Speeding Up the Android* Emulator on Intel® Architecture

By Costas Styliano... (/en-us/user/128696), Wed, Nov 27, 2013

Abstract:

If you are an Android* (http://software.intel.com/en-us/android) developer who is unhappy with the performance of the Android emulator, then this document is for you. Over and over again, we have heard from many Android developers that the emulator is slow and painful to work with, but this should not be the case! If you are using a fairly up-to-date computer with an Intel® processor that has Intel® Virtualization Technology enabled running Microsoft Windows* or Apple Mac OS*, then you can use the Intel® Hardware Accelerated Execution Manager (http://software.intel.com/en-us/android/articles/intel-hardware-accelerated-execution-manager) (Intel® HAXM), or KVM for Linux*, to accelerate the Android Emulator by an order of magnitude very easily, which will speed-up your testing and debugging of your Android applications. This document explains all the steps required to accelerator the emulator and how to work with it. We then explain how to use the NDK to compile x86 native code and the correct way to submit APKs containing x86 native libraries to the Google Play store. Intel HAXM is also used to accelerate the Tizen* emulator, but this is out of scope in this documentation. For more information, go to tizen.org within the SDK section.

Contents

1. Introduction
2. Installation
2.1. Prerequisites
2.2. Installation on Windows
2.3. Installation on Linux
2.3.1. Installation of KVM
2.4. Creating an AVD (Android* Virtual Device)
3. Best Known methods
3.1. Testing your application with the emulator from Eclipse
3.2. Submitting multiple APKs for different ABIs vs. submitting fat binaries to Google Play
3.3. Compile your NDK for x86
3.3.1. Adding the path of NDK to your Environment Variable
3.3.2. Compiling with the NDK
3.3.3. Another way of compiling with the NDK

1. Introduction

This document will guide you through installing the Intel® Hardware Accelerated Execution Manager (http://software.intel.com/en-us/android/articles/intel-hardware-accelerated-execution-manager) (Intel® HAXM), a hardware-assisted virtualization engine (hypervisor) that uses Intel® Virtualization Technology (http://software.intel.com/en-us/articles/intel-virtualization-technology-a-primer) (Intel® VT) to speed up Android* development on Windows*. It also explains how to setup a hardware-assisted

KVM on Linux* and best known methods in compiling natively and submitting apps to the Google Play Store for x86.

2. Installation

2.1. Prerequisites

- You need to have the Android SDK (http://software.intel.com/en-us/articles/installing-the-android-sdk-for-intel-architecture) installed.
- Your computer must have an Intel processor with support for Intel VT-x, EM64T and Execute Disable(XD) Bit functionality enabled from the BIOS.

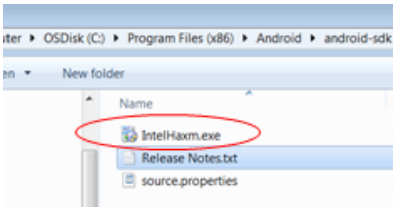
2.2. Installation on Windows

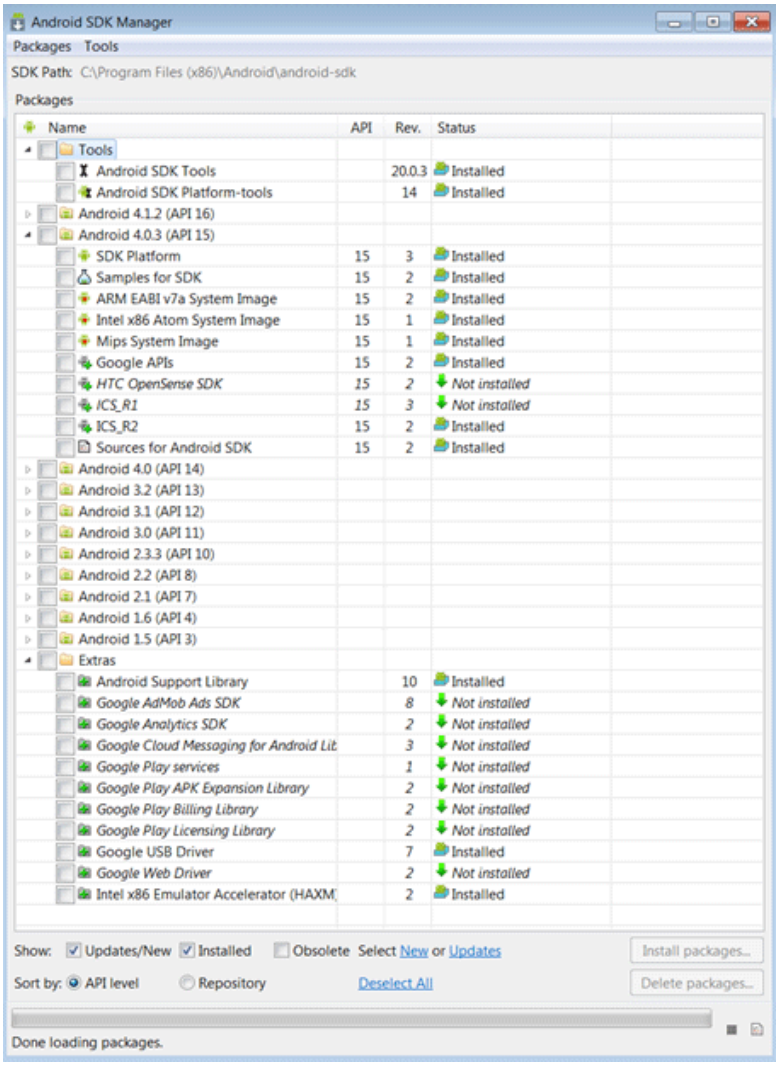
After you have installed the Android SDK, open the SDK Manager. In the extras section, you can find the Intel HAXM.

Check the box and click the 'Install packages…' button, once you have installed the package, the status will appear as 'Installed', ***which is misleading*** as this is not the case. The SDK only copies the Intel HAXM executable on your machine, and it is up to you to install the executable.

