# TI-Android-GingerBread-2.3.4-DevKit-2.1 DeveloperGuide


## TI Android GingerBread 2.3.4 DevKit 2.1 Developer Guide

The objective of this document is to guide Android developers to get access to Android sources for TI devices, setting up host environment for compilation and enabling debug environment to ease the app development, debugging and deployment.

## Host (PC) setup requirements

The host development environment for Android is based on ubuntu, please install ubuntu version 10.04 or later http://www.ubuntu.com/desktop/get-ubuntu/download. The host installation would need few more Android specific dependencies, these can be installed dynamically over network using below commands.

### For Ubuntu on 32-bit machines

```
  $ sudo add-apt-repository "deb http://archive.canonical.com/
lucid partner"
  $ sudo add-apt-repository "deb-src http://archive.canonical.com/
ubuntu lucid partner"
  $ sudo apt-get update
  $ sudo apt-get install git-core gnupg sun-java6-jdk flex bison gperf
libsdl-dev libesd0-dev libwxgtk2.6-dev build-essential zip curl
libncurses5-dev zlib1g-dev minicom tftpd uboot-mkimage expect
  $ sudo update-java-alternatives -s java-6-sun
```

NOTE: Android Gingerbread (2.3.4) needs Java 6 on ubuntu, whereas the previous version FroYo (2.2) was using Java 5.

## Source locations

TI provides Android sources for all the supported devices in multiple locations, developers can download the sources from the gitorious.org/rowboat repository or use the pre-packaged repo in the DevKit.

### Using gitorious.org/rowboat

A tool called 'Repo' helps to fetch the android sources from gitorious.org/rowboat. Repo is a tool that makes it easier to work with Git in the context of Android.
For more information about Repo, see the link http://source.android.com/source/version-control.html.
To install, initialize, and configure Repo, follow these steps:
Make sure you have a bin/ directory in your home directory, and that it is included in your path:

```
  $ mkdir ~/bin
  $ PATH=~/bin:$PATH
```

Download the Repo script and ensure it is executable:

```
  $ curl https://android.git.kernel.org/repo > ~/bin/repo
  $ chmod a+x ~/bin/repo
```

The following commands help developers to clone sources from Gitorious.org/rowboat repository

```
  $ mkdir rowboat-android
  $ cd rowboat-android
  $ repo init -u git://gitorious.org/rowboat/manifest.git -m
TI-Android-GingerBread-2.3.4-DevKit-2.1.xml
  $ repo sync
```

## Using pre-packaged repo

The above method is ideal but is time consuming, so developers can use the pre-packaged Android sources in DevKit package.

NOTE: Once the sources are generated developers can do a “repo sync” to pull latest updates (if any) from gitorious.org/rowboat

```
  $ mkdir rowboat-android
  $ cd rowboat-android
  $ Download the pre-packaged repo from http://software-dl.ti.com/
dsps/dsps_public_sw/sdo_tii/TI_Android_DevKit/
TI_Android_GingerBread_2_3_4_DevKit_2_1/index_FDS.html
  $ tar -xvzf XXXX.tar.gz
  $ cd XXXX
  $ ./.repo/repo/repo sync --local-only
```

This will generate the sources for

- Android File system
- Android Linux Kernel (in kernel directory)
- u-boot boot loader (in u-boot directory)
- x-loader (in x-loader directory)
- Toolchain (prebuilt/linux-x86/toolchain/arm-eabi-4.4.3/bin)

# Tool chain setup

Setup the tool-chain path to point to arm-eabi- tools in prebuilt/linux-x86/toolchain/arm-eabi-4.4.3/bin

```
  $ export
PATH=rowboat-android/prebuilt/linux-x86/toolchain/arm-eabi-4.4.3/bin:$PATH
```

# Compilation procedure

## To build x-loader

Execute the following commands

```
  $ make CROSS_COMPILE=arm-eabi- distclean
  $ make CROSS_COMPILE=arm-eabi- omap3evm_config
  $ make CROSS_COMPILE=arm-eabi-
```

This command will build the x-loader Image "x-load.bin"

To create the MLO file used for booting from a MMC/SD card, sign the x-loader image using the signGP tool found in the Tools/signGP directory of the Devkit.

```
  $ ./signGP ./x-load.bin
```

The signGP tool will create a .ift file, rename the x-load.bin.ift to MLO

```
  $ mv x-load.bin.ift MLO
```

## To build boot loader (u-boot)

- Change directory to u-boot-omap3

```
  $ cd u-boot-omap3
```

- Do the following to build u-boot for AM37x EVM

```
  $ make CROSS_COMPILE=arm-eabi- distclean
  $ make CROSS_COMPILE=arm-eabi- omap3_evm_config
  $ make CROSS_COMPILE=arm-eabi-
```

This command will generate the u-boot Image "u-boot.bin"

NOTE: Copy the "mkimage" from "tools" folder to /usr/bin folder on your host machine, this is needed for kernel uImage generation

## To build Linux kernel

- Change directory to kernel

```
  $ cd kernel
```

- Do the following to build sources for AM37x EVM

```
  $ make ARCH=arm CROSS_COMPILE=arm-eabi- distclean
  $ make ARCH=arm CROSS_COMPILE=arm-eabi- omap3_evm_android_defconfig
  $ make ARCH=arm CROSS_COMPILE=arm-eabi- uImage
```

