A step-by-step guide to setup developing and debugging environment in Eclipse for a Native Android Application.

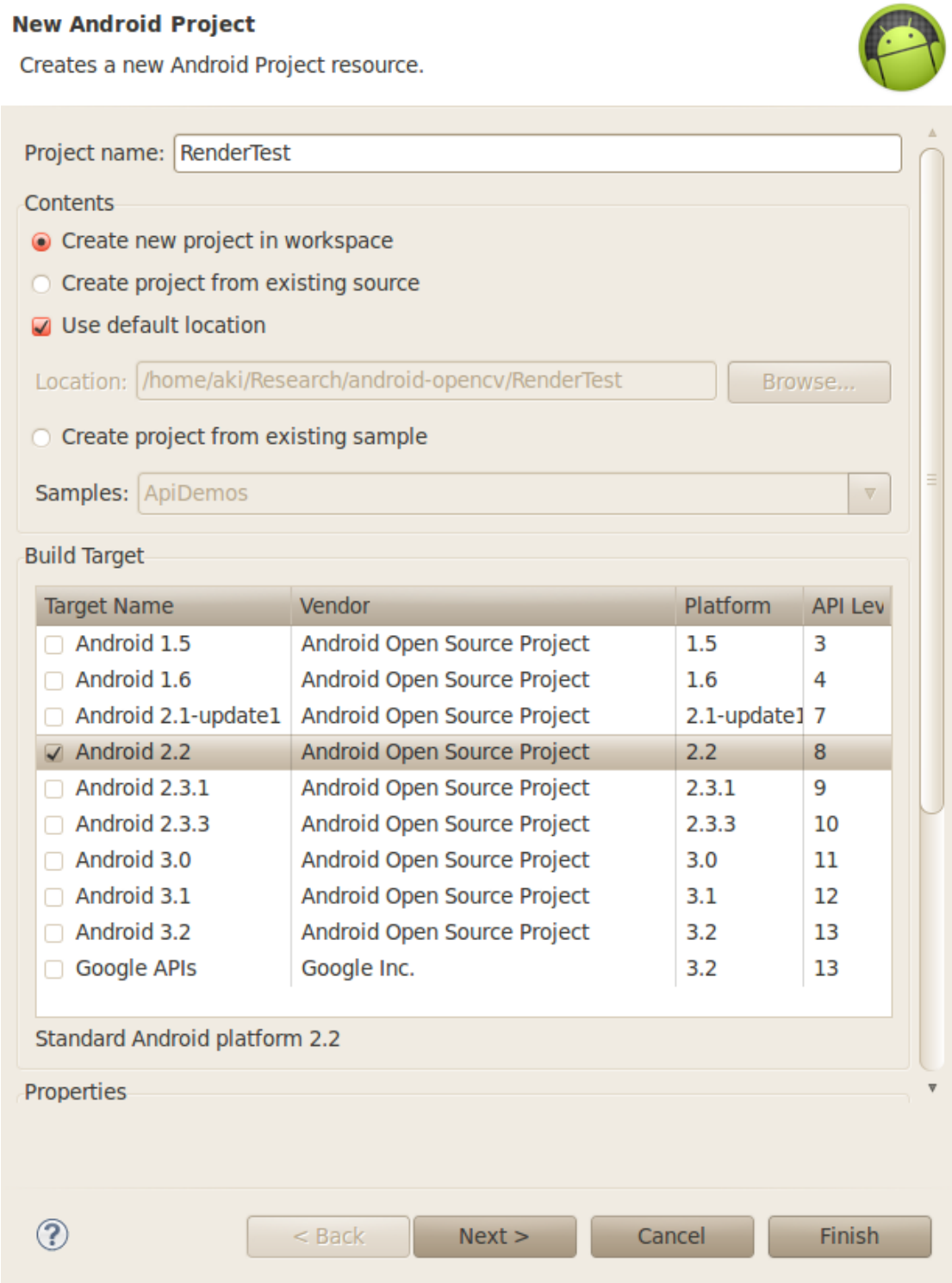
By Yu Lu (Referenced from two guides by MartinH)

