**Using native codes with Android NDK**

|  |  |
| --- | --- |
| Article ID | 15 |
| Last Reviewed on | **July 15,2010** |
| Created by | **Senthil prabu** |
| Review Board | **Murari Krishnan** |
| Approved by |  |

**Description**

This article explains the steps to using Android NDK and developing android native application with eclipse on a windows platform.

**Installable required**

* **Android SDK**
  + <http://developer.android.com/sdk/download.html?v=android-sdk_r05-windows.zip>
* **Android NDK**
  + <http://dl.google.com/android/ndk/android-ndk-r4b-windows.zip>
* **Cygwin**
  + <http://www.cygwin.com/setup.exe>

* **Eclipse 3.5 IDE**
  + <http://www.eclipse.org/downloads/download.php?file=/technology/epp/downloads/release/galileo/SR2/eclipse-jee-galileo-SR2-win32.zip>

**Introduction:**

NDK provides you with native code libraries from C and C++ sources. These libraries will be transformed into packages files. Android 1.5 provides stable headers for some applications like the standard C library, the standard math library, the JNI interface. Developer should be aware of is that by using NDK the application will become more complicated and harder to debug.

**JNI in J2SE:**



**Developing Native library in Windows:**

SpimeJNI.java

SpimeJNI.class

SpimeJNI.h

SpimeJNI.c

(C Implementations)

SpimeJNI.dll

SpimeJNI

Javac javah

cl -Ic:\java\include -Ic:\java\include\win32

-MD -LDSpimeJNI.c -FeSpimeJNI.dll Java

**Developing Native library in Linux or Solaris:**

SpimeJNI.java

SpimeJNI.class

SpimeJNI.h

SpimeJNI.c

(C Implementations)