This will generate uImage (kernel image) in kernel/arch/arm/boot folder

## To build Android filesystem

To Build the root file system for AM37x (or Beagle XM, AM35x EVM, AM37x REV G EVM)

```
  $ make TARGET_PRODUCT= <product name> OMAPES=5.x -j8
```

```
  NOTE: product name can be beagleboard (For Beagleboard Rev Cx and
Beagleboard XM A/B/C)or omap3evm (For AM37x)or am3517evm(For AM35X)
  OMAPES variable controls the ES version of the device
  For AM35x                = 3.x
  For AM37x                = 5.x
  For Beagleboard Rev Cx   = 3.x
  For Beagleboard XM A/B/C = 5.x
```

The above command will build Android FS, kernel, SGX drivers and WLAN drivers (In case of AM37x). After successfull build, the kernel image can be found at kernel/arch/arm/uImage. Android rootfs components (root and system folders) will be located in out/target/ptoduct/omap3evm. SGX drivers and libraries are installed in Android rootfs components. WLAN drivers and firmware are installed in system component of the Android rootfs(For AM37x only).

Prepare the root filesystem:

```
  $ cd out/target/product/omap3evm
```

```
  $ mkdir android_rootfs
```

```
  $ cp -r root/* android_rootfs
```

```
  $ cp -r system android_rootfs
```

```
  $ sudo ../../../../build/tools/mktarball.sh
../../../host/linux-x86/bin/fs_get_stats android_rootfs . rootfs
rootfs.tar.bz2
```

The rootfs.tar.bz2 is the android filesystem, it can be put on a SD/MMC Card or used our NFS.

## RowboPERF Integration

- Download the sources from rowboat

```
   #> git clone -b rowboat-gingerbread
git://gitorious.org/rowboat/rowboperf.git
   #> cd rowboperf
```

- Refer the instructions in the README file for building and installing rowboperf components

NOTE: Clone rowboperf sources outside the rowboat-android tree.

- Refer to RowboPERF User Guide to know more.

## Adobe Flash 10 Integration

The Android version of Flash10 that runs on GingerBread is now available for customer download (by registration) at, http://focus.ti.com/docs/toolsw/folders/print/adobeflash-a8.html

The below steps give the procedure to download the Adobe Flash 10 library for Android GingerBread and installing the same in File system.

- Download the flashplayer installer “TI Flash10.3 Android Webkit Plugin-0.80-Linux-x86-Install.bin” from http://focus.ti.com/docs/toolsw/folders/print/adobeflash-a8.html
- Execute the installer

```
  $ ./TI Flash10.3 Android Webkit Plugin-0.80-Linux-x86-Install.bin
```

Will result in following instruction, press "Y"

```
  This will install Flash10.1 Android Webkit Plugin on your computer.
Continue? [n/Y] Y
  Select the source install location
  Where do you want to install Flash10.1 Android Webkit Plugin?
  [/home/user/flash10_androidplugin] /home/user/flash10_androidplugin
```

```
  Installing Flash10.1 Android Webkit Plugin...
  Installing Program
  Files...

  Installation complete.
  After Installation the following directory structure is resulted
```

- Change to Flash installed directory on Host PC

```
  $ cd flash10_androidplugin
  $ ls
  install_flash_player.apk  uninstall
```

- Install flash player plug in on target via adb. To set up adb tool for the host machine, follow the instructions at http://processors.wiki.ti.com/index.php/TI-Android-GingerBread-2.3.4-DevKit-2.1_DeveloperGuide#Using_ADB_Android_Debugger.2C_Downloader

```
  $ adb install install_flash_player.apk
```

- Do the browser configuration
  - Explained in the below section
- Test the Adobe Flash installation
  - Browse the link http://www.adobe.com/software/flash/about/
  - Should display Adobe Flash Player Successfully Installed

### To generate SD/MMC card to boot Android

These compiled Images can be copied to a SD / MMC card to boot Android on AM37x EVM The utility mk-bootscr from Tools package can be used to generate a boot script(boot.scr) for the evm to boot automatically.(Provided the NAND environment is empty)

- Generate boot.scr using mk-bootscr from tools folder
  - Edit mk-bootscr with the desired 'bootcmd' and 'bootargs'. For a reference see http://processors.wiki.ti.com/index.php/TI-Android-GingerBread-2.3-DevKit-1.0_UserGuide#Boot_arguments

```
$ ./mkbootscr
```

- Copy all the images to one folder

```
$ mkdir image_folder
$ cp kernel/arch/arm/boot/uImage image_folder
$ cp u-boot/u-boot.bin image_folder
$ cp x-loader/MLO image_folder
$ cp Tools/mk-bootscr/boot.scr image_folder
$ cp out/target/product/omap3evm/rootfs.tar.bz2 image_folder
$ cp media_clips image_folder
```

```
$ cp Tools/mk-mmc/mkmmc-android.sh image_folder
$ ./mkmmc-android <sd card mounted dev folder example:/dev/sdc>
MLO u-boot.bin uImage boot.scr rootfs.tar.bz2 Media_Clips
```

NOTE: mkmmc-android.sh and mk-bootscr are provided in tools folder in DevKit

## Hardware Setup and Requirements

This release of TI Android GingerBread 2.3.4 DevKit 2.1 is evaluated on the below given list of platforms. This package should be easily portable on other platforms on similar TI devices.