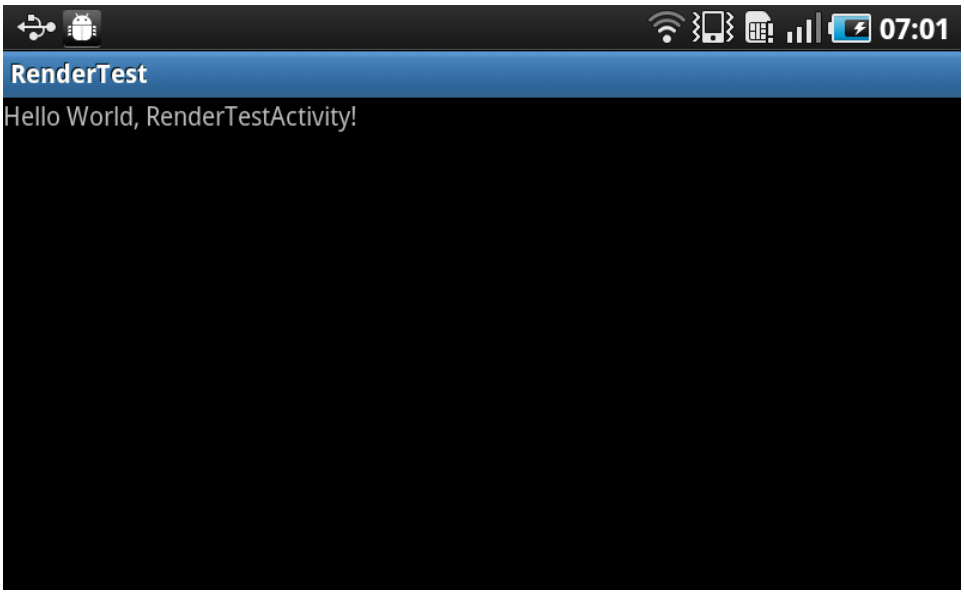
Jan, 2012

Development_Setting

Step I: Create an Android Project

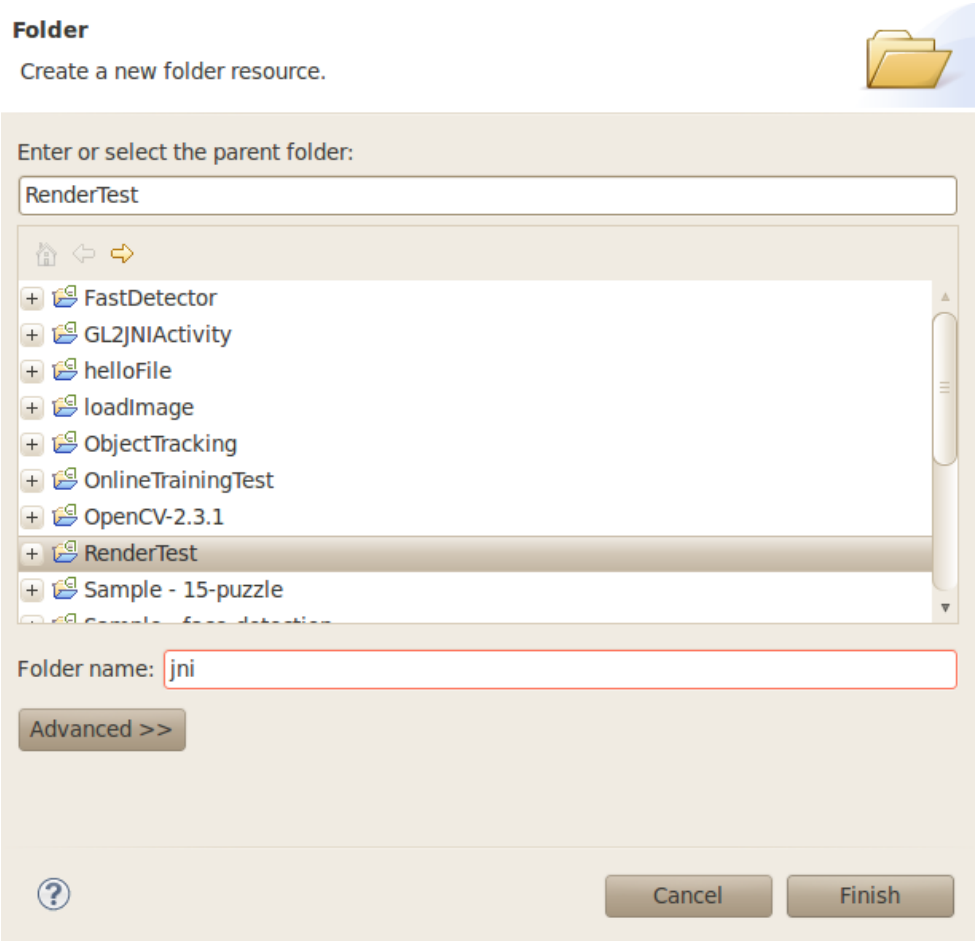
Here package name is given as “com.research.phoneARt” based on my own research project. The created project now is already OK to be running with a “Hello...” message shown.





Step II: Create the jni/ folder and include the Android.mk and Application.mk makefile

In file manager create **jni/** directory in the project directory and place the C/C++ sources file here. Also put here **Android.mk** file which is a makefile that tells Android build-system how to build the files, and the **Application.mk** file is optional, but in case of project using OpenCV, when STL and exceptions are used in C++, it also should be written.



An example of the ***Android.mk*** (include the native OpenCV library) is shown below

Android.mk | Android.mk

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

include ../includeOpenCV.mk
ifeq ("$(wildcard $(OPENCV_MK_PATH))",)
    #try to load OpenCV.mk from default install location
    include $(TOOLCHAIN_PREBUILT_ROOT)/user/share/OpenCV/OpenCV.mk
else
    include $(OPENCV_MK_PATH)
endif

LOCAL_MODULE    := native_sample
LOCAL_SRC_FILES := jni_part.cpp
LOCAL_CFLAGS    := -g
LOCAL_LDLIBS +=  -llog -ldl

include $(BUILD_SHARED_LIBRARY)
```

If the application utilize both native OpenCV and its Java API you need to put the following line before including OpenCV.mk to avoid conflict between C++ and Java builders:

```
OPENCV_CAMERA_MODULES:=off
```

For example:

Android.mk | Android.mk

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

OPENCV_CAMERA_MODULES:=off
include ../includeOpenCV.mk
ifeq ("$(wildcard $(OPENCV_MK_PATH))",)
    #try to load OpenCV.mk from default install location
    include $(TOOLCHAIN_PREBUILT_ROOT)/user/share/OpenCV/OpenCV.mk
else
    include $(OPENCV_MK_PATH)
endif

LOCAL_MODULE    := mixed_sample
LOCAL_SRC_FILES := jni_part.cpp
LOCAL_LDLIBS +=  -llog -ldl

include $(BUILD_SHARED_LIBRARY)
```