LibSpimeJNI.so

SpimeJNI

Javac javah

cc -G -I/java/include -I/java/include/linux

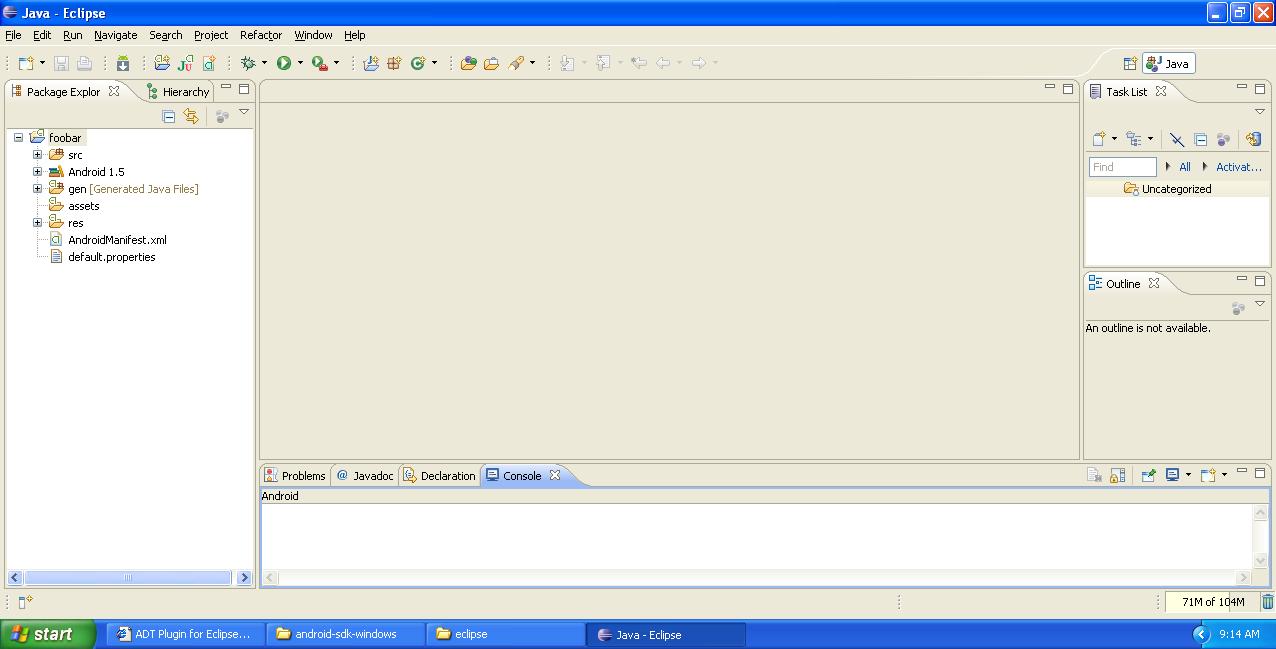
SpimeJNI.c -o libSpimeJNI.so Java

**Notes:** Interface library may have a different name depending on the OS. For example, under Linux or Solaris, the Java System.loadLibrary("SpimeJNI"); searches for a library named “libSpimeJNI.so.” But under Windows, it searches for “SpimeJNI. dll”

**JNI in Android:**

Android NDK (Native Development Kit) simplifies working with native code. It includes the entire tool chain needed to build for your target platform (ARM). It is designed to help you to create that *shared library.* Following sequence is lead to develop the Android NDK application.

## Install Eclipse and Android SDK



Installation Note on: [\\spimeone\Resources\KB Docs \KB\_13\_Adroid SDK\_Eclipse Integration.doc](file:///\\spimeone\Resources\KB%20Docs%20\KB_13_Adroid%20SDK_Eclipse%20Integration.doc)