### Requirements

| TI Device | Platform Supported | Version | Other Accessories |
|---|---|---|---|
| OMAP35x | | | |
| | OMAP35x EVM [1] | Rev G | DVI Monitor, USB HUB, USB Keyboard, USB Mouse, Ethernet, UART Cable, Audio Speakers, MMC/SD Card (2GB min) |
| | Beagleboard [2] | Rev Cx | DVI Monitor, USB HUB, USB Keyboard, USB Mouse, Ethernet, UART Cable, Audio Speakers, MMC/SD Card (2GB min) |
| AM35x | | | |
| | AM3517 Evaluation Module [3] | Rev C | DVI Monitor, USB HUB, USB Keyboard, USB Mouse, Ethernet, UART Cable, Audio Speakers, MMC/SD Card (2GB min) |
| AM37x | | | |
| | AM37x Evaluation Module [4] | Rev C | DVI Monitor, USB HUB, USB Keyboard, USB Mouse, Ethernet, UART Cable, Audio Speakers, MMC/SD Card (2GB min) |
| | BeagleBoard [2] | XM Rev A/B | DVI Monitor, USB HUB, USB Keyboard, USB Mouse, Ethernet, UART Cable, Audio Speakers, MMC/SD Card (2GB min) |
| DM37x | | | |

### Setup

- EVM

This section describes the setup and instructions to run TI's Android DevKit OOB demo on AM37x/DM3730 EVM.

Step 1: Insert SD card into MMC/SD slot on the EVM

Step 2: Set the DIP switch settings to boot from MMC/SD card

AM37x EVM the DIP switch SW4 should be set as shown below

| Switch | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| State | OFF | ON | ON | ON | OFF | OFF | OFF | OFF |

AM35x EVM the DIP switch S7 should be set as shown below

| Switch | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| State | ON | OFF | OFF | ON | OFF | OFF | OFF | ON |

Step 3: Turn ON the EVM

Refer to UserGuide and RowboPERF user guide from document folder in DevKit release

- Beagle XM

Step 1: Insert SD card into MMC/SD slot on the Beagleboard

Step 2: Turn ON the Beagleboard

Refer to UserGuide and RowboPERF user guide from document folder in DevKit release

## Application Development and Debugging

### Using TI's Code Composer Studio V5 (Eclipse based)

Code Composer Studio (CCS) is the integrated development environment for TI's DSPs, microcontrollers and application processors based on the Eclipse open source software framework which includes a suite of tools used to develop and debug embedded applications. Since CCS is based on Eclipse, it is possible to integrate the Android Development Tools (ADT) like Android Debug Bridge (ADB), Dalvik Debug Monitoring System (DDMS) and ndk-gdb in CCS to enable the debugging of Android Applications directly on Android Device (i.e TI EVM) along with the inherent CCS capability of Linux Aware Debug and DSP Debugging.

This wiki walks you through installation of Android Debugging Tools in CCS, connecting CCS with Android Device and illustrates a debugging session of an HelloWorld Application based on android NDK having a mix of Java and Native C Code.

- Pre-requisite Installations :
  - CCSv5 (or Eclipse Helios) - http://processors.wiki.ti.com/index.php/Category:Code_Composer_Studio_v5
  - Android SDK - http://developer.android.com/sdk/index.html
  - Android NDK - http://developer.android.com/sdk/ndk/index.html
  - Cygwin - http://cygwin.com/install.html ( For Windows Host)
  - Android Device running Android 2.3.4 Gingerbread – www.arowboat.com

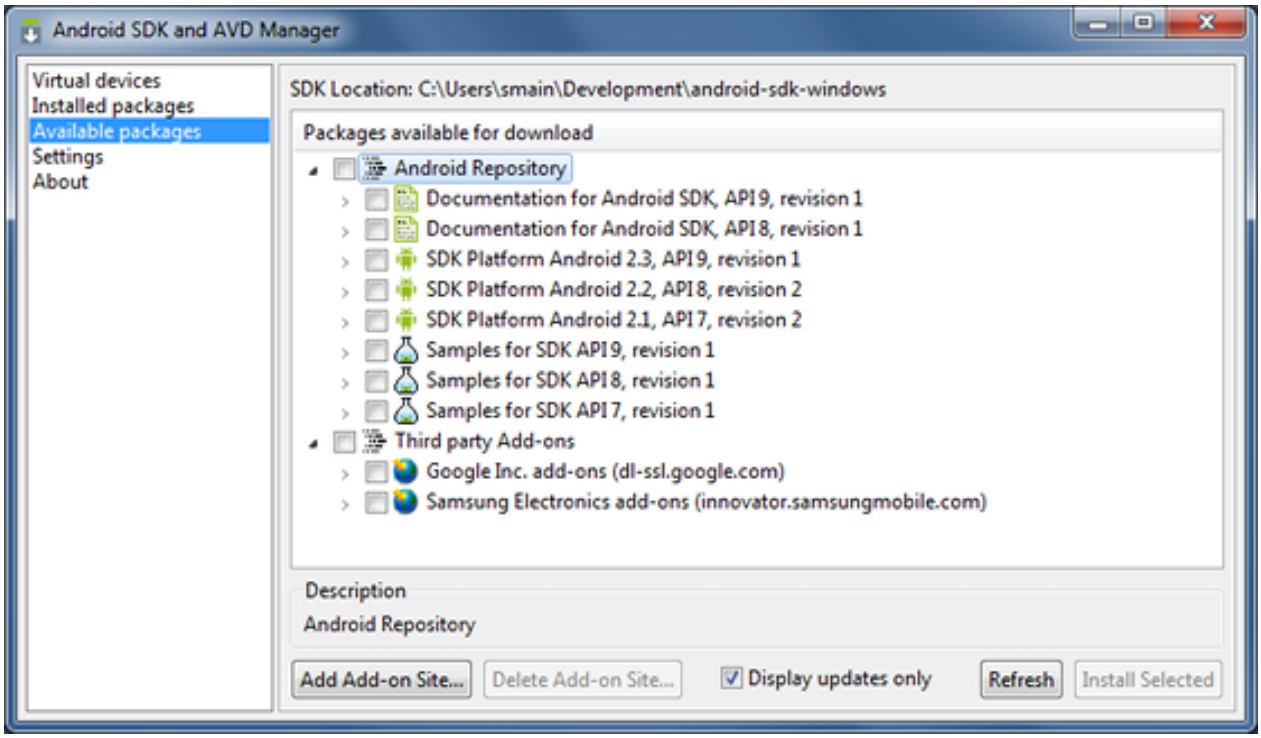
## Preparing CCS for Android Development