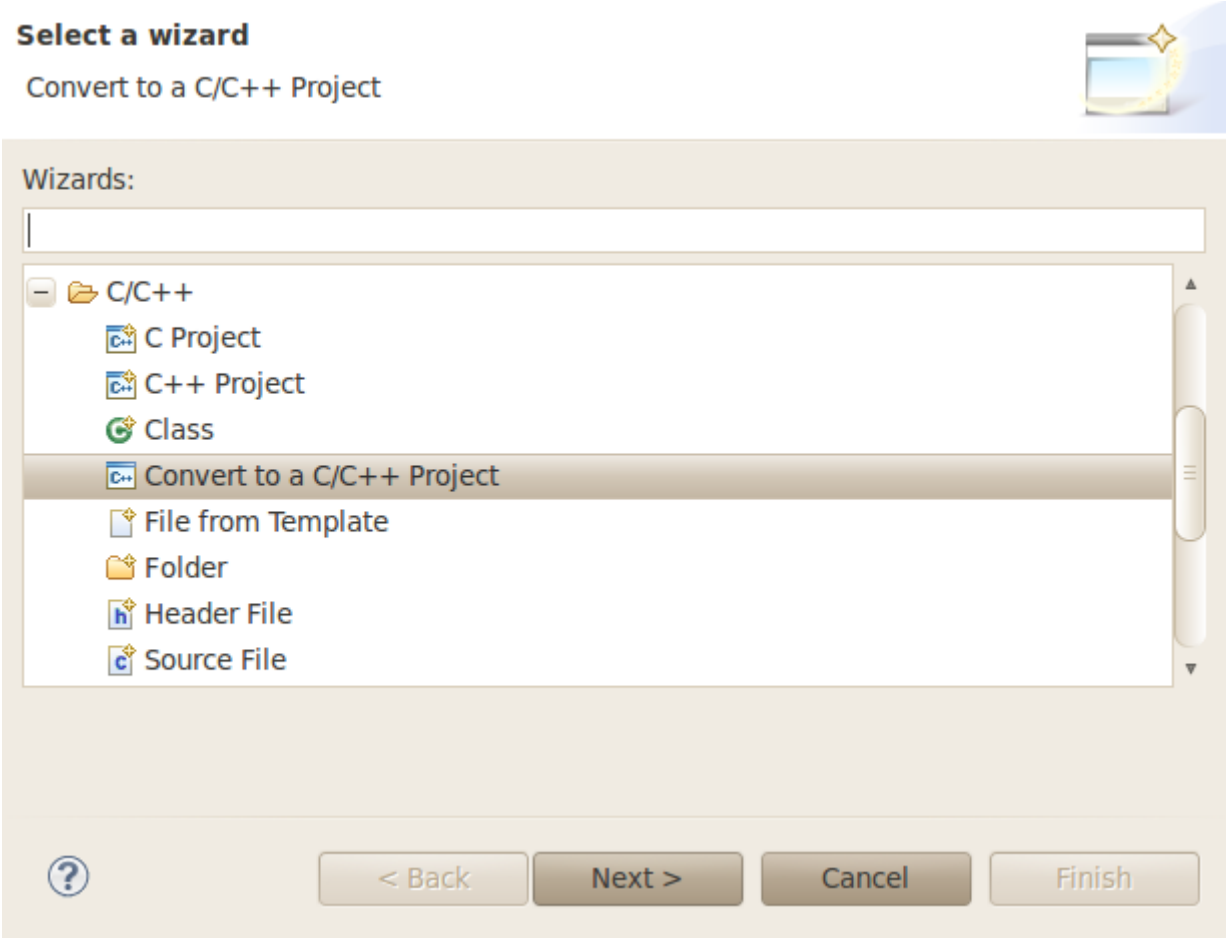
Application.mk example*:*

Application.mk

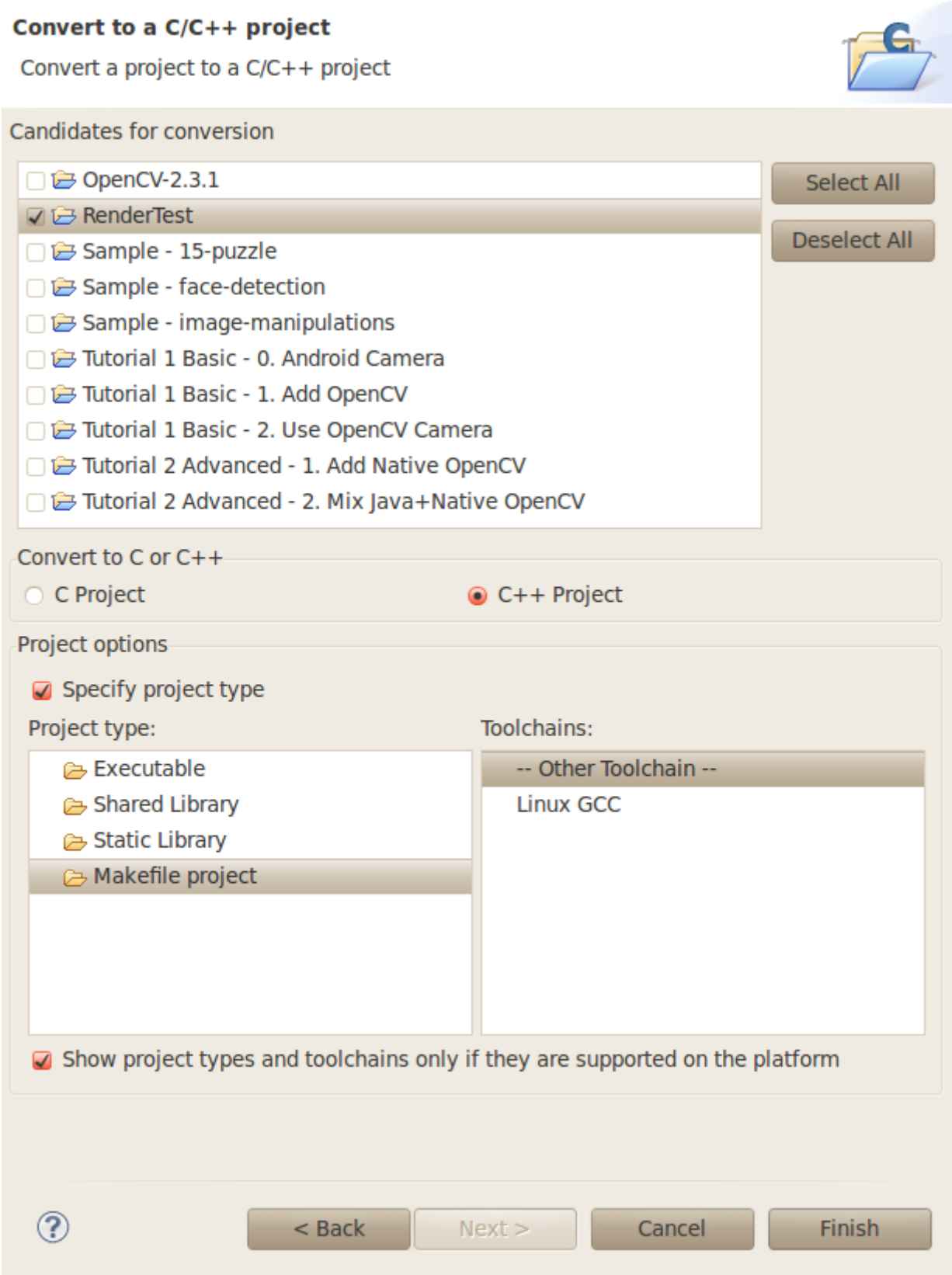
```
APP_STL := gnustl_static
APP_CPPFLAGS := -frtti -fexceptions
APP_ABI := armeabi-v7a
APP_OPTIM := debug
```

Step III: Put the native codes file in jni/ directory and convert