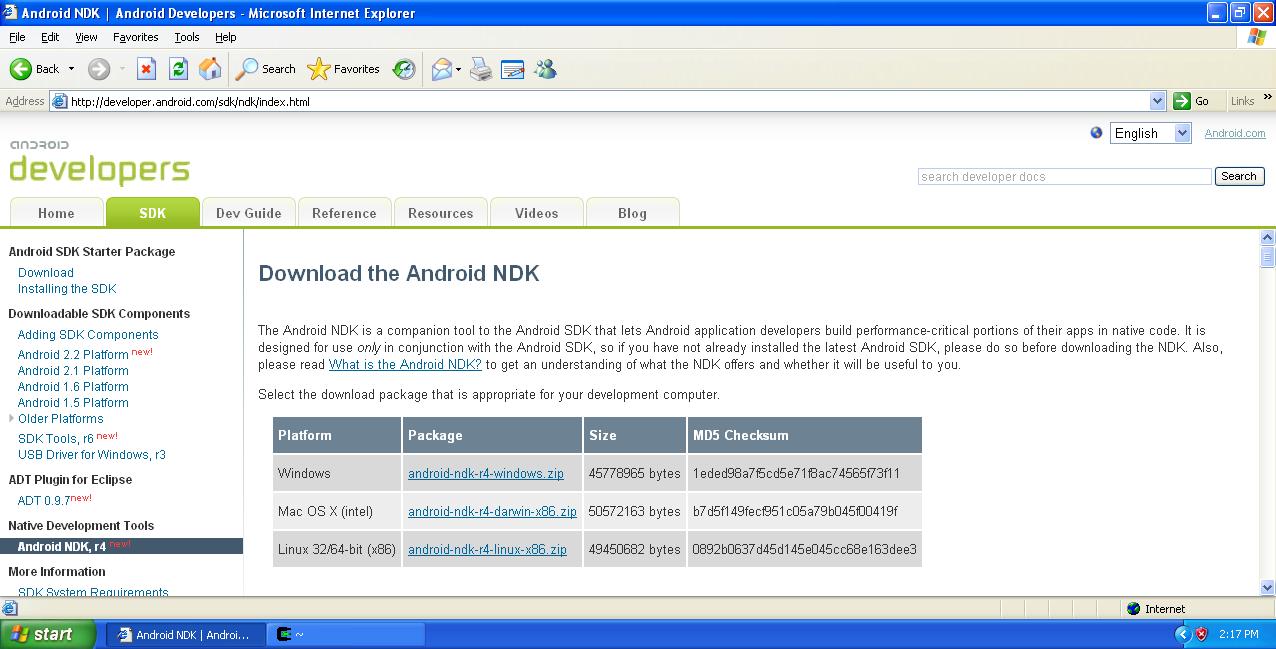
## Installing Android NDK

**Android NDK**: The Android NDK provides tools that allow Android application developers to embed components that make use of native code in their Android applications.

Android applications run in the Dalvik virtual machine. The NDK allows developers to implement parts of their applications using native-code languages such as C and C++. This can provide benefits to certain classes of applications, in the form of reuse of existing code and in some cases increased speed. The NDK provides A way to embed the corresponding native libraries into application package files (.apk) that can be deployed on Android devices, A set of native system headers and libraries that will be supported in all future versions of the Android platform, starting from Android 1.5.

Download the NDK zip for Windows and extract it somewhere, be sure that **there are no blank spaces in the path <** NDK home>. In my case, I extracted it to **C:\**, so the path is **C:\android-ndk-r4**.

NDK download page:



## Install cygwin

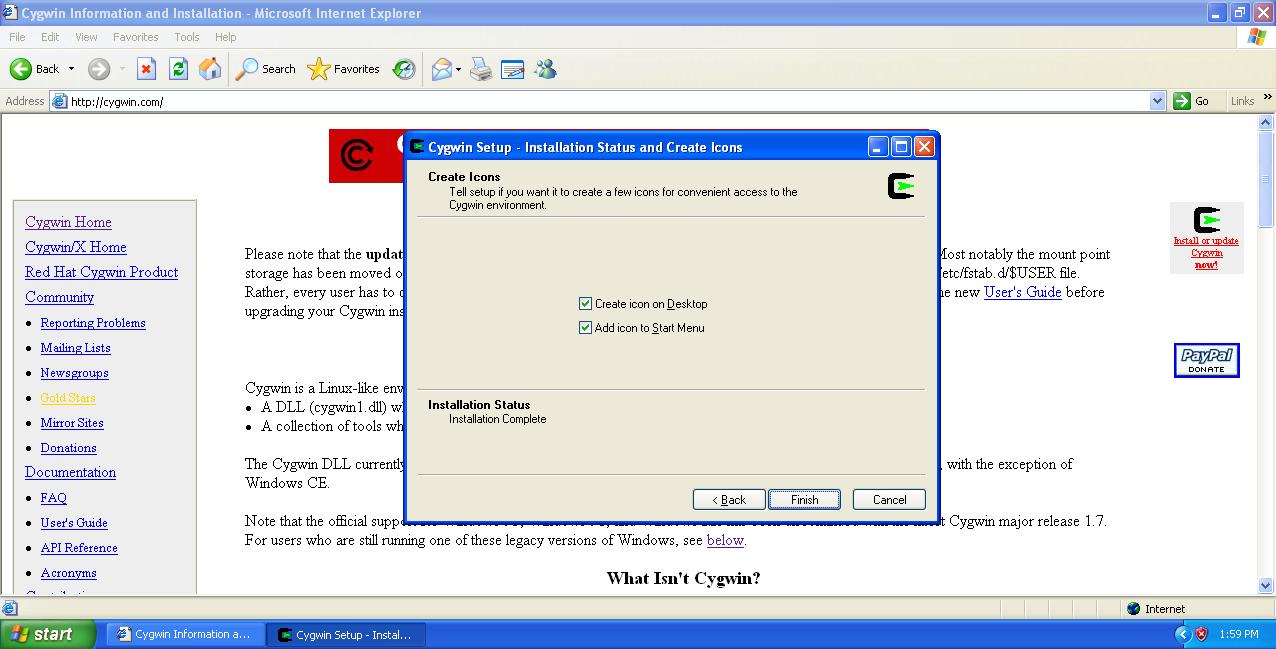
**Cygwin:** Cygwin is a way to make Windows support some linux functionality. If you install cygwin on your windows machine you'd be able to run some linux software on windows. Cygwin provides native integration of Windows-based applications, data, and other system resources with applications, software tools, and data of the linux -like environment.