Note: The steps below are condensed version of steps mentioned on http://developer.android.com/sdk/installing.html

1. Installing the ADT Plugin in CCS Refer http://developer.android.com/sdk/eclipse-adt.html#installing

2. Adding Platforms and Other Components

- On Windows, double-click the SDK Manager.exe file at the root of the Android SDK directory
- On Linux, run '<SDK>tools/android'
- Proxy Settings (If needed): Go to Settings and set the Proxy to work with your network
- To download components, use the graphical UI of the Android SDK and AVD Manager, shown in Figure below, to browse the SDK repository and select new or updated components. The Android SDK and AVD Manager will install the selected components in your SDK environment. Refer the list of Recommended Components: http://developer.android.com/sdk/installing.html#which




- To Install the Terminal (similar to Teraterm) in CCS, please follow the steps mentioned in the wiki: http://processors.wiki.ti.com/index.php/How_to_install_the_terminal_plugin_in_CCSv5

## Setup debugging via ADB over network

Step 1: Upon installing the ADT plugin, DDMS should have been setup. DDMS configuration should be changed as in below:

```
Click on Window->Preferences; Select Android -> DDMS
Change - ADB debugger base port: 8700; Logging Level: Verbose
Click on Apply
```

Step 2: DDMS perspective can now be opened from the eclipse menu via:

```
Window -> Open Perspective -> Other -> DDMS;
Click on OK
```

Step 3: Get CCS to attach to your TI EVM:

Things to do on Android Device: Obtain IP of your EVM via the executing following command on your EVM's Serial Terminal:

```
      # netcfg
```

ADB stub on target defaults to USB. To fix this, execute the following on your EVM's Serial Terminal :

```
# setprop service.adb.tcp.port 5555
# stop adbd
# start adbd
```

On the Host machine run the following commands from cmd prompt:

```
 $ export ADBHOST=<IP_ADDRESS_OF_YOUR_TI_EVM> (Linux)
 $ set ADBHOST=<IP_ADDRESS_OF_YOUR_TI_EVM> (Windows)
 $ cd <ANDROID_SDK_ROOT>\platform-tools
 $ adb kill-server
 $ adb start-server
 $ adb connect <target_ip_address>:5555
```

Check if you are now connected to the TI EVM device by running the following command on the cmd prompt:

```
 $ adb devices
```

It should output something like:

```
 emulator-5554 device
```

This confirms that EVM is connected. With this setup, you should be able to use ADB, logcat, DDMS and other tools directly from CCS ADT environment for creating your applications for Android.

Note: you can refer the following for more details: http://developer.android.com/guide/developing/device.html http://www.omappedia.org/wiki/Android_Debugging#Debugging_on_Zoom2_with_Eclipse_ADT

## Debugging Android with CCS

### Hello World Application on Android Virtual Device (AVD)

http:/ / processors. wiki. ti. com/ index. php/ TI-Android-FroYo-DevKit-V2_CCSv5SetupGuide#Hello_World_Using_CCSv5
http://developer.android.com/resources/tutorials/hello-world.html

### Hello-jni of Android NDK on Android Device

- In CCS:

1. Click File > New Android Project...
2. Select the Create project from existing source radio button.
3. Select any API level above Android 1.5.
4. In the Location field, click Browse... and select the <ndk-root>/samples/hello-jni directory.
5. Click Finish.
6. Go to C/C++ Perspective. Click File->New->Convert to C/C++ Project

- Edit jni/Android.mk and add the following line before BUILD_SHARED_LIBRARY: APP_CFLAGS := -g. This will add debugging symbols in to your native C code.
- Compile the native code using the ndk-build command from cmd prompt:

```
cd <ndk-root>/samples/hello-jni
<ndk_root>/ndk-build
```

- In CCS, create a debug configuration for a C/C++ application:
  - Create a new debug configuration for a C/C++ application. Click Run->Debug Configurations.
  - Set the process launcher to “Standard Create Process Launcher”
  - On “Main” tab, set:
    C/C++ Application: android-ndk-r5-windows\android-ndk-r5\samples\hello-jni\obj\local\armeabi\app_process
    Select Disable auto build
  - On “Debugger” tab, set:
    - Debugger: gdbserver
    - Stop on startup at: Java_com_example_hellojni_HelloJni_stringFromJNI (It is the entry function of the native C code)
    - Main Tab:
      - GDB debugger:
        android-ndk-r5-windows\android-ndk-r5\toolchains\arm-eabi-4.4.0\prebuilt\windows\bin\arm-eabi-gdb.exe
      - GDB command file: android-ndk-r5-windows\android-ndk-r5\samples\hello-jni\libs\armeabi\gdb.setup
      - GDB command set: Standard
      - Protocol: mi
      - Select only Verbose console mode
    - Connection Tab:
      - Type: TCP
      - Host name or IP address: localhost
      - Port number: 5039
- Open the ndk-gdb script that came with the android NDK and comment the last line (we are not calling the usual gdb client, but we will attach an Eclipse gdb session instead)

```
# $GDBCLIENT -x $GDBSETUP -e $APP_PROCESS
```