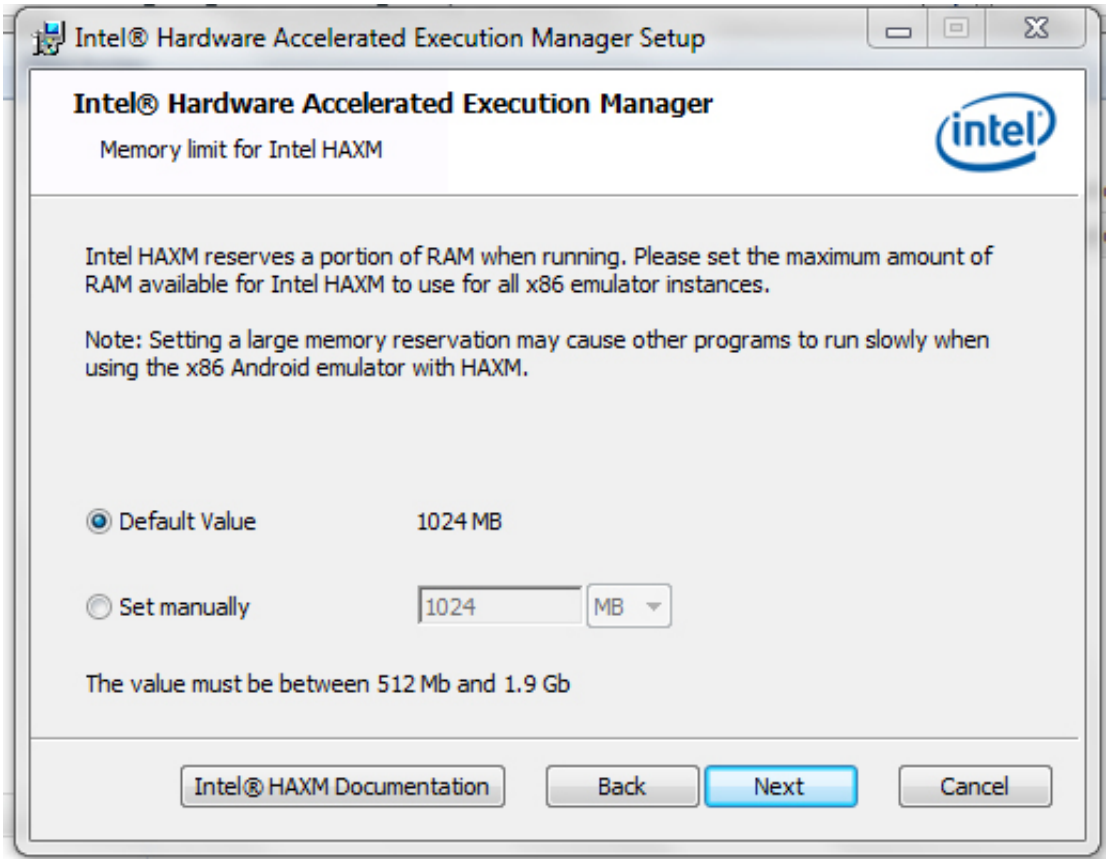
To install the Intel HAXM executable, search your hard drive for IntelHaxm.exe (or IntelHAXM.dmg on Mac OS X). If you left everything to default, it should be located at C:\Program Files\Android\android-sdk\extras\Intel\Hardware_Accelerated_Execution_Manager\IntelHaxm.exe.

Intel HAXM only works in combination with one of the Intel® Atom™ processor x86 system images, which are available for Android 2.3.3 (API 10), 4.0.3 (API 15), 4.1.2 (API 16), 4.2.2 (API 17). These Intel system images can be installed exactly the same way as the ARM-based images via the SDK manager.