The NDK is the Android Native Development Kit. It allows you to write parts of your application in native code (C/C++) and integrate them into your application. Your application still runs under the Dalvik VM but it can load shared objects creating using a cross compiler. The NDK contains all the necessary tools and build scripts to generate native code binaries.

Install Cygwin to path without spaces for maximum compatibility. Later am install directory is "**c:\cygwin\**".

Installation Note on: [\\spimeone\Resources\KB Docs \ KB\_14\_Installing Cygwin for Android.pdf](file:///\\spimeone\Resources\KB%20Docs%20\KB_13_Adroid%20SDK_Eclipse%20Integration.doc)

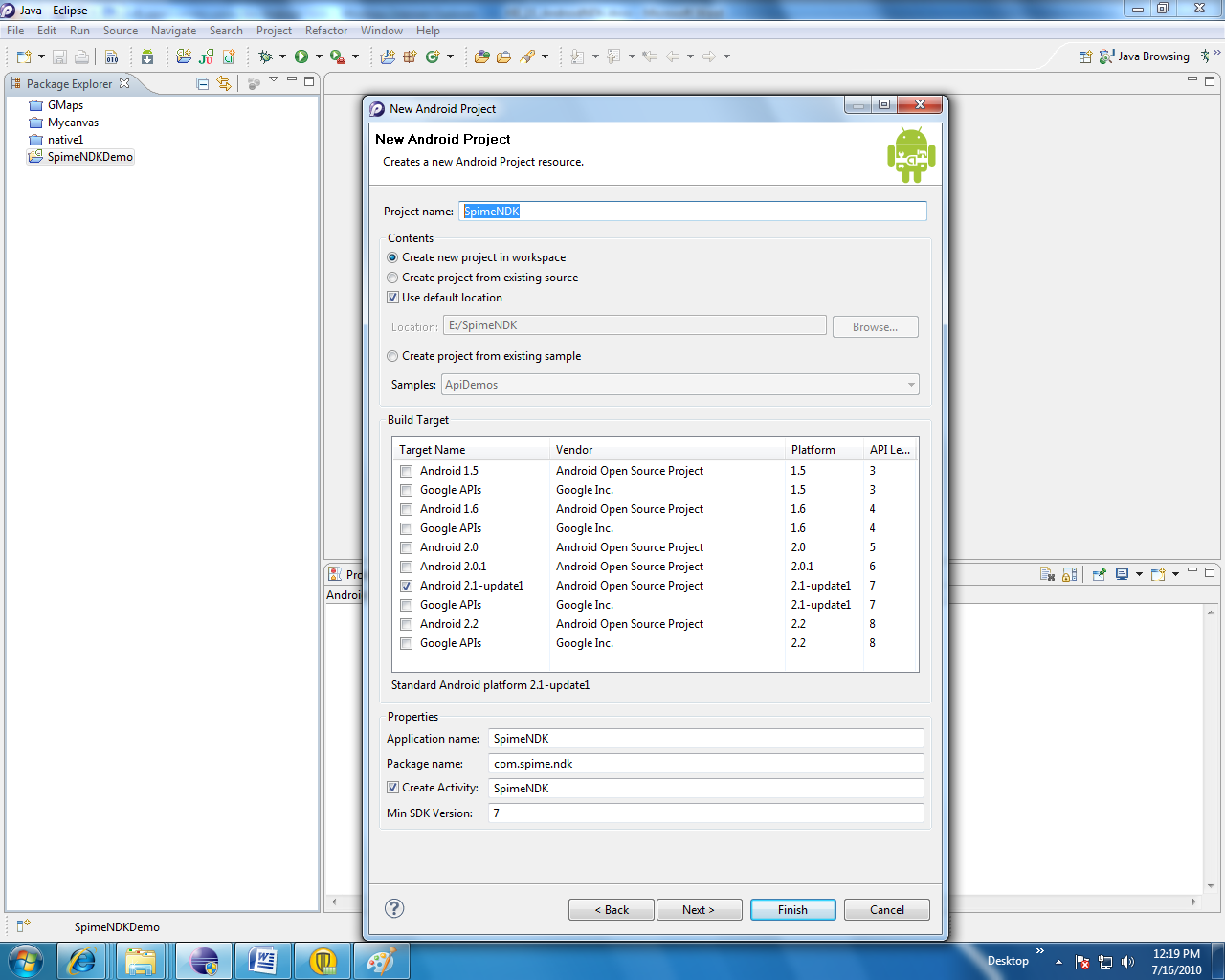
**Cygwin installation:**

****

**Developing Android NDK in Windows:**

## Step 1:

Create a Normal Android Project. Let’s call it ‘SpimeNDK’

****

## Step 2:

The main source file should look like this and Build the project.

**package** com.spime.ndk;

**import** android.app.Activity;

**import** android.app.AlertDialog;

**import** android.os.Bundle;

**public** **class** SpimeNDK **extends** Activity {

// load the library - name matches jni/Android.mk