- Only For Windows: Run dos2unix.exe on ndk-gdb from cygwin after editing to make sure that we do not have any unwanted dos symbols in the script
- Starting Java Debugger:
  - Make sure you are in Java Perspective.
  - Click Run->Debug Configurations.
  - Place a breakpoint in Java code just before the native code call
  - Select Android Debug Configuration instance “HelloJni” and click “Debug”
  - It will prompt you to switch and open to CCS Debug Perspective. Select “yes”
- Starting GDB Debugger:
  - In cygwin, make sure that the root of android-ndk is in PATH.
  - In cygwin, Go to /android-ndk-r5-windows/android-ndk-r5/samples/hello-jni and execute the following and wait until you get a prompt again:

```
ndk-gdb
–adb=<PATH_TO_android-sdk-windows/platform_tools/adb.exe>

sample command:
$ ndk-gdb
--adb=/cygdrive/c/PROGRA~1/Android/android-sdk-windows/platform-tools/adb.exe
```

- CCS, go to C/C++ Perspective and click Run->“Debug”. This will launch the gdbserver in CCS Debug Perspective.
- Now, you can seamlessly debug between Java and Native C code

## Using ADB Android Debugger, Downloader

Android Debug Bridge (adb) is a versatile tool lets you manage the state of the Android-powered device. For more information about what is possible with adb, see Android Debug Bridge page at http://developer.android.com/guide/developing/tools/adb.html.The ADB tool can be used to

- Download an application from a host machine, install & run it on the target board.
- Start a remote shell in the target instance.
- Debug applications running on the device using the debugging tool DDMS ( Dalvik Debug Monitor Server) which runs on top of adb connection.
- Copy files to and from the board to host machine

### Downloading "ADB" & Host setup

The adb tool is a part of Android SDK package located at http://developer.android.com/sdk/index.html.For an overview of how to install and set up the Android SDK, follow download & setup instructions from http://developer.android.com/sdk/index.html.Once you install Android SDK, the directory contents look like this.

```
  add-ons/
  docs/
  platforms/
    <platform>/
       data/
       images/
       skins/
       templates/
       tools/
       android.jar
  samples/
  tools/
SDK Readme.txt
```

The adb tool is located in tools/ directory under the Android SDK installation. Export the tools directory path as shown below.

```
  $ export PATH=${PATH}:<your_sdk_dir>/tools
```

### Connecting Host machine to board through adb

This release of DevKit has been tested for three different methods of connecting a given board with host machine

- adb over USB
- adb over USB Ethernet
- adb over Ethernet

The below sections describe each of these methods and provides necessary instructions for the same.

**adb over USB**

- Make sure that the mini-usb cable is connected between the host usb port and the target's USB OTG port
- Turn on "USB Debugging" on your board. On the board (UI screen)-
  - Go to home screen, press MENU,
  - Select Applications, select Development, then enable USB debugging.
  - Alternatively, you can navigate to Settings->Applications->Development and then enable the "USB debugging" option.
- Setup host machine to detect the board. On Ubuntu Linux host machines this is done by adding a rules file to configure device vendor ID of on-board OMAP device.
- For the EVMs and Boards covered here, the vendor ID is "18d1".
  - Log in as root and create this file: **/etc/udev/rules.d/51-android.rules**

```
    For Gusty/Hardy, edit the file to read:
    SUBSYSTEM=="usb", SYSFS{idVendor}=="18d1", MODE="0666"
```

```
    For Dapper, edit the file to read:
    SUBSYSTEM=="usb_device", SYSFS{idVendor}=="18d1", MODE="0666"
```

  - Execute the following to change the user mode for the rules file.

```
    $ chmod a+r /etc/udev/rules.d/51-android.rules
```

  - Verify the adb connectivity between host and target board

```
    $ adb devices
```

```
    If device is connected, then output on screen should list the
device, example:

       List of devices attached
       20100720    device
```

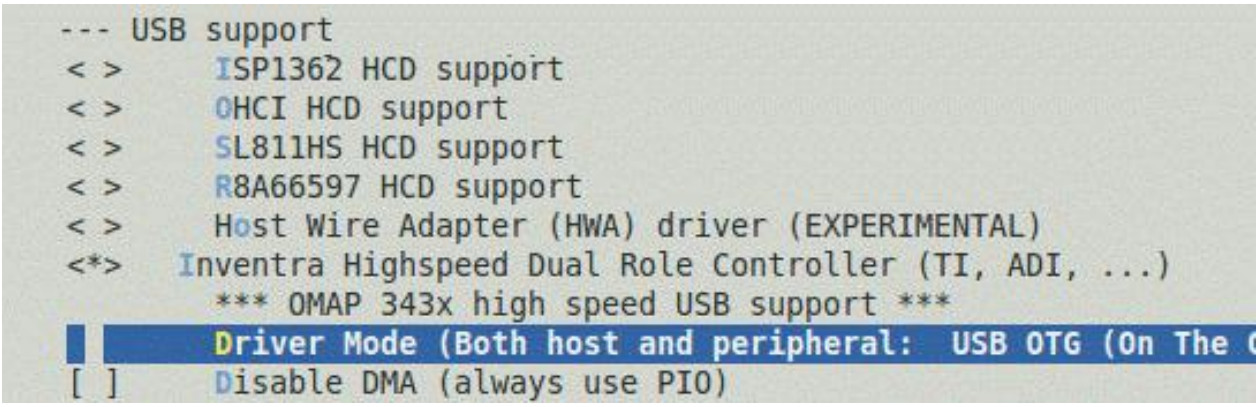
**adb over USB Ethernet (Ethernet over USB)**

- Make sure that the mini-usb cable is connected between the host usb port and the target's USB OTG port.
- Configure the Linux kernel to use Ethernet gadget. Enable USB support, configure the Inventra controller, and add USB gadget support.