the project into a C/C++ project to avoid sencountering syntax errors

choose ***File->New->Other*** and select ***Convert to a C/C++ Project***:



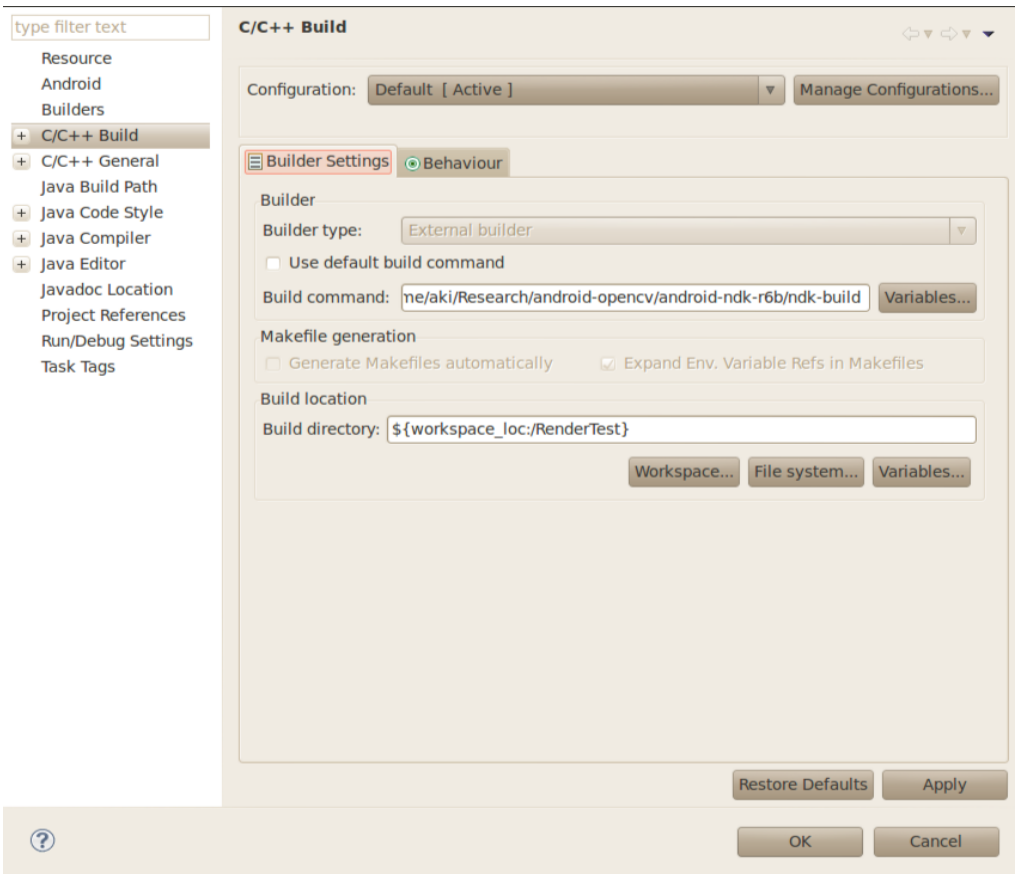
Click Next, then choose your project and choose ***Makefile project*** and -***Other Toolchain-.*** Click Finish:



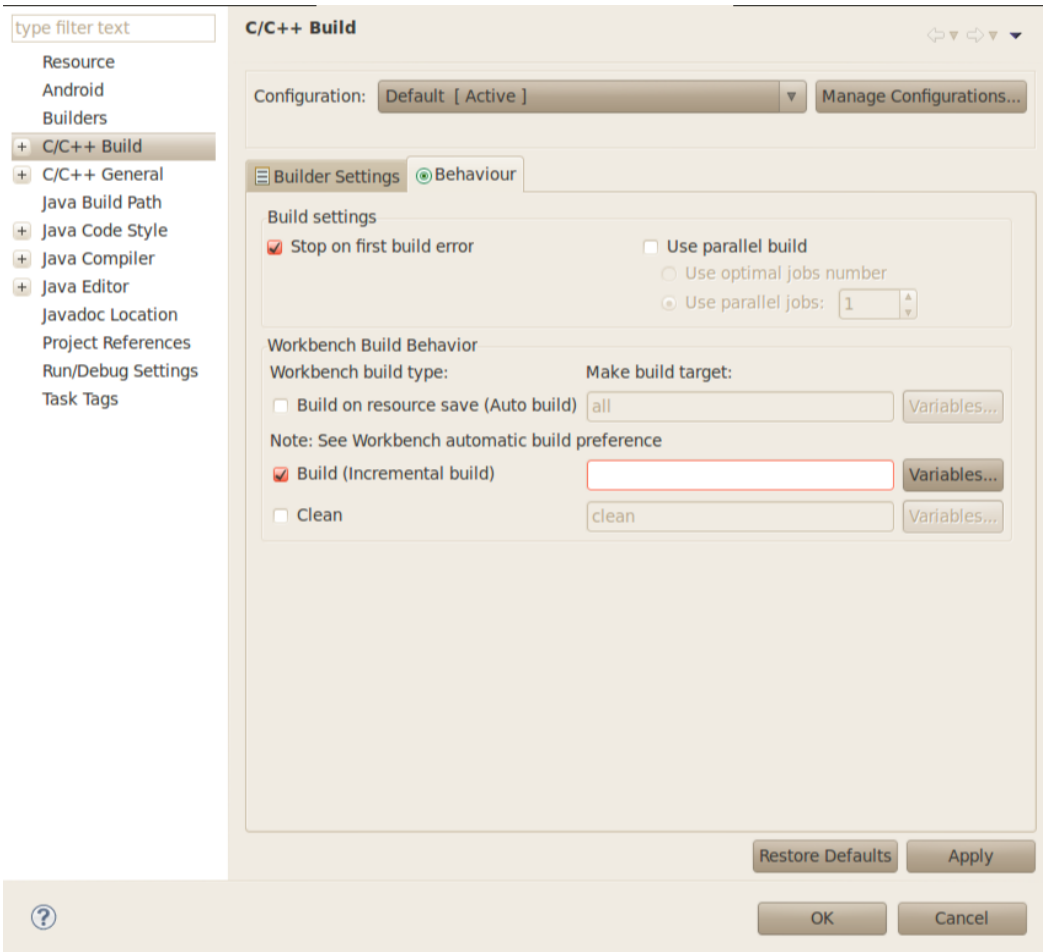
Step IV: Configure ndk-build as a build command

Right click the name of the project and select the ***Properties*** button at the bottom. In ***Builder Settings*** change the ***Build Command*** to ndk-build (include the full path)

My path of ndk is : /home/aki/Research/android-opencv/android-ndk-r6b/ndk-build



In Behaviour tab, configure as shown below:



Step V: Library and function decleration in Java and Native

Declare in **Android.mk** file as

```
LOCAL_MODULE    := librender_test
```

Load in JAVA

```
static {
        System.loadLibrary("render_test");
}
```

Declare native function in C/C++ as

```
JNIEXPORT void JNICALL Java_com_research_phoneARt_RenderLib_step(JNIEnv *
env, jobject obj)
{
        renderFrame();
}
```