**static** {

System.*loadLibrary*("SpimeNDK");

}

// declare the native code function - must match SpimeNDK.c

**private** **native** String invokeNativeFunction();

/\*\* Called when the activity is first created. \*/

@Override

**public** **void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(R.layout.*main*);

// this is where we call the native code

String hello = invokeNativeFunction();

**new** AlertDialog.Builder(**this**).setMessage(hello).show();

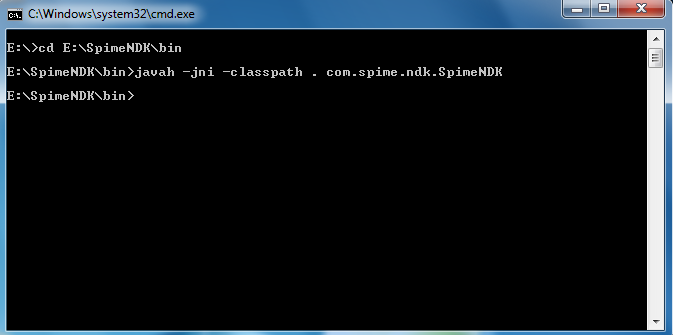
}

}

## Step 3:

If you are using Eclipse, the class files built will be in the 'bin' directory. Navigate to bin then

javah -jni –classpath . com.spime.ndk.SpimeNDK



Ea

## 

Then you will get the com\_spime\_ndk\_SpimeNDK.h in **<EclipseWorkspace>/SpimeNDK/bin**

## 

## Step 4:

Next create a ‘**jni**’ folder in the Eclipse project path **<**EclipseWorkspace**>/**SpimeNDK/jni

Now move **<EclipseWorkspace>/SpimeNDK/bin/ com\_spime\_ndk\_SpimeNDK.h** file to **<EclipseWorkspace>/SpimeNDK/jni** folder

## Step 5:

Now, create the C implementation file SpimeNDK.c in **<EclipseWorkspace>/SpimeNDK/jni** folder. It should look like this.