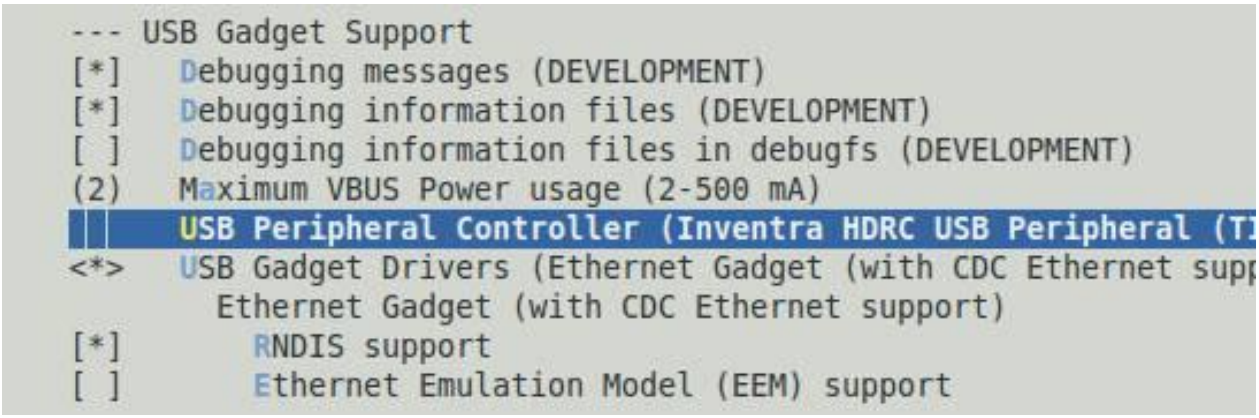
IMPORTANT NOTE: Inventra configuration must occur in two places as shown in non-highlighted lines of the screen shots below.

```
    $ make ARCH=arm CROSS_COMPILE=arm-eabi- menuconfig
```

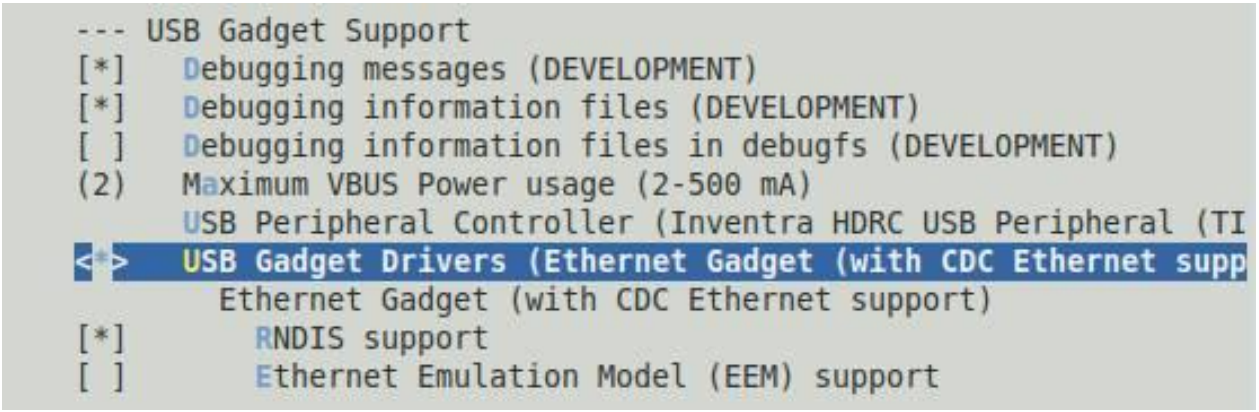
Device Drivers --- USB Support




Device Drivers --- USB Support --- USB Gadget Support




Device Drivers --- USB Support --- USB Gadget Support --- Enable Gadget Ethernet support




- Build the Kernel with the above configuration changes and use the uImage to boot the board. Refer to Kernel compiling instructions above.
- Establish network connection
  - Assign an IP address to the usb ethernet adapter.

The USB network gadget g_ether is named usb0 (instead of eth0 or other network interface names). The normal set of Ethernet configuration tools should work, such as ifconfig, netstat, and route.

