

# Intel® Application Debugger 2.3 for Intel® Atom™ Processor Installation Guide and Release Notes

---

Installation Guide and Release Notes

Document number: 322072-009US

8 November 2011

## Contents:

[Introduction](#)

[What's New](#)

[System Requirements](#)

[Installation Notes](#)

[Key Features](#)

[Usage Notes](#)

[Issues and Limitations](#)

[Attributions](#)

[Disclaimer and Legal Information](#)

## 1 Introduction

This package contains the Intel(R) Application Debugger 2.3 for Intel(R) Atom(TM) processor.

It is a Linux\* hosted Eclipse\* RCP based cross-debug solution for Linux\* software developers to debug C and C++ applications via a remote TCP-IP debug handler. It works with both networked devices running MeeGo\* or Linux\* as well as QEMU based virtual machines. The Intel® Debugger is designed to attach to an application that has been launched locally on the physical or virtual device or that has been launched remotely. Furthermore it can deploy and launch an application on the target. The automatic application deployment does however only apply for the application binary itself. If there are dependencies on shared object files, those need to be copied manually.

In addition the Intel® Application Debugger 2.3 for Intel® Atom™ Processor offers the full range of in depth threading awareness, thread level breakpoints, and run control with thread grouping and thread freezing and thawing.

The offering for application developers is completed through a set of features like

- Hardware watchpoint support

- Remote module download and target debuggee process connect
- Remote view of target application threads
- Advanced OS signal handling

The developer thus gains the extra level of visibility into the system they develop on they want. This minimizes the time it takes to isolate and correct application level problems by understanding the system environment the application runs on.

Information on Intel Software Development Products is available at <http://www.intel.com/software/products/>.

This document provides system requirements, installation instructions, issues and limitations, and legal information.

## Technical Support and Documentation

To submit issues related to this product please visit the [Intel Premier Support](#) webpage and submit issues under the product **Intel(R) Embedded SW Dev Tools Atom**.

For information on how to register for and purchase support for the Intel(R) Embedded Software Development Tool Suite Intel(R) Atom(TM) processor please visit the [Intel\(R\) Software Development Products](#) webpage.

Additional information on the Intel® Embedded Software Development Tool Suite for Intel® Atom™ Processor is available at the following web resources:

- Product Page: <http://www.intel.com/software/products/atomtools>
- User Forum: <http://software.intel.com/en-us/forums/software-development-toolsuite-atom/>
- Knowledge Base Articles: <http://software.intel.com/en-us/articles/software-development-toolsuite-atom-kb/all/1/>

Please remember to register your product at <https://registrationcenter.intel.com/> by providing your email address. This helps Intel recognize you as a valued customer in the support forum.

For information about how to find Technical Support, product documentation and samples, please visit <http://www.intel.com/software/products/atomtools>

## Product Contents

- Intel® Application Debugger 2.3 for Intel® Atom™ Processor, Build [76.xxx.x]
- Intel® Debugger Remote Server for IA32 Linux\*, Build [76.xxx.x]

## 2 What's New

### 2.1.1 Updated Graphical User Interface

The button icons have been updated and the overall look and feel of the debugger has been modernized

### 2.1.2 Support for Yocto Project\* 1.1 targeted application debug

The Intel® Application Debugger for Intel® Atom™ Processor has been validated against Yocto Project\* 1.1 and Application Development Toolkit 1.1 compatibility

### 2.1.3 Thread Specific Run-Control and Thread Grouping

The debugger now supports thread grouping and thread specific run-control including the setting of thread specific breakpoints.

### 2.1.4 Extended Breakpoints Feature

With this feature you can set breakpoints on routines in shared libraries which have not yet been loaded. The requested breakpoint will be realized whenever possible. You'll see unrealized breakpoints marked with a yellow triangle (not having an address, file and symbol name) in the GUI. On the command line those are marked as <PENDING>. Any ambiguity is directly resolved and you will get multiple realizations, e.g. requesting a breakpoint for an overloaded function. In the GUI, those are visualized as a tree with the requesting breakpoint as its node. On the command line the requesting breakpoint is marked as <MULTIPLE> and its realizations follow.

Please note that for the command line this feature is only available in GDB mode.

### 2.1.5 Command solib-search-path now implemented

The command line debugger idbc and the Command window of the GUI debugger now support the existing gdb command solib-search-path which is used to look up images or shared libraries when they have not been found in the usual places such as \$LD\_LIBRARY\_PATH.

Please invoke the command line help to see the solib-search-path command usage:

```
(idb) help set solib-search-path
(idb) help show solib-search-path
```

or the abbreviated commands:

```
(idb) h set sol
(idb) h sho sol
```

### 2.1.6 Qt C++ Class Awareness

The debugger is able to resolve Qt C++ classes such that when monitoring variables in such classes they will be resolved correctly and their value will be directly displayed without having to follow the expanded C++ class treeview.