**#include** "com\_spime\_ndk\_SpimeNDK.h"

/\*

\* Class: com\_spime\_ndk\_SpimeNDK

\* Method: invokeNativeFunction

\* Signature: ()Ljava/lang/String;

\*/

JNIEXPORT jstring JNICALL **Java\_com\_spime\_ndk\_SpimeNDK\_invokeNativeFunction**(JNIEnv \* env, jobject obj){

**return** (\*env)->NewStringUTF(env, "Hello Spime NDK DEMO !");

}

**Step 6:**

Now you have to create the ‘**Android.mk’** file. This file describes yournative code module to the build system. This file is created in the same dir as the **c source** file.

# Copyright (C) 2009 The Android Open Source Project

#

# Licensed under the Apache License, Version 2.0 (the "License");

# you may not use this file except in compliance with the License.

# You may obtain a copy of the License at

#

# http://www.apache.org/licenses/LICENSE-2.0

#

# Unless required by applicable law or agreed to in writing, software

# distributed under the License is distributed on an "AS IS" BASIS,

# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

# See the License for the specific language governing permissions and

# limitations under the License.

#

LOCAL\_PATH := $(call my-dir)

include $(CLEAR\_VARS)

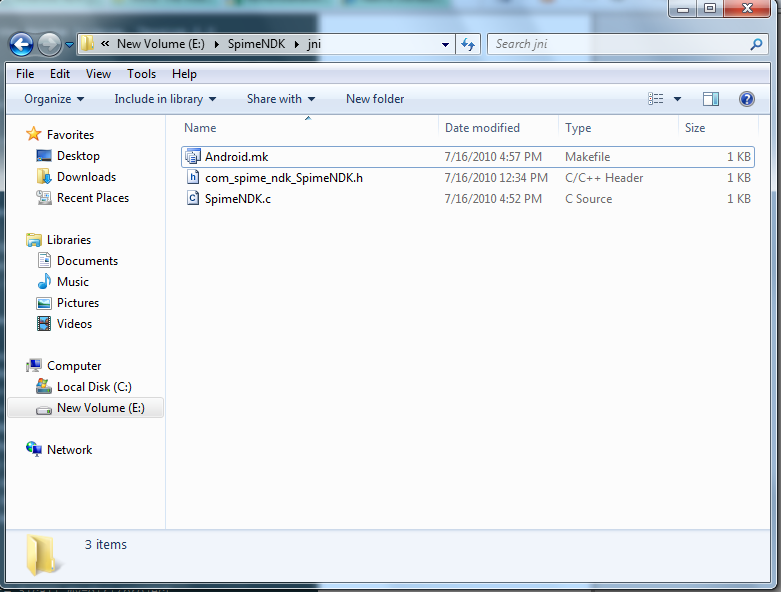
LOCAL\_MODULE := SpimeNDK

LOCAL\_SRC\_FILES := SpimeNDK.c

include $(BUILD\_SHARED\_LIBRARY)

e1.NativeLib

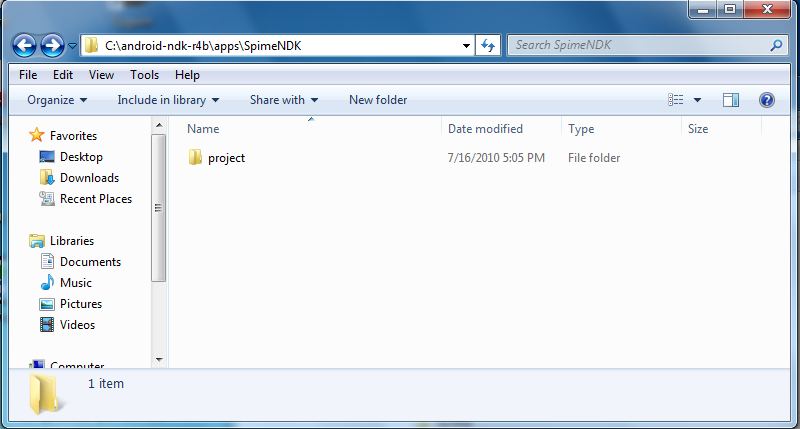
Now your ‘**<EclipseWorkspace>/SpimeNDK/jni** ‘folders look like this

****

**Step 7:**

Now you go to <NDKHOME> in my case **C:\android-ndk-r4**.