For example, the following commands will assign the network address 192.168.194.2 to the target. Run this on the target:

```
$ ifconfig usb0 192.168.194.2 netmask 255.255.255.224 up
```

On Host machine, run the following commands to establish the connection to the target:

```
$ sudo ifconfig usb0 192.168.194.1 netmask 255.255.255.224 up
$ sudo route add 192.168.194.2 dev usb0
```

The target and the host machine should be connected, run ping command to test the same:

```
$ ping -c 3 192.168.194.2
PING 192.168.194.2 (192.168.194.2) 56(84) bytes of data.
64 bytes from 192.168.194.2: icmp_seq=1 ttl=64 time=6.08 ms
64 bytes from 192.168.194.2: icmp_seq=2 ttl=64 time=0.511 ms
64 bytes from 192.168.194.2: icmp_seq=3 ttl=64 time=0.485 ms
--- 192.168.194.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.485/2.361/6.089/2.636 ms
```

- Establish ADB connection

On the host machine execute following commands to establish adb connection

```
$ export ADBHOST=<target's ip address>
$ adb kill-server
$ adb start-server
$ adb connect <target_ip_address>:5555
```

Verify the connection by executing

```
$ adb devices
```

If connected, device name should be listed as a "emulator"

```
$ adb devices
List of devices attached
emulator-5554    device
$ adb shell
```

**adb over Ethernet**

- Make sure Ethernet port on board and host machine are connected to the network
- Check Ethernet configuration for the board

```
target #> netcfg

   lo       UP    127.0.0.1       255.0.0.0       0x00000049

   eth0     UP    172.24.190.59   255.255.252.0   0x00001043
```

- If Ethernet was not configured, configure Ethernet of the board using ifconfig/netcfg as shown below.

```
target #> netcfg eth0 dhcp
```

- Configure the ADB Daemon to use an ethernet connection using setprop as shown below.

```
target #> setprop service.adb.tcp.port 5555
```

- If network is configured successfully (above steps) then Restart service adbd on the target,

```
target #> stop adbd
target #> start adbd
```

- On the host machine use following commands to establish adb connection

```
    $> export ADBHOST=<target's ip address>
    $> adb kill-server
    $> adb start-server
    $> adb connect <target_ip_address>:5555
```

- Verify for device connectivity, by executing the following commands

```
    $> adb devices If connected, you'll see the device name listed
as a "emulator"
```

```
    $> adb devices
```

```
    If connected, find the device name listed as a "emulator"
```

```
    List of devices attached
    emulator-5554    device
    $ adb shell
```

For more information about adb commands, see Android Debug Bridge page at http://developer.android.com/guide/developing/tools/adb.html

**adb over USB on Windows Machine**

Follow the below instructions to get ADB over USB work on a Windows PC

- Download latest Android SDK

(http://developer.android.com/sdk/index.html) and uncompress it in a local folder (i.e. c:\android_sdk).

- Optionally, you may want to add the location of the SDK's primary tools directory to your system PATH. Right-click on My Computer, and select Properties. Under the Advanced tab, hit the Environment Variables button, and in the dialog that comes up, double-click on Path (under System Variables). Add the full path to the tools\ directory to the path.
- Download Android USB Driver

(https://dl-ssl.google.com/android/repository/usb_driver_r03-windows.zip) and uncompress it in a local folder (i.e. c:\android_sdk\usb_driver)

- Edit (or create and then edit if it doesn't already exist) file in

"%USERPROFILE%\.android\adb_usb.ini":

```
  echo 0x18D1 > "%USERPROFILE%\.android\adb_usb.ini"
```

- Edit android_winusb.inf to match EVM/Beagle vendor and product ids:

Under [Google.NTx86] section add:

```
  ;TI EVM
  %SingleAdbInterface%        = USB_Install, USB\VID_18D1&PID_9018
  %CompositeAdbInterface%     = USB_Install,
USB\VID_18D1&PID_9018&MI_01
```

Note: Be careful to add it under Google.NTx86 and not under Google.NTamd64 unless your machine is AMD 64 bits. If you skip this step you won't be able to later install the driver as windows will reject it.

- Boot the board as normal and wait until shell prompt is available (micro-B USB cable must be disconnected).
- Connect micro-B USB cable between board and Windows PC.

- If it is proceeding as planned, Windows will tell you it found a new hardware asks you to install the driver. Install driver that was downloaded as described in step 3 above:

Answer "No, not this time" to the question about running Windows Update to search for software.

- Choose "Install the hardware that I manually select from a list (Advanced)" this is the 2nd option, then click "Next"
- Select "Show All Devices", then click "Next"
- You are going to see a grayed-out text box with "(Retrieving a list of all devices)", click the "Have Disk..." button
- Browse" to your driver folder (c:\android_sdk\usb_driver). It will be looking of a .inf file so select "android_winusb.inf" and click "Open" then "OK". It's the only file there so you shouldn't go wrong.
- Select "Android ADB Interface" then click the "Next" button.
- A warning will appear, answer "Yes" but read the warning anyway.
- Click the "Close" when the wizard is completed.

- Disconnect and reconnect micro-B USB cable from Board(probably reboot it as well).
- Open command prompt and restart adb server just to make sure it is in a proper state:

```
adb kill-server
adb start-server
```

- List the attached devices with "adb devices". It should show your board/device with a random number.
- Type "adb shell". You should see the "#" indicating it works.

## Running Applications

The root File System provided in this DevKit releases contains only standard Android components and applications. User might be interested to download & run android applications (.apk) available in the market. The below procedure gives the steps to be followed to download any .apk file to the board and run it on the platform.

#### Installing (.apk files) application on Target Platform

- From the host: You can use adb tool for package installation.

```
   $> adb install <package>.apk.
```

NOTE: Use -s option with the adb tool, to install the package on external storage.

On successful installation adb tool will report SUCCESS on host terminal, and the application would be listed on the android main menu.

#### Un-installing applications (.apk) using adb

- To un-install non-default components (that were installed later)
  - Method 1: On the host machine execute the following

```
   $> adb uninstall <package>.apk
```

  - Method 2: On target:

```
   Main menu -> Menu -> Settings -> Applications -> Manage
applications -> Find the package
   Tap on it -> Uninstall -> OK -> OK
```

  - On successful removal, the application would have been removed from the android main menu. All the short-cuts to the application also removed.