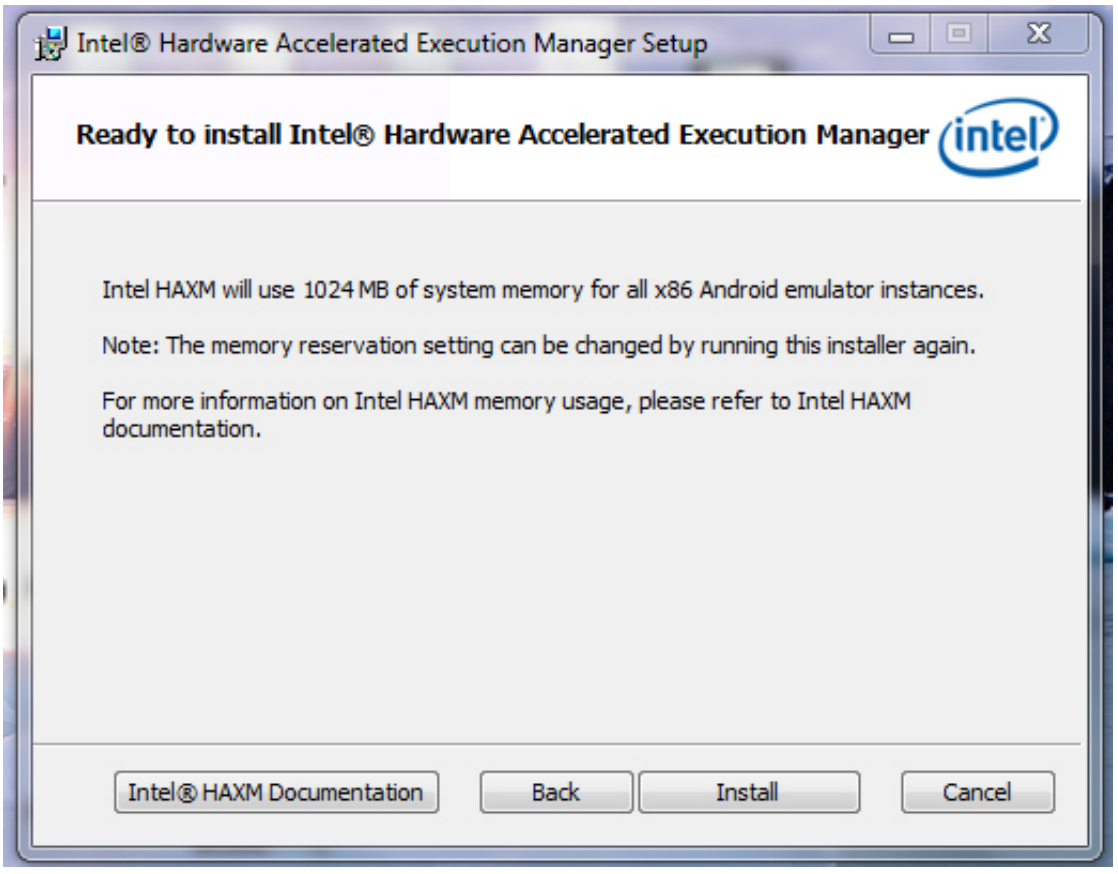




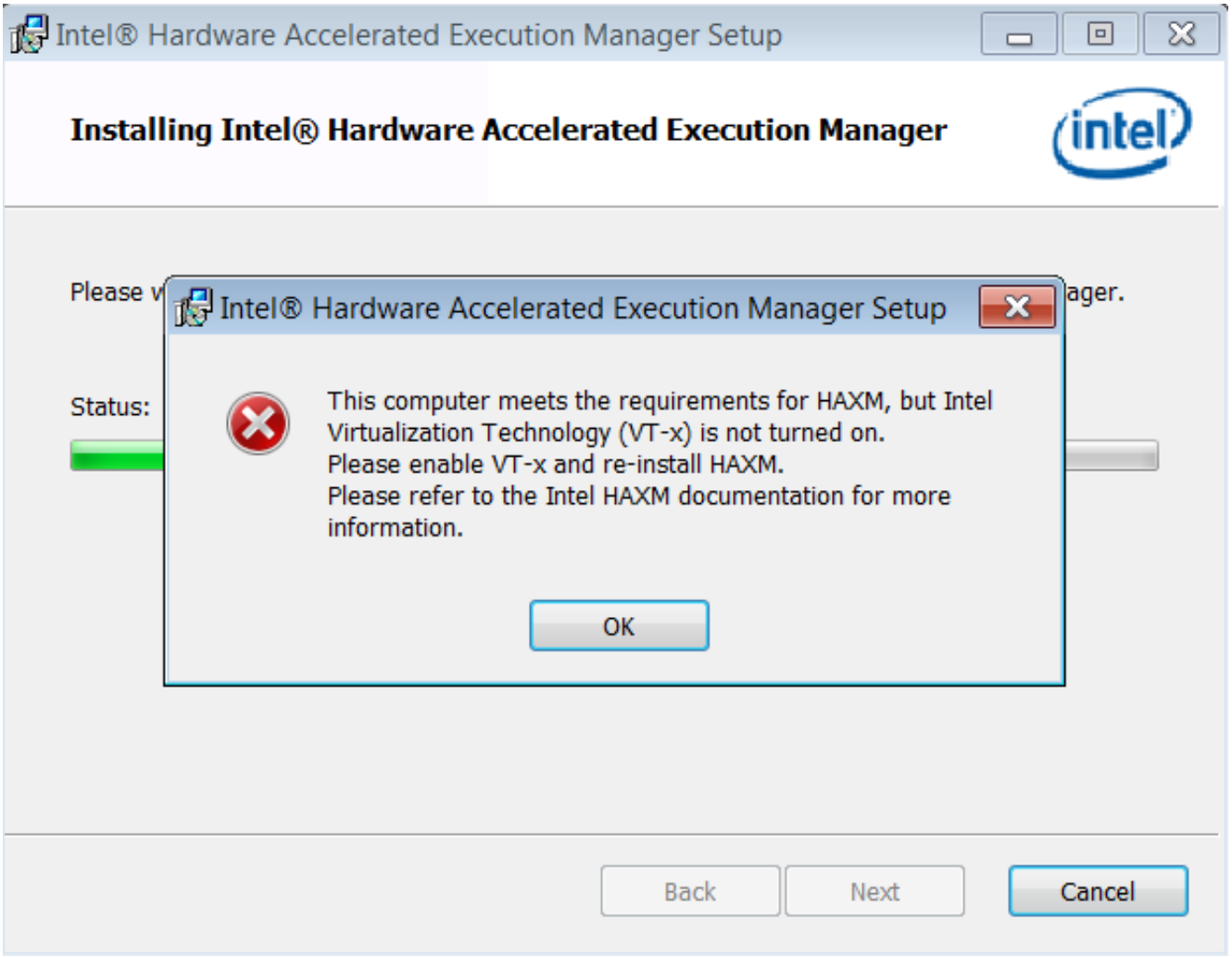
When you click on the IntelHaxm executable, a welcome screen is displayed like this:



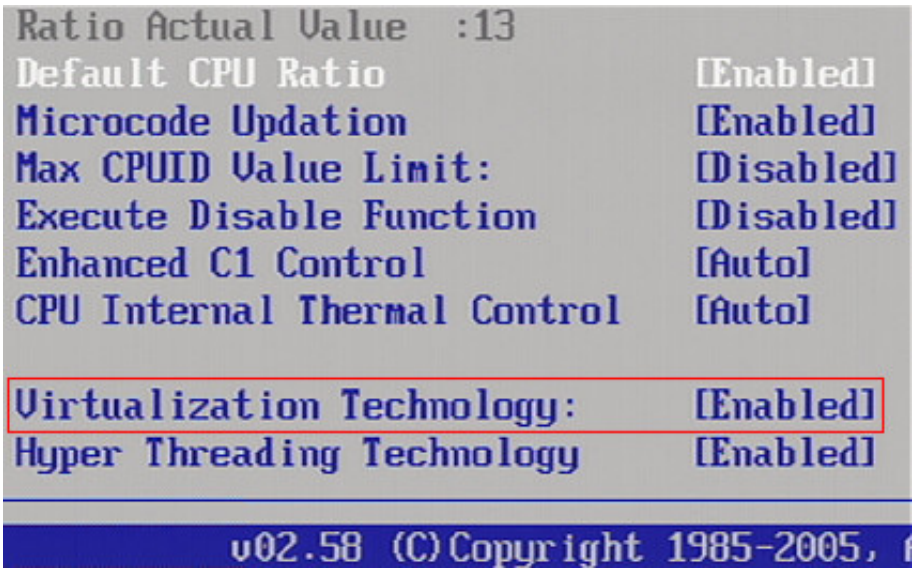
You can adjust how much RAM memory is allocated to Intel HAXM. After you do this, click Next. The next screen confirms the memory allocation. If everything is as you wish, click **Install**.



In order to be able to install the Intel HAXM, you need to have Intel VT-x enabled in your BIOS, otherwise you will get an error like this during install:



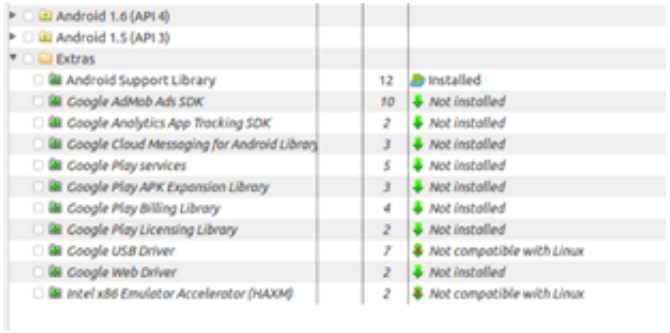
If you get this error, go to your BIOS and enable this feature.



The second option to download the Intel HAXM and x86 Emulator System image is to go directly to the web site: http://software.intel.com/en-us/android and download the necessary components from there.