## 3 System Requirements

### 3.1.1 Host Software Requirements

- Linux\* system running Fedora\* 14, Fedora\* 15 or Ubuntu\* 11.04.
- Fedora\* 10 has only been partly validated
- Java runtime environment (JRE) 1.5 or 1.6 (also known as 6.0) to use the Eclipse\* framework. In a web browser, access [www.java.com](http://www.java.com), and download and install JRE 1.6. Make sure that the \$PATH environment variable contains the path to the JRE bin-directory.
- A successful install requires the following standard packages to be present: g++, gcc, libc6, libstdc++6, binutils

### 3.1.2 Target Software Requirements

The target platform should be based on one of the following environments:

- Yocto Project\* 1.1 based environment
- MeeGo\* 1.x based environment
- MeeGo\* compliant OS

### 3.1.3 Hardware Requirements

- Intel® IA-32 architecture based host computer supporting Intel® Streaming SIMD Extensions 2 (Intel® SSE2) instructions (Intel® Pentium® 4 processor or later), or compatible non-Intel processor.
  - For the best experience, a multi-core or multi-processor system is recommended.
- 1GB RAM (2GB recommended)
- 4GB free disk space for all product features and all architectures
  
- Ethernet TCP/IP Connection and ethernet cable
- Development platform based on the Intel® Atom™ processor Z5xx, Z6xx, N2xx, N3xx, N4xx, D4xx, D5xx, Intel® Atom™ processor CE4100, CE4200 or the Intel® Media processor CE3100

## 4 Installation Notes

### Installing the Debugger

The default installation directory is `/opt/intel/atom/idb/2.3.xxx`

The Intel® Application Debugger for Intel® Atom™ Processor is intended for Intel® Atom™ processor targeted cross-development. It is recommended to install the debugger on your software development host system. It is further recommended to install the idbserver debug server in one of the following locations:

- Intel® Atom™ processor based target device
- MeeGo\* Image Creator 2 based jailroot environment
- MeeGo\* 1.x virtual image running inside QEMU\*
- Yocto\* 1.x virtual image running inside QEMU\*

For installation of the debugger update on the development host please follow the steps below:

1. Unpack the tool suite package in a directory to which you have write access.  
> `tar -zxvf l_MID_DBG_p_2.3.xxx.tgz`
2. If you do not have a license file, please note the product serial number. You will need it to complete the installation process. Otherwise copy the license file you may have received via email from the Intel® Software Development Products Registration Center to `/opt/intel/licenses/`.
3. It is recommended to register your product at <https://registrationcenter.intel.com>. If you purchased support for this product you will need to register to take full advantage of Intel Premier Support at <https://premier.intel.com>.
4. Change into the directory the tar file was extracted to `../l_MID_DBG_p_2.3.xxx`
5. Run the installation script

Execute the install script in the directory where the tar file was extracted.

```
>./install.sh
```

6. If you are not logged in as root, you will be asked if you want to install as root, install as root using `sudo`, or install without root privileges. Installing as root (using `sudo` if you have that privilege) is recommended, as that will update the system RPM database. Use the `install as current user` option if you want to install to a private area.
7. The welcome message to the Intel® Embedded Software Development Tool Suite 2.3 for Intel® Atom™ Processor appears along with an outline of the installation process. Press the `Enter` key to continue.

8. The installation routine checks for the availability of all product dependencies. Please take care of these dependencies, if a warning message appears.
9. Afterwards you will be asked to read the end-user license agreement for the tool suite. Press the `Enter` key to continue with reading the license agreement. Once done type `accept` to continue with the installation.
10. When asked whether you would like to activate and install your product select one of the options provided depending on whether you have a license file available or not. If there is already a valid license file available and installed on your system, the installation routine will recommend to simply use the existing license file. If you do not have access to the internet at the time of installation, select the alternative activation option.
11. The next screen let's you review your installation options. If you would like to only install only the Intel® Application Debugger and not the Intel® JTAG Debugger as well, select `[3]` and change the components settings. Else, continue with the default choice `[1]` to start the installation.

Step no: 4 of 6 | Options

---

You are now ready to begin installation. You can use all default installation settings by simply choosing the "Start installation Now" option or you can customize these settings by selecting any of the change options given below first. You can view a summary of the settings by selecting "Show pre-install summary".

---

1. Start installation Now
  2. Change install directory [ /opt/intel ]
  3. Change components to install [ All ]
  4. Show pre-install summary
  - h. Help
  - b. Back to the previous menu
  - q. Quit
- 

Please type a selection or press "Enter" to accept default choice `[1]`:

12. During the prerequisites check the presence of the Yocto Project\* Application Development Toolkit 1.1 (ADT) will be checked. If it is not present on your installation you will get a warning message.