Then create ‘**apps’** folder in the <NDKHOME> next go into the ‘**apps**’ folder and create ‘**SpimeNDK** ‘ folder and next go into ‘**SpimeNDK** ‘ and create ‘**project**’ folder

Now your folders look like this

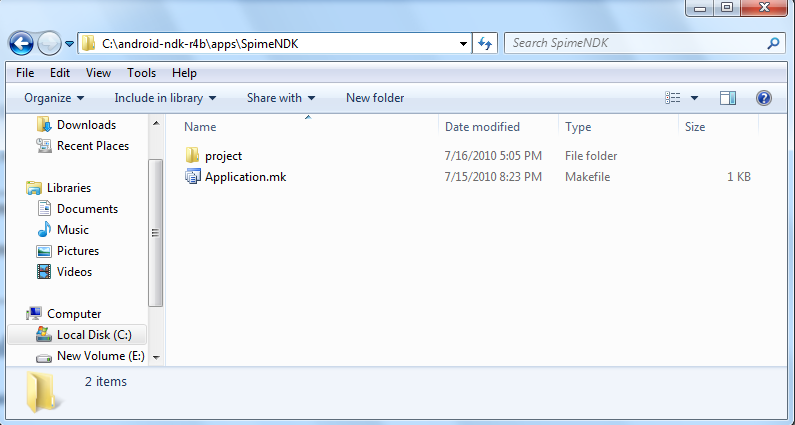
**Step 8:**

Then, navigate one level up to ‘<NDKHOME> /apps/**SpimeNDK’** and create the **Application.mk** file.

APP\_PROJECT\_PATH:= $(call my-dir)/project

APP\_MODULES := SpimeNDK

Now your folder looks like this:

****

**Step 9:**

In the **'project'** folder copy your entire eclipse project ‘**SpimeNDK**’

copy entire Eclipse <EclipseWorkspace>/ SpimeNDK project into <NDKHOME>/apps/SpimeNDK/project folder

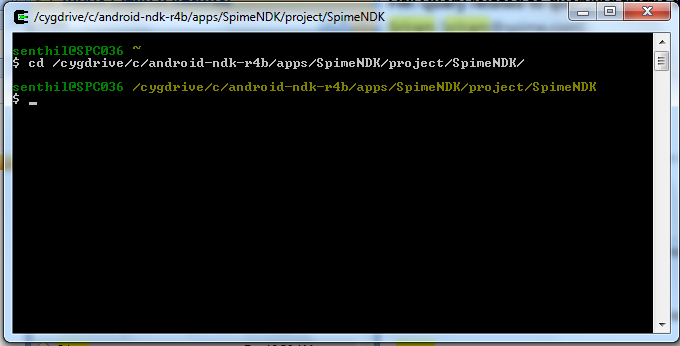
**Step 10:**

Now, in order to create a **binary library** from the C source that we wrote, we will use **a combination of Cygwin and Android NDK tools**. Launch the Cygwin console and use the **cd** command to, go to the folder where your project is.

Notice that Windows drives are mapped under **/cygdrive** within the emulated Linux environment you work with in the Console. Open **C:\cygwin\Cygwin.bat** and type In my case, the command line is:

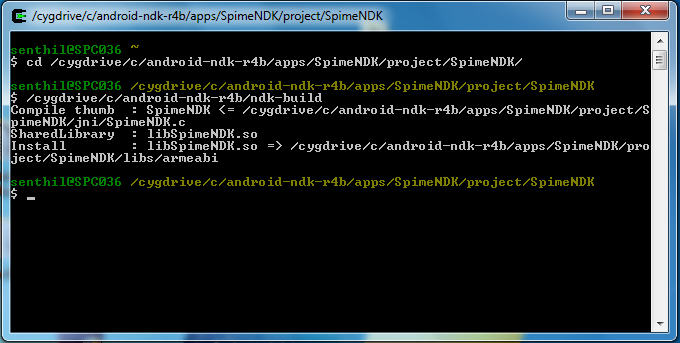
**cd /cygdrive/c/android-ndk-r4b/apps/SpimeNDK/project/SpimeNDK/**