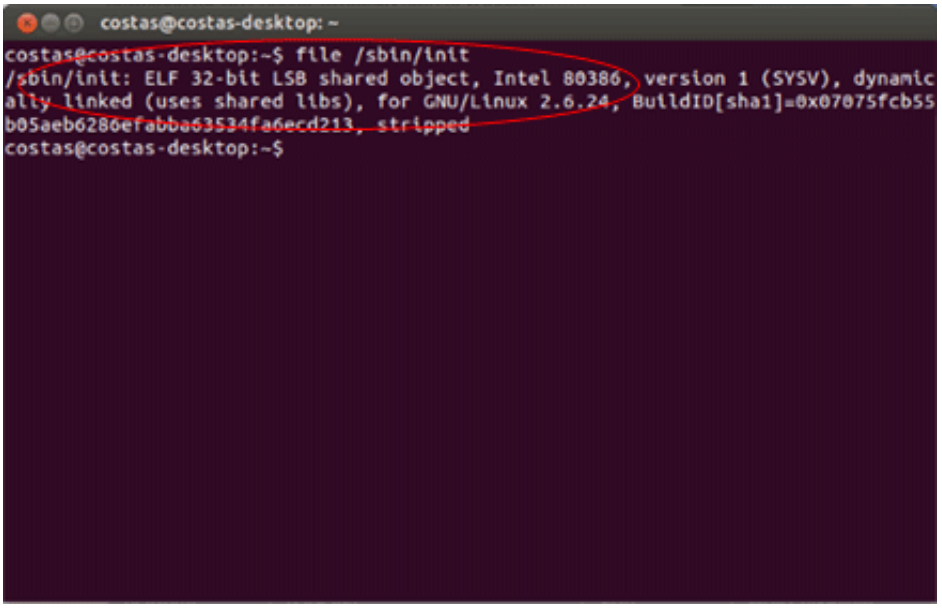
2.3. Installation on Linux

The steps to accelerate the Android emulator for Linux are different than for Windows and Mac OS X as Intel HAXM is not compatible with Linux so you would need to use KVM (kernel-based virtual machine) instead. The steps below were carried out using Ubuntu* 12.04 and may differ slightly with other Linux distributions.



As with Windows (and Mac OS X), first you need to download the Android SDK from the Android developer site. You will find an ADT (Android Developer Tool) bundle that contains both Eclipse* IDE and the Android SDK. Download the zip file and extract it to your Linux machine. Make sure you choose the right version for your Linux distribution, either 32- or 64-bit. You can check this easily by using the command:

file /sbin/init



Before you start installing the packages required for KVM, it is recommended to make sure you have the latest repository, and you can do this by typing:

sudo apt-get update

2.3.1. Installation of KVM

To install and run KVM, which is a full virtualization solution for Linux on x86 hardware (i.e., Intel VT), you first need to check if your CPU supports hardware virtualization, you can do this by typing:

egrep –c '(vmx|svm)' /proc/cpuinfo

If the result is 0, that means your CPU does not support hardware virtualization, which is necessary to run the KVM. If you get 1 or more, that means you're good to go, but still make sure it is enabled in your BIOS (see Section 2.2).

Next, you need to install KVM unless you already have it installed. You can check if your processor supports KVM by typing:

kvm-ok

If you have KVM, you will see this:

"INFO: Your CPU supports KVM extensions
INFO: /dev/kvm exists
KVM acceleration can be used"

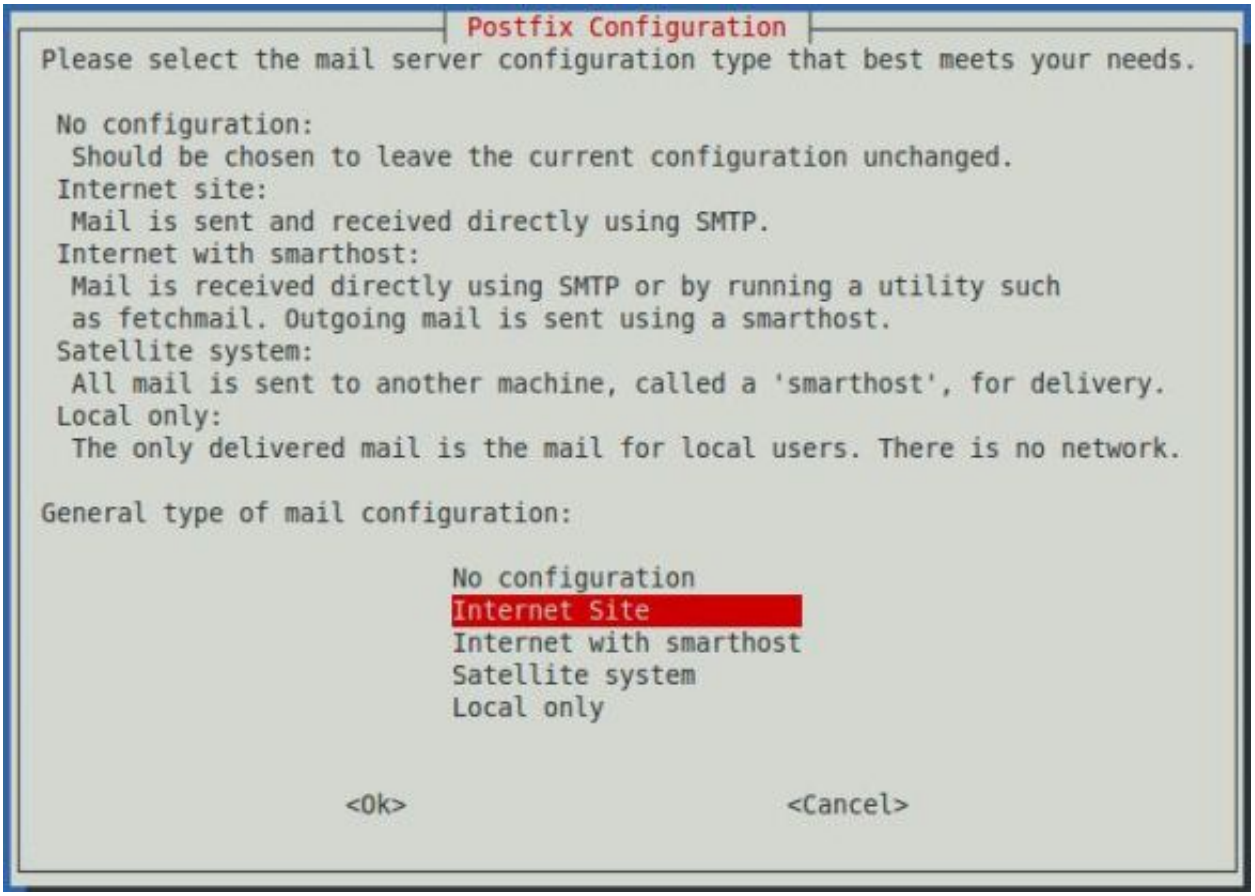
Otherwise, if you see this, you need to go to BIOS and turn on Intel VT:

"INFO: KVM is disabled by your BIOS
HINT: Enter your BIOS setup and enable Virtualization Technology (VT),
and then hard poweroff/poweron your system
KVM acceleration can NOT be used"

The next step is to install the KVM and a few other packages needed. To do so, type

sudo apt-get install qemu-kvm libvirt-bin ubuntu-vm-builder bridge-utils

In the next window, you can select No configuration if you want to leave your config unchanged:



Next, add your user to the **KVM** group and **libvirtd group.** To do so, type:

sudo adduser your_user_name kvm
sudo adduser your_user_name libvirtd

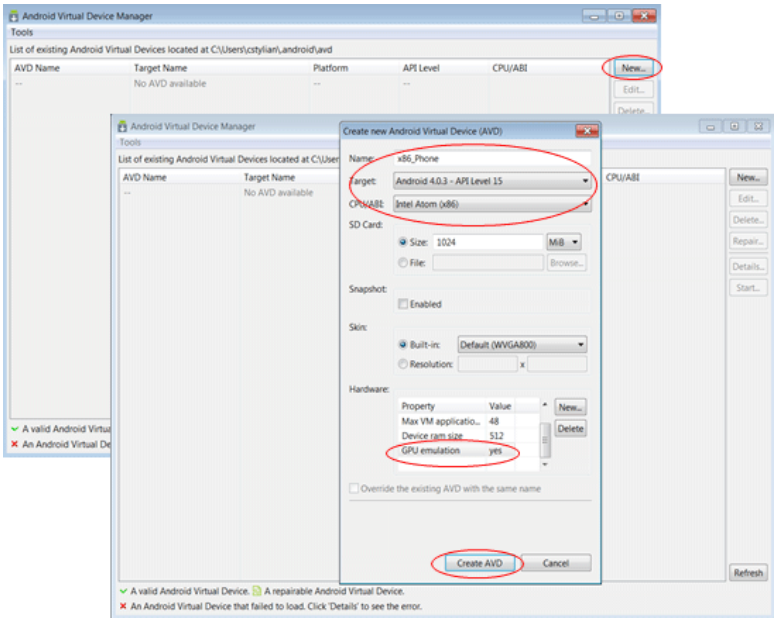
After the installation, re-login so that the changes take effect. You can test the installation by typing:

sudo virsh -c qemu:///system list

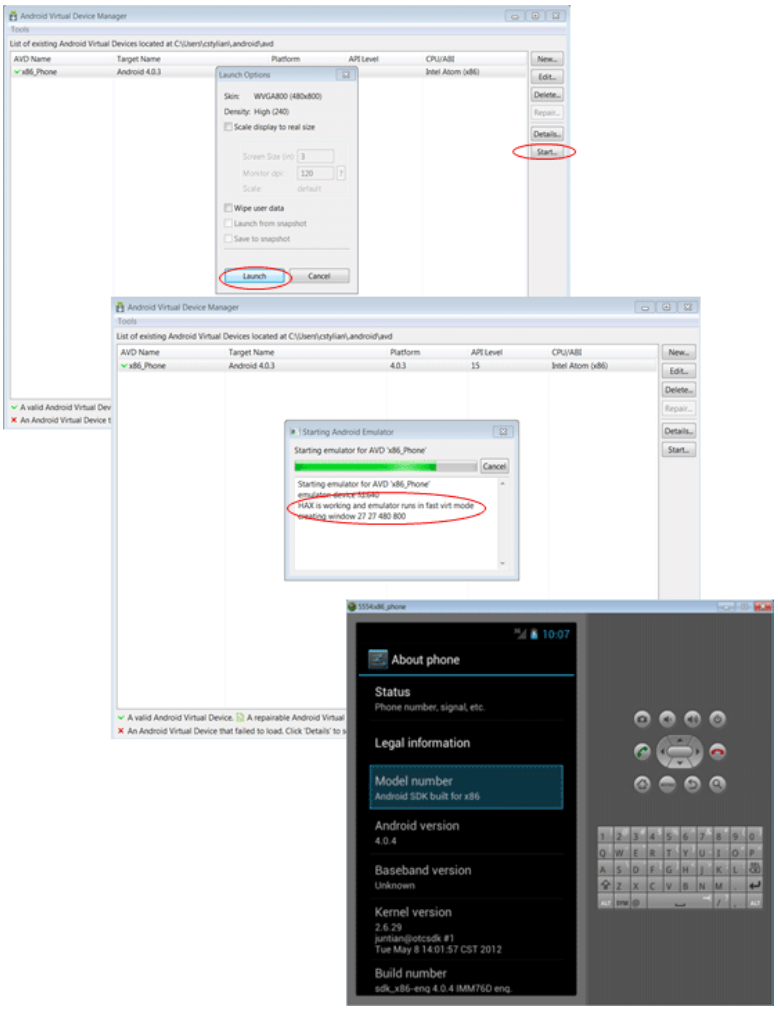
You are now ready to go to the next step, which is creating and running the AVD (Android Virtual Device). This procedure is the same for both Linux and Windows.

2.4. Creating an AVD (Android* Virtual Device)

After installing the SDK and Intel HAXM (or KVM with Linux), you can create a virtual device that has hardware-accelerated emulation. To do that, go to the AVD Manager and create a new device. Make sure you select **Intel Atom (x86)** as the CPU/ABI. The selection appears in the drop-down only if you have the Intel x86 system image installed, for additional graphics smoothness switch on the GPU emulation when creating your AVD.



Click on **New** and create your x86 AVD. Make sure you select an API that is supported by an x86 system image, CPU/ABI is set to x86, and you have enabled GPU (OpenGL ES*) emulation. Once you have done that, click on the **Create AVD** to create the AVD.



You can launch the x86 AVD by clicking on Start, then Launch.

If you are successful with the installation, when the emulator is starting, you will get a dialog box showing that Intel HAXM is running in fast virtual mode.

If you need further convincing that you are using an x86 system image, you can always check the details in the 'About phone' within the emulator.