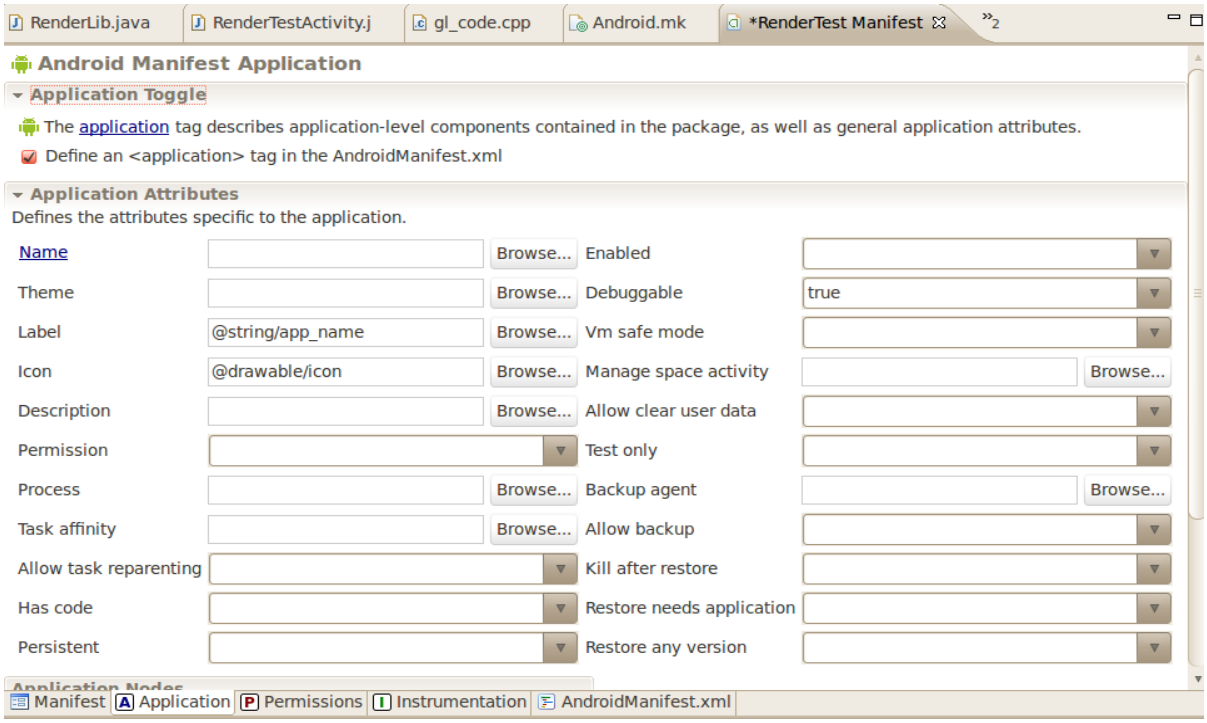
Declare native function in JAVA as

```
public static native void step();
```

Now the app should be OK for compiling and building.

Debug Setting

Step VI: Set android:debuggable = "true"



Step VII: run ndk-gdb from console

Open terminal and cd to the application directory. Export path for adb with command:

export PATH=$PATH:~/workspace/android-sdk-linux_x86/platform-tools

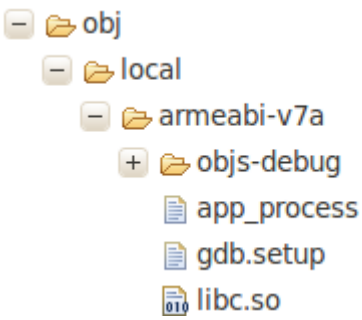
Run ndk-gdb with command:

~/Research/android-opencv/android-ndk-r6b/ndk-gdb (or PATH/ndk-gdb)

As shown in the figure below. Sometimes a bp need to be set in JAVA to make sure the gdb has enough time to start

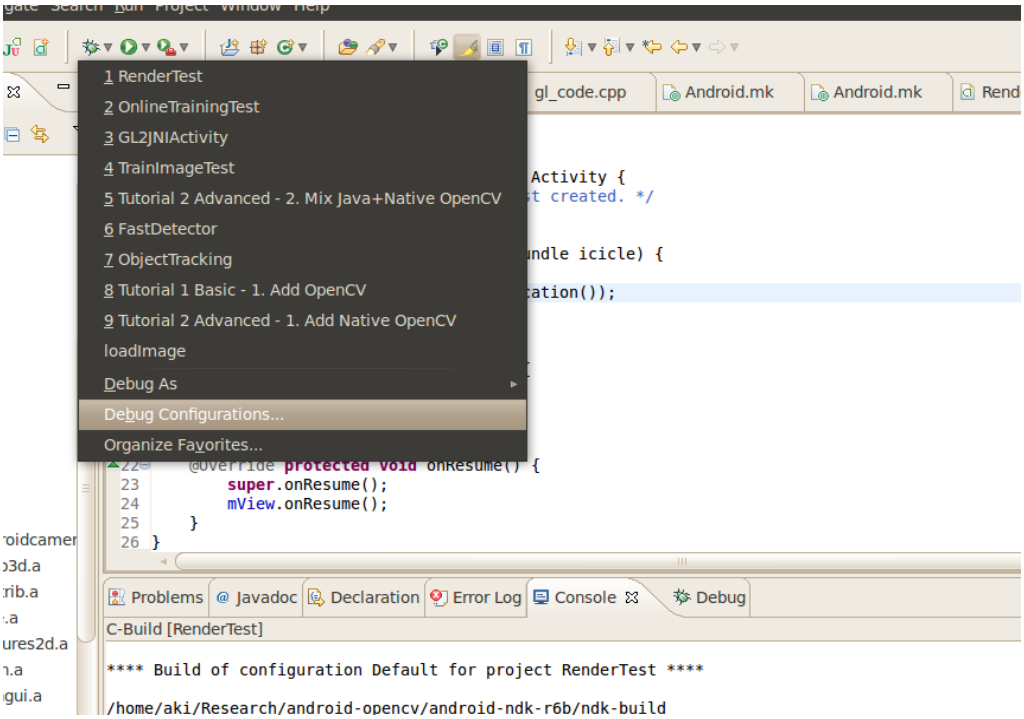
```
File  Edit  View  Terminal  Help
gralloc.s5pc110.so: No such file or directory.
Error while mapping shared library sections:
libIMGegl.so: No such file or directory.
Error while mapping shared library sections:
libEGL_POWERVR_SGX540_120.so: No such file or directory.
Error while mapping shared library sections:
libGLESv1_CM_POWERVR_SGX540_120.so: No such file or directory.
Error while mapping shared library sections:
libGLESv2_POWERVR_SGX540_120.so: No such file or directory.
Error while mapping shared library sections:
libpvrANDROID_WSEGL.so: No such file or directory.
Error while mapping shared library sections:
libglslcompiler.so: No such file or directory.
(no debugging symbols found)
warning: Unable to find dynamic linker breakpoint function.
GDB will be unable to debug shared library initializers
and track explicitly loaded dynamic code.
0xafd0ee48 in __futex_syscall3 ()
   from /home/aki/Research/android-opencv/RenderTest/obj/local/armeabi-v7a/libc.
so
(gdb) q
The program is running.  Exit anyway? (y or n) y
aki@Aki:~/Research/android-opencv/RenderTest$ ~/Research/android-opencv/android-
ndk-r6b/ndk-gdb
```