**Note** : Folder names and commands are case-sensitive. You can use tab to auto-complete folder names. If you installed NDK to c drive remember to change /cygdrive/d/ to /cygdrive/c/

****

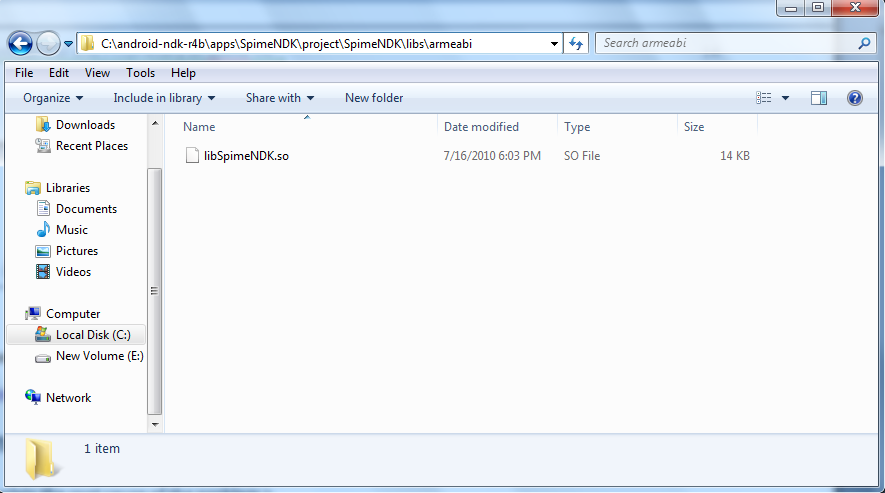
**Step 11:**

Type ‘ **/cygdrive/c/android-ndk-r4b/ndk-build**’ and press Enter. This will build the ‘SpimeNDK’ component. You should see the following output if everything goes well:



If everything goes well, libSpimeNDK.so should appear in

<NDKHOME>/apps/SpimeNDK/project/SpimeNDK/libs/armeabi

****

**Step 12:**

Finally move shared libraries to the proper location in the application's project directory. Now you copy the ‘NDKHOME>/apps/SpimeNDK/project/SpimeNDK/libs’ folder to<EclipseWorkspace>/ SpimeNDK

You have to repeat the **ndk-build** command every time you modify the C/C++ source of your NDK code. Eclipse ADT does not support NDK so you need to do it from the Cygwin console. Don’t forget to refresh Eclipse every time!. You can force rebuild by **ndk-build -B**. By default compilation is done when it's needed (if the code is changed).

Continue using Eclipse as normal, it includes the above component (the ‘**.so**’ file) automatically next time it builds the java code. So now that we have the native C library implemented, compiled, and placed in the right place, let's see how we can call it from our Activity. It's actually rather simple you just have to instantiate the instance of your SpimeNDK class and from there on, it's just a regular Java object.



# Reference

* <http://mindtherobot.com/blog/452/android-beginners-ndk-setup-step-by-step/>
* <http://earlence.blogspot.com/2009/07/writing-applications-using-android-ndk.html>
* <http://marakana.com/forums/android/examples/49.html>
* http://www.techjini.com/blog/2009/10/26/android-ndk-an-introduction-how-to-work-with-ndk/
* <http://fixnum.org/blog/2010/android_ndk/>
* <http://www.telesense.co.uk/?page_id=106>
* http://www.brighthub.com/mobile/google-android/articles/49732.aspx