The performance gain that you'll see with Intel HAXM or KVM depends on your PC, drive, memory, etc., but should be between 5x to 10x order of magnitude. The screen shot below shows a side-by-side comparison from an x86/HAXM enabled-AVD versus an ARM-based AVD. The x86 AVD booted to the lock screen within 15 seconds whereas the non-Intel AVD took 40 seconds – a big difference.

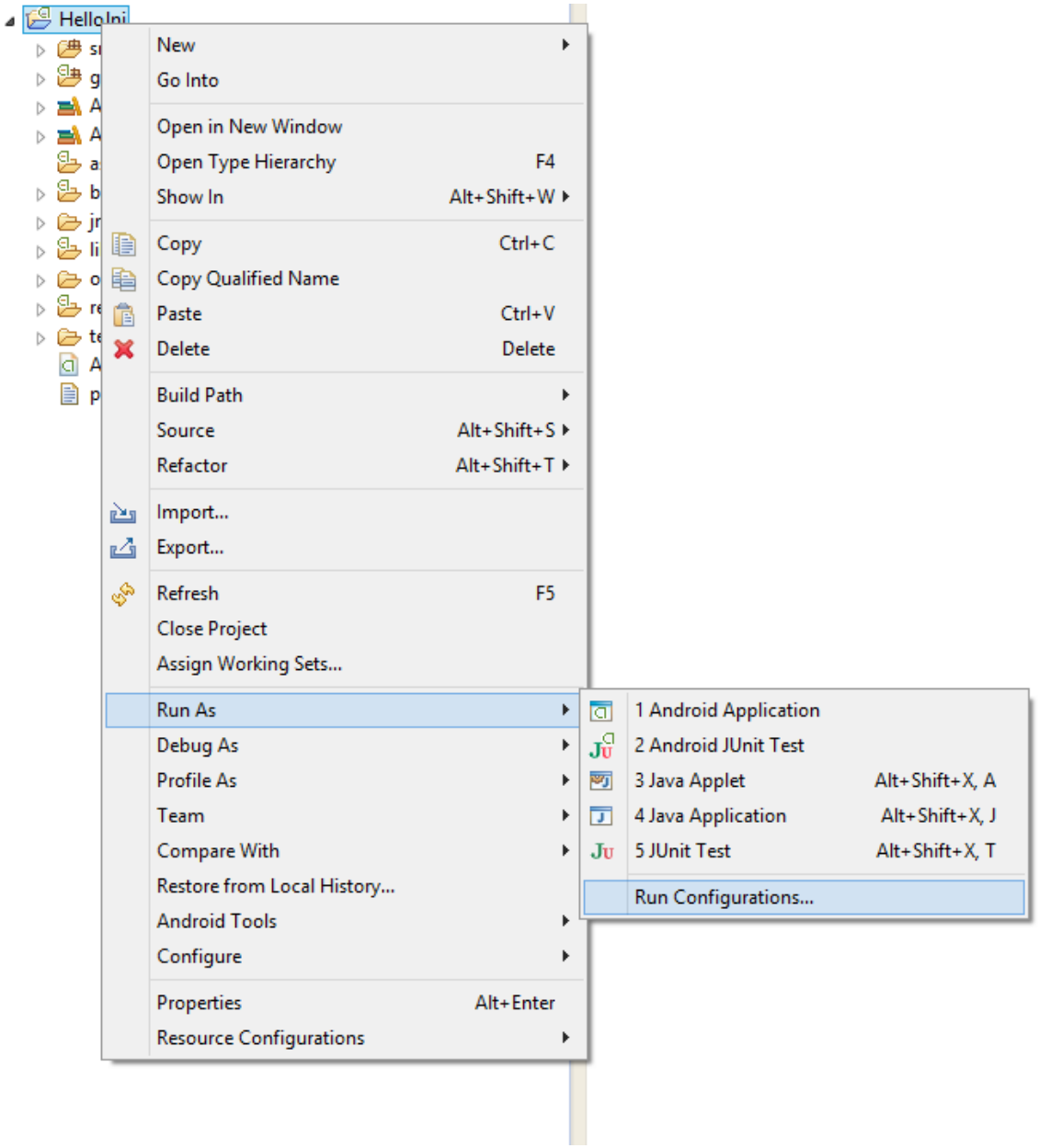
[The performance gain that you'll see with the Intel HAXM (or KVM) should be between 5x to 10x depending on your system configuration: Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark* and MobileMark*, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. Configuration: Mac Book Pro was used for testing in this case. **For more information go to http://www.intel.com/performance** (http://www.intel.com/performance)"]

3. Best Known methods

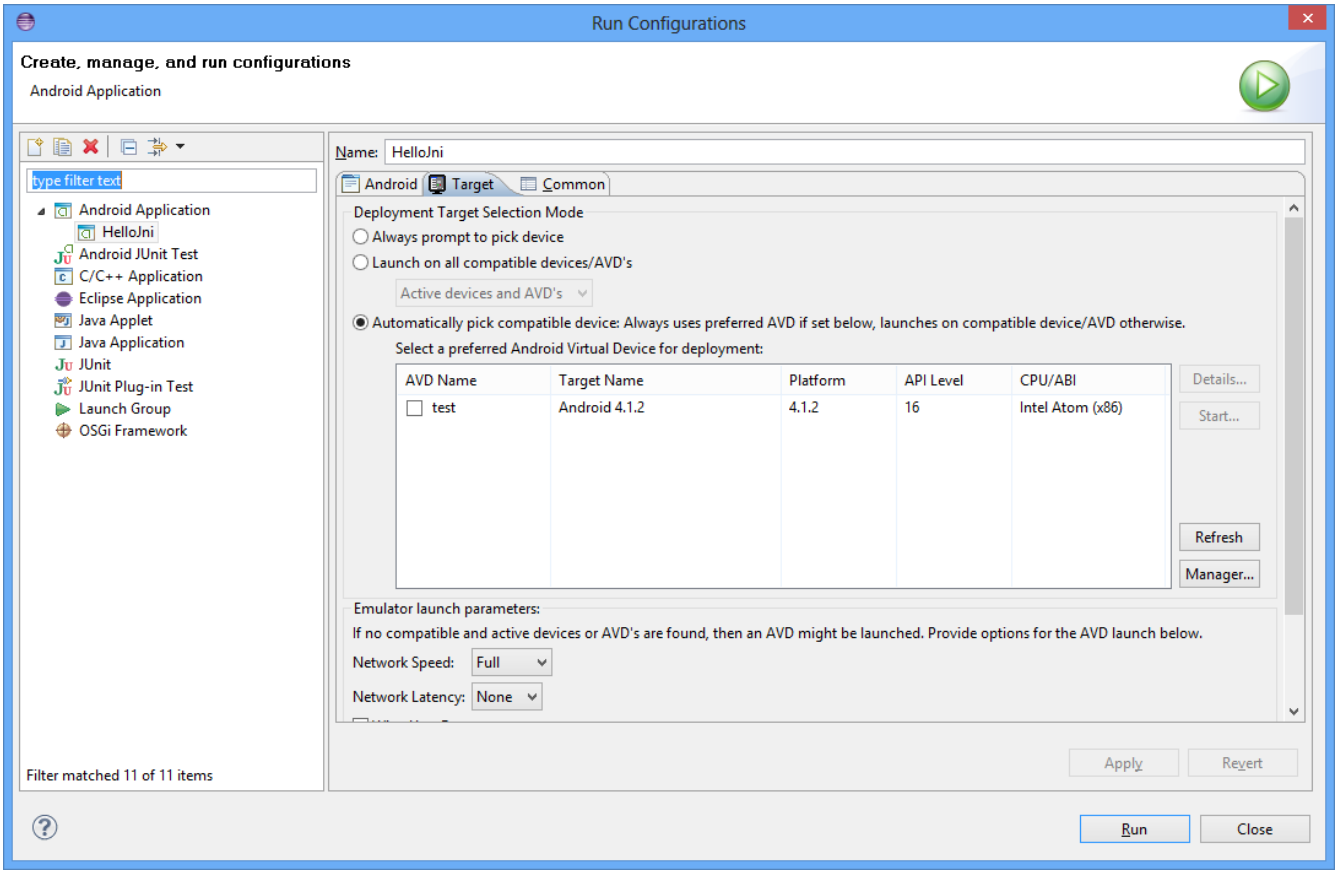
3.1. Testing your application with the emulator from Eclipse