Running ndk-gdb creates ***app_process, gdb.setup*** and ***libc.so*** files in obj/local/armeabi/ of the project directory. Those files will be needed later to configure the debudding environment. Check those files to make sure they are created.

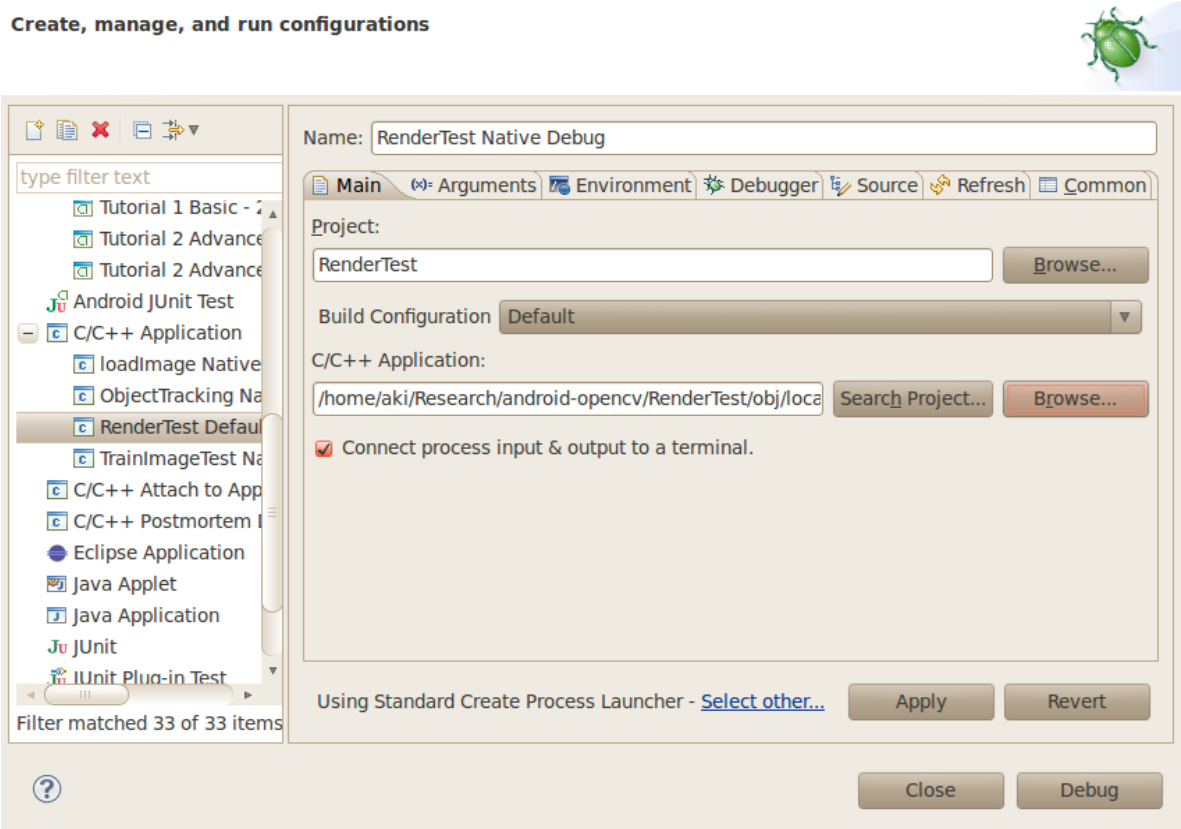


Step VIII: Create C/C++ debug configuration

Click on combo-box like down arrow next to debug button. Pop-up menu will appear and then select ***Debug Configurations..***



In Debug Configurations window select C/C++ Application and press New button as advised on the right pane. Define a new name for the debug config, such as “ProjectName Native Debug”. In the C/C++ Application entry fill the path to the ***app_process*** binary which is located in ***obj/local/armeabi*** of the project directory.