- To un-install default components, use the following commands from abd on host machine

```
$ adb shell
#rm /system/app/app.apk
```

On successful removal, the application would have been removed from the android main menu.

## Setup ADB for application Debugging

ADB and Eclipse, with ADT( Android Development Tools plug-in) allow users to create and debug Android applications. Follow Developing In Eclipse, with ADT at http:// developer. android. com/ guide/ developing/ eclipse-adt.html

Steps to connect Eclipse to the board.

- Setup the adb connection with the board by following the instructions given above in connecting board ...

```
Verify the connectivity by executing
$ adb devices
```

- Open Eclipse IDE. Eclipse, with ADT plugin enable users to
  - Create an android project.
  - Build and Run the project on a connected board.
  - Debug the project using the Debug perspective.
  - Use DDMS (Dalvik Debug Monitor Server) to monitor the connected board.

For more detailed and complete information on the above follow Developing In Eclipse, with ADT at http:// developer.android.com/guide/developing/eclipse-adt.html

- Open DDMS(Dalvik Debug Monitor Server) perspective. This DDMS perspective can be opened from the eclipse menu via:

```
Window -> Open Perspective -> Other -> DDMS;
Click on OK
```

- DDMS provides port-forwarding services, screen capture on the device, thread and heap information on the device, logcat, process, and radio state information,incoming call and SMS spoofing, location data spoofing, and more.
- For more information on DDMS and to use it, follow Using the Dalvik Debug Monitor page at http://developer. android.com/guide/developing/tools/ddms.html

## Copy any files to and from the board over ADB

- Using the adb commands "pull" and "push" user can copy files to and from the board.
- Unlike the install command, which only copies an .apk file to a specific location, the pull and push commands let you copy arbitrary directories and files to any location on the board.
- To copy a file or directory (recursively) from the board, use

```
adb pull <remote> <local>
```

- To copy a file or directory (recursively) to the board, use

```
adb push <local> <remote>
```

In the commands, <local> and <remote> refer to the paths to the file or directory on your development host (local) and on the target instance (remote).

```
    Here's an example:
    adb push foo.txt /sdcard/foo.txt
```

## Acronyms

- SDK – Software Development Kit
- NDK – Native Development Kit (For more information, refer http://developer.android.com/sdk/ndk/overview.html)
- ADT – Android Development Tools
- ADB – Android Debug Bridge
- DDMS – Dalvik Debug Monitoring System
- AVD – Android Virtual Device

## Quick References

| Content | Link Ref# |
|---|---|
| User Guide | [5] |
| Release Notes | [6] |
| Porting Guides | [7] |
| Downloads | [8] |

## Support

For further information or to report any problems, contact http://e2e.ti.com/android or http://support.ti.com.
For community support join http://groups.google.com/group/rowboat
For IRC #rowboat on irc.freenode.net

## References

[1] http://focus.ti.com/docs/toolsw/folders/print/tmdsevm3530.html
[2] http://beagleboard.org
[3] http://focus.ti.com/docs/toolsw/folders/print/tmdxevm3517.html?DCMP=dsps_arm_102109&HQS=Other+OT+sitara-prtools
[4] http://focus.ti.com/docs/toolsw/folders/print/tmdxevm3715.html?DCMP=am37x_060710&HQS=Other+OT+am37xprtf
[5] http://processors.wiki.ti.com/index.php/TI-Android-GingerBread-2.3.4-DevKit-2.1_UserGuide
[6] http://processors.wiki.ti.com/index.php/TI-Android-GingerBread-2.3.4-DevKit-2.1_ReleaseNotes
[7] http://processors.wiki.ti.com/index.php/TI-Android-GingerBread-2.3.4-DevKit-2.1_PortingGuides
[8] http://focus.ti.com/docs/toolsw/folders/print/androidsdk-sitara.html

# Article Sources and Contributors

**TI-Android-GingerBread-2.3.4-DevKit-2.1 DeveloperGuide** *Source*: http://processors.wiki.ti.com/index.php?oldid=69098 *Contributors*: Arunjoseph05, Pankaj.bharadiya, Pundiramit

# Image Sources, Licenses and Contributors

**Image:TIBanner.png** *Source*: http://processors.wiki.ti.com/index.php?title=File:TIBanner.png *License*: unknown *Contributors*: Nsnehaprabha

**File:Android-sdk-manager.png** *Source*: http://processors.wiki.ti.com/index.php?title=File:Android-sdk-manager.png *License*: unknown *Contributors*: Satish npatel

**Image:Android USB ADB ENABLE.JPG** *Source*: http://processors.wiki.ti.com/index.php?title=File:Android_USB_ADB_ENABLE.JPG *License*: unknown *Contributors*: Khasim

**Image:Android USBGadget ADB.JPG** *Source*: http://processors.wiki.ti.com/index.php?title=File:Android_USBGadget_ADB.JPG *License*: unknown *Contributors*: Khasim

**Image:Android USBEthernet ADB.JPG** *Source*: http://processors.wiki.ti.com/index.php?title=File:Android_USBEthernet_ADB.JPG *License*: unknown *Contributors*: Khasim

# License

### 7. Termination

1. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Adaptations or Collections from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.
2. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

### 8. Miscellaneous

1. Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.
2. Each time You Distribute or Publicly Perform an Adaptation, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.
3. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.
4. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.
5. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.
6. The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.