Whether it's an NDK-based application or Dalvik* application, you can use Intel HAXM to speed up the emulator where you do testing. If you are developing with Eclipse, you can follow these easy steps to make sure you are using the Intel HAXM when you start the emulator.

First, make sure you created your AVD as described in step 2. If you have the AVD ready, go to Run As -> Run Config as shown below:



You should land on a page like this:



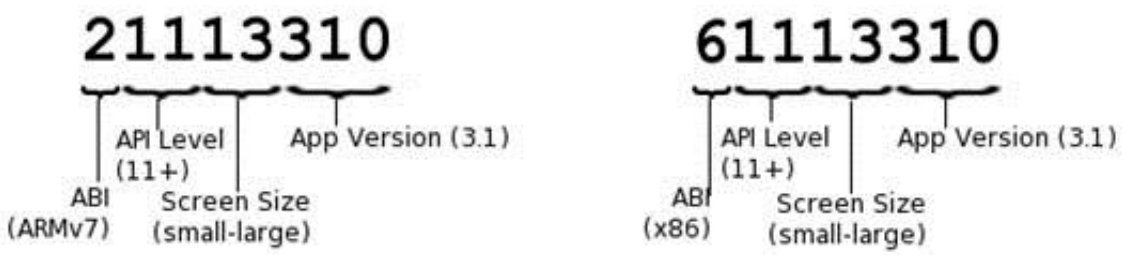
Here, you can select the AVD you want by checking the box. After you have created your AVD and set up your configuration, start compiling your project and debug it with the emulator by selecting Run As -> Android Application. This will automatically start the hardware-accelerated AVD.

After the AVD has started, you should see the home screen of your application (after unlocking the screen).

3.2. Submitting multiple APKs for different ABIs vs. submitting fat binaries to Google Play

In the past, you would submit a fat binary for your developed application, which would contain all of your libraries and NDK file(s), not being able to differentiate between the architectures. That would mean that the users had to download the whole APK containing files that are *not* relevant to the specific architectures, i.e., x86 users would have ARM code and vice-versa. The downside of this is that if you have a really fat binary, the user was forced to download a big amount of data that would not apply to the device. Typically, this is still acceptable if your APK is under 10 to 20 MB.

Intel/Google have now implemented a CPU filtering mechanism, meaning you can now submit multiple APKs containing the different libs specific for each architecture by following the suggested versioning code shown below.



The first digit refers to the ABI, i.e., 6 for x86; then the API level that you target, i.e., 11; the screen size, 13; and then the version number of your application: 3.1.0.

Make sure that you have at least an 8-digit version number and assign the highest first digit to the x86 version. In the example above, you would put 6 for x86, 2 for ARMv7 and 1 for ARMv5TE. Doing so will make x86 versions preferred on the x86 devices and ARM versions on ARM devices.

By following these guidelines, you can make sure that your users get the best performance out of the device they own. Furthermore, you may avoid users trying to run applications on specific devices due to code translation problems.

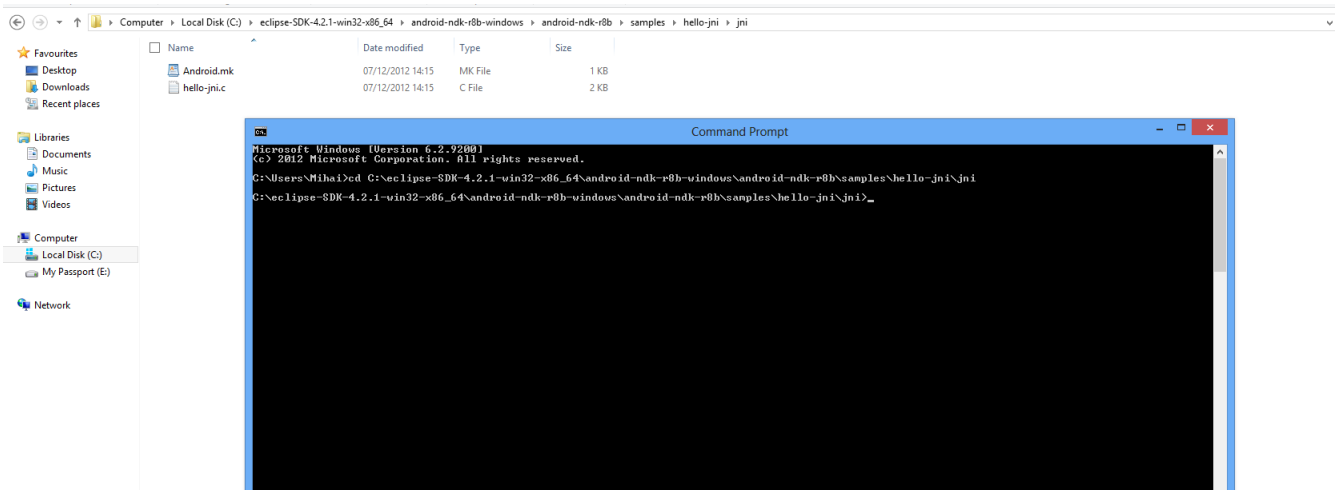
More info can be found here: http://software.intel.com/en-us/articles/google-play-supports-cpu-architecture-filtering-for-multiple-apk.