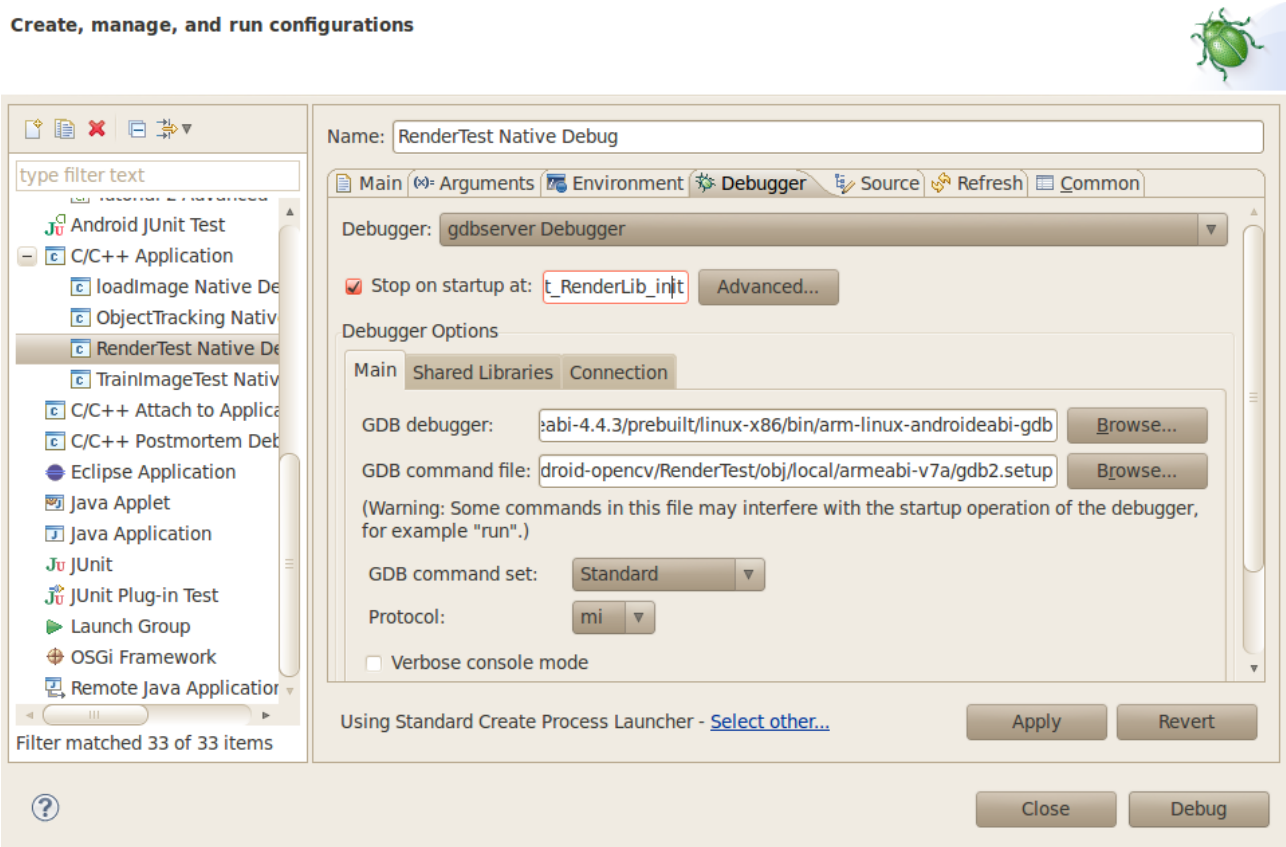


In the debugger tab:
choose gdbserver Debugger as a Debugger.

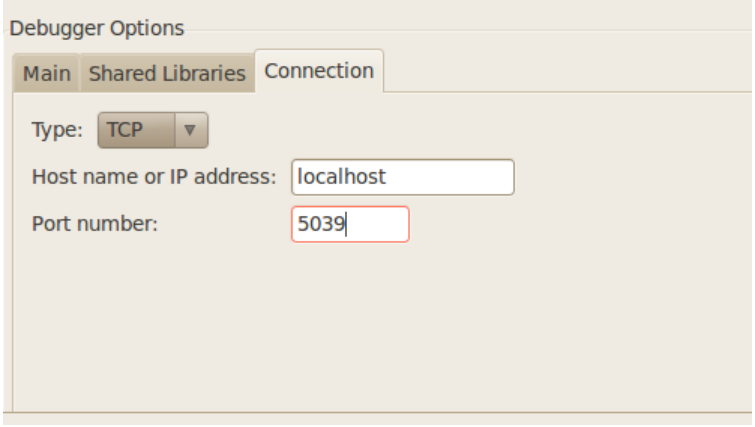
- Set initial breakpoint to some function but Android projects do not contain main function so set to the first function that is expected to be called in native code.

- Set path to GDB debugger: android-ndk-r6b/toolchains/arm-linux-androideabi-4.4.3/prebuilt/linux-x86/bin/arm-linux-androideabi-gdb

- Set path to GDB command line: obj/local/armeabi-v7a/gdb2.setup. Note gdb2.setup does not exist, but will be created soon



In the connection tab below, choose TCP as a type of connection and choose 5039 as a Port Number



Now the debug configuration is done, apply and close.

Step IX: Create gdb2.setup file and ndk-gdb-eclipse file.

Go to the obj/local/armeabi/ subdirectory of your project and copy gdb.setup file to gdb2.setup file. Remove target remote :5039 line from gdb2.setup.

Go the directory with Android ndk and copy ndk-gdb to ndk-gdb-eclipse. Remove execution of adb command form ndk-gdb-eclipse. Because Eclipse will run the gdb binary itself. So we have to remove the execution of dbg from ndk-gdb. (This only needs to be done once, not need for every proejct)

Step X: Finally you can debug Native codes in Eclipse!

Put breakpoint in Java code after the execution of System.loadLibrary(). Start the application in debug mode by clicking on Debug button. It will automatically choose Android debug mode. Later you will have to take care to choose Android Java debugging configuration by clicking on combo arrow associated with the debug button.

When execution reach the breakpoint run ndk-gdb-eclipse from your project directory and start debugging in C/C++ debug mode (You need to have the terminal concole standby).