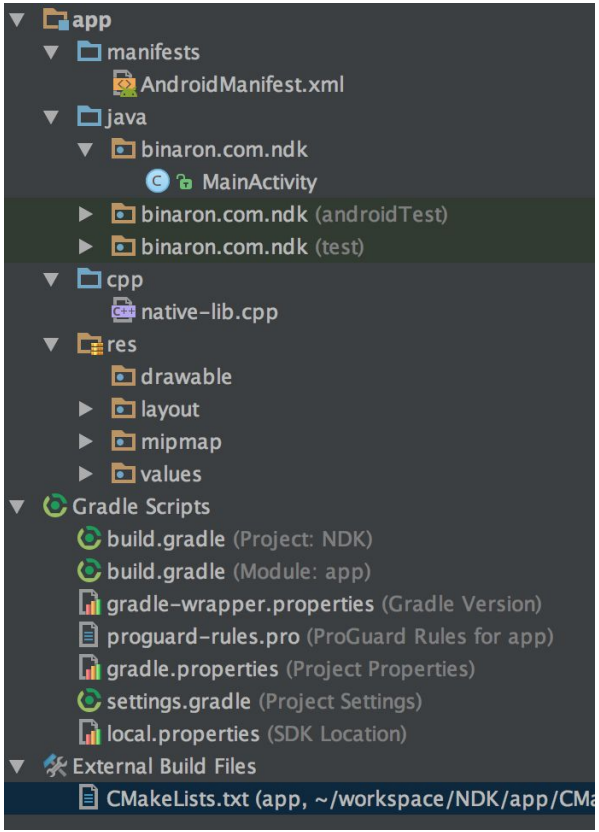
- To compile and debug native code for your app, you need the following components:
  - The Android Native Development Kit (NDK): a set of tools that allows you to use C and C++ code with Android.
  - CMake: an external build tool that works alongside Gradle to build your native library. You do not need this component if you only plan to use ndk-build.
  - LLDB: the debugger Android Studio uses to debug native code.

Download the NDK and Tools

- You can install these components using the SDK Manager:
  - From an open project, select Tools > Android > SDK Manager from the main menu.
  - Click the SDK Tools tab.
  - Check the boxes next to LLDB, CMake, and NDK
  - Click Apply, and then click OK in the next dialog.
  - When the installation is complete, click Finish, and then click OK.



NDK Hello World




```java
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

    // Example of a call to a native method
    TextView tv = (TextView) findViewById(R.id.sample_text);
    tv.setText(stringFromJNI());
    }

    /**
     * A native method that is implemented by the 'native-lib' native library,
     * which is packaged with this application.
     */
    public native String stringFromJNI();

    // Used to load the 'native-lib' library on application startup.
    static {
        System.loadLibrary("native-lib");
    }
}
```

Add C/C++ Code to an Existing Project

- Create new native source files
- Create a CMake build script
- Link Gradle to your native library

Reference : Add C and C++ Code to Your Project

Create new native source files

- To create a cpp/ directory with new native source files in the main sourceset of your app module, proceed as follows:
  - Open the Project pane from the left side of the IDE and select the Project view from the drop-down menu.
  - Navigate to your-module > src, right-click on the main directory, and select **New > Directory.**
  - Enter a name for the directory (such as cpp) and click **OK**.
  - Right-click on the directory you just created and select **New > C/C++ Source File**.
  - Enter a name for your source file, such as native-lib.
  - From the **Type** drop-down menu, select the file extension for your source file, such as .cpp.
  - You can add other file types to the drop-down menu, such as **.cxx** or **.hxx**, by clicking **Edit File Types** . In the **C/C++** dialog box that pops up, select another file extension from the **Source Extension** and **Header Extension** drop-down menus and click **OK**.
  - If you also want to create a header file, check the **Create an associated header** checkbox.
  - Click **OK**.

Create a CMake build script

- To create a plain text file that you can use as your CMake build script, proceed as follows:
  - Open the **Project** pane from the left side of the IDE and select the **Project** view from the drop-down menu.
  - Right-click on the root directory of your-module and select **New > File**.
  - Enter "CMakeLists.txt" as the filename and click **OK**.

Create a CMake build script

- Configure your build script by adding CMake commands

```
# Sets the minimum version of CMake required to build your native library.
# This ensures that a certain set of CMake features is available to
# your build.

cmake_minimum_required(VERSION 3.4.1)

# Specifies a library name, specifies whether the library is STATIC or
# SHARED, and provides relative paths to the source code. You can
# define multiple libraries by adding multiple add.library() commands,
# and CMake builds them for you. When you build your app, Gradle
# automatically packages shared libraries with your APK.

add_library( # Specifies the name of the library.
             native-lib

             # Sets the library as a shared library.
             SHARED

             # Provides a relative path to your source file(s).
             src/main/cpp/native-lib.cpp )
```

Create a CMake build script

- Add command to your CMake build script and specify the path to your headers:

```
add_library(...)

# Specifies a path to native header files.
include_directories(src/main/cpp/include/)
```

Link Gradle to your native library

- To manually configure Gradle to link to your native library, you need to add the externalNativeBuild block to your module-level build.gradle file and configure it with either the cmake or ndkBuild block:

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

  // Encapsulates your external native build configurations.
  externalNativeBuild {

    // Encapsulates your CMake build configurations.
    cmake {

      // Provides a relative path to your CMake build script.
      path "CMakeLists.txt"
    }
  }
}
```

Write Native Functions

- Load Libraries
- Declaration
- Implement

Load Libraries

- The convention CMake uses to name the file of your library is as follows:

```
liblibrary-name.so
```

- For example, if you specify "native-lib" as the name of your shared library in the build script, CMake creates a file named libnative-lib.so. However, when loading this library in your Java code, use the name you specified in the CMake build script:

```
static {
    System.loadLibrary("native-lib");
}
```

Declaration

- You need to declare the native function with native prefix declaration in the class where you call that function.

```
/**
 * A native method that is implemented by the 'native-lib' native library,
 * which is packaged with this application.
 */
public native String stringFromJNI();
```

Implement

- The implementation is same as JNI. You just need to follow the NDK naming rule : Java_package_of_the_class_class_name_function_name.

```
#include <jni.h>
#include <string>

extern "C"
JNIEXPORT jstring JNICALL
Java_binaron_com_ndk_MainActivity_stringFromJNI(
        JNIEnv* env,
        jobject /* this */) {
    std::string hello = "Hello from C++";
    return env->NewStringUTF(hello.c_str());
}
```

Challenge

- Write a native function : String func(String s).
- Use “校務資訊系統” as the input.
- Combine “記得要到”, input string, and “填教學問卷調查喔” into a string in the native function.
- The native function returns the combined string as the output.