3.3. Compile your NDK for x86

This section will show you how to compile your application's NDK part for x86.

In order for your NDK-based application to run on an x86 AVD, you need to compile your NDK lib for the x86 architecture. To do so, follow these simple steps:

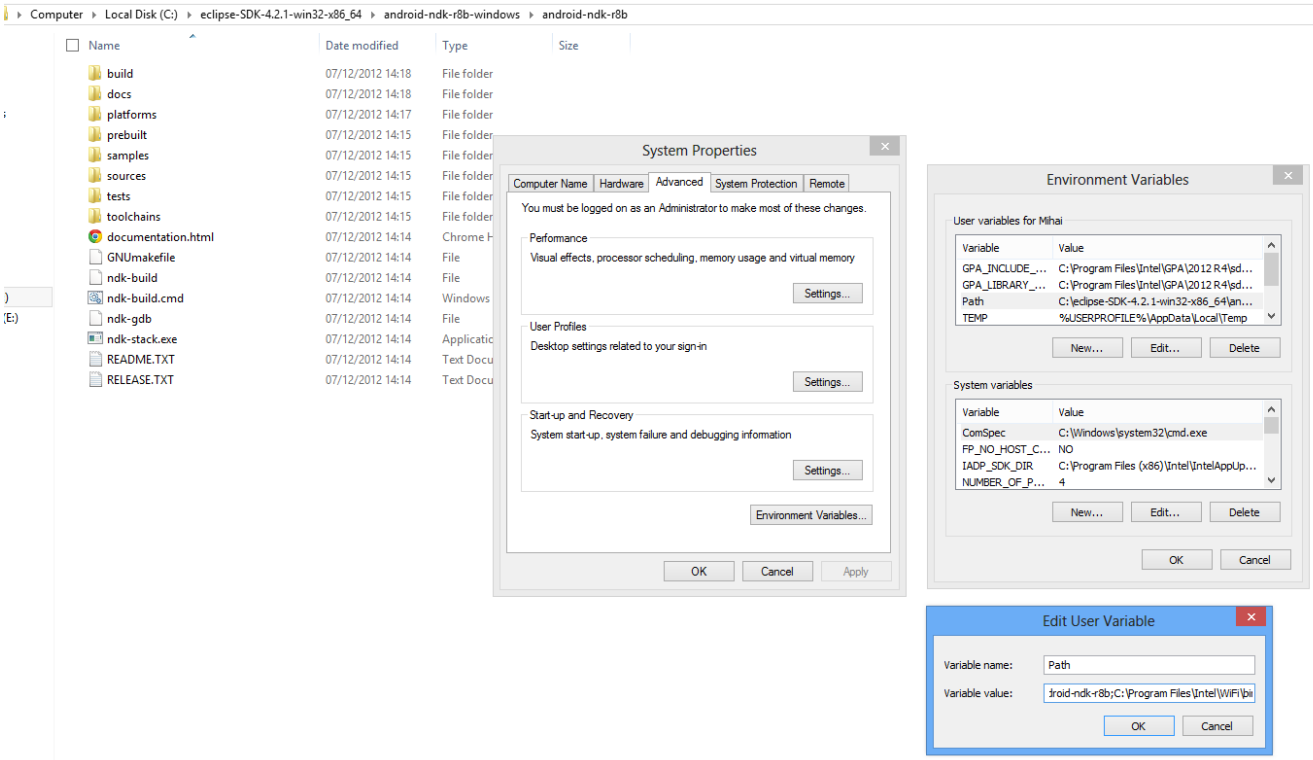
Open a command prompt and navigate to the folder of your NDK files, like below:



Make sure you have set the Environment Variable Path so you can use the ndk-build script from anywhere.

3.3.1. Adding the path of NDK to your Environment Variable

In order to setup your environment variable for the NDK, you need right-click on **Computer**, and select **Properties**. Go to **Advanced system settings** and find Environment variables. Select **Path** and click **Edit**. At the end of the 'Variable Value' string, add the path to your root folder of NDK, the one that contains the ndk-build.cmd file as in the image below:



3.3.2. Compiling with the NDK

After you have navigated with the command prompt to your NDK folder, execute:

ndk-build APP_ABI:=all

This will compile your NDK file for each architecture available, i.e., ARMv5TE, ARMv7, x86, and mips.

To compile for a specific architecture, replace 'all' with the different architectures. For example:

ndk-build APP_ABI:=armeabi armeabi-v7a x86 mips

Make sure you refresh your project in Eclipse in order to capture your latest settings, i.e., latest folders created by the ndk-build script. In the Folder libs of your projects, you should now see four folders, one for each of the architectures.

You are now ready to use the x86 AVD with your NDK application.

3.3.3. Another way of compiling with the NDK

Another way of compiling your native code for all the architectures, including x86 is to modify your **Application.mk** file which you would find in the jni folder. If you do not have an Application.mk file, you can create one yourself and add the instruction below:

APP_ABI:=armeabi armeabi-v7a x86 mips

This way, when you run the batch file, i.e., the ndk-build script, it will compile the libraries for all the architectures available.

Also, for easier use, instead of listing all the architectures, you can just put 'all':

APP_ABI:=all

Related Links and Resources

- Intel Android* Developer Zone (http://software.intel.com/en-us/android)
- Intel HAXM (http://software.intel.com/en-us/android/articles/intel-hardware-accelerated-execution-manager)
- Intel Virtualization Technology (http://software.intel.com/en-us/articles/intel-virtualization-technology-a-primer)
- Installing the Android* SDK for Intel Architecture (http://software.intel.com/en-us/articles/installing-the-android-sdk-for-intel-architecture)

To learn more about Intel tools for the Android developer, visit Intel® Developer Zone for Android (/en-us/android).

Categories: Android* (/en-us/search/site/language/en?query=Android%2A) Phone (/en-us/search/site/language/en?query=Phone) Developers (/en-us/search/site/language/en?query=Developers) Android* (/en-us/search/site/language/en?query=Android%2A)

Comments

Flowck I. Flowck I. Tue, 02/25/2014 - 06:16

I have installed the HAXM in my notebook with a processor
Core I5 and 4GB of RAM but Android Emulator still slow, it stop on screen with the animated word: android.

Top