The Yocto Project\* Application Development Toolkit is required if you want to use the Intel® Composer XE for building Yocto Project\* targeted applications. For automatic Intel® Composer XE integration with the Application Development Toolkit during installation, please install Toolkit first and re-check the prerequisites.

If you do not intend to develop applications targeting Yocto Project\* builds, you can ignore this warning message.

13. The installation will be finalized and you will be informed when the installation has been completed successfully.

Step no: 6 of 6 | Complete

---

Thank you for installing and for using the Intel(R) Embedded Software Development Tool Suite for Intel(R) Atom(TM) processor version 2.3.  
Support services start from the time you install or activate your product. If you have not already done so, please create your support account now to take full advantage of your product purchase.  
Your support account gives you access to free product updates and upgrades as well as interactive technical support at Intel(R) Premier Support.  
To create your support account, please visit the Subscription Services web site  
<https://registrationcenter.intel.com/RegCenter/registerexpress.aspx?media=WJP>

---

q. Quit [default]

---

Please type a selection or press "Enter" to accept default choice [q]:

## Installing the Debugger Remote Server

To install the Intel® Debugger Remote Server for IA32 Linux\* go its installation directory in the Intel® Application Debugger 2.3 for Intel® Atom™ Processor installation:  
`/opt/intel/atom/idb/2.3.xxx/bin/ia32.`

Copy the file `idbserver` onto your cross-debug target device or target virtual machine via sftp.

## Uninstalling the Debugger

To uninstall the Intel® Application Debugger, change into the `/opt/intel/atom/idb/2.3.xxx/bin` directory and run the `uninstall_dbg.sh` script.

## 5 Key Features

### 5.1.1 Updated Graphical User Interface

The button icons have been updated and the overall look and feel of the debugger has been modernized

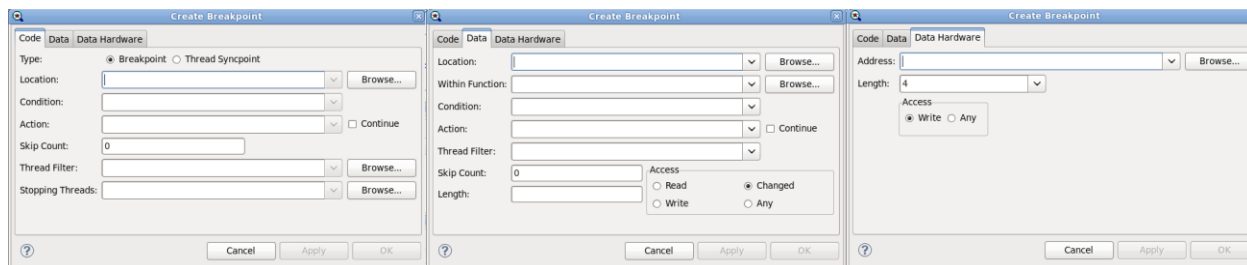
### 5.1.2 Support for Yocto Project\* 1.1 targeted application debug

The Intel® Application Debugger for Intel® Atom™ Processor has been validated against Yocto Project\* 1.1 and Application Development Toolkit 1.1 compatibility


### 5.1.3 Thread Specific Run-Control and Thread Grouping

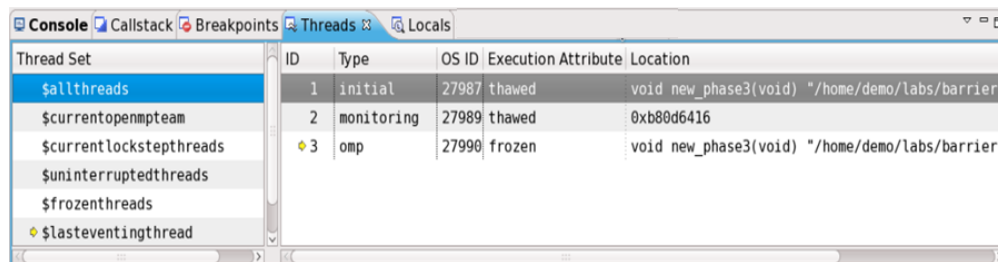
The debugger now supports thread grouping and thread specific run-control including the setting of thread specific breakpoints.

In general when you stop (i.e. suspend) or hit a breakpoint all threads are stopped and when you continue all threads will continue. In the Intel® Application Debugger for Intel® Atom™ Processor however it is possible in the breakpoint dialog to specify which thread or thread group to apply a breakpoint to.



This is true for both code breakpoints and data breakpoints (watchpoints). If you are debugging with data that located in read-only memory, even hardware data breakpoints are available.

The Thread View accessible via the Thread View button  let's you pick which threads to focus on while debugging. It shows the defined threads and their state. You can define thread groups and freeze or thaw specific threads as needed.



A thread can have three different attributes for debug purposes:

**frozen:** The thread does not resume when you resume executing a set of threads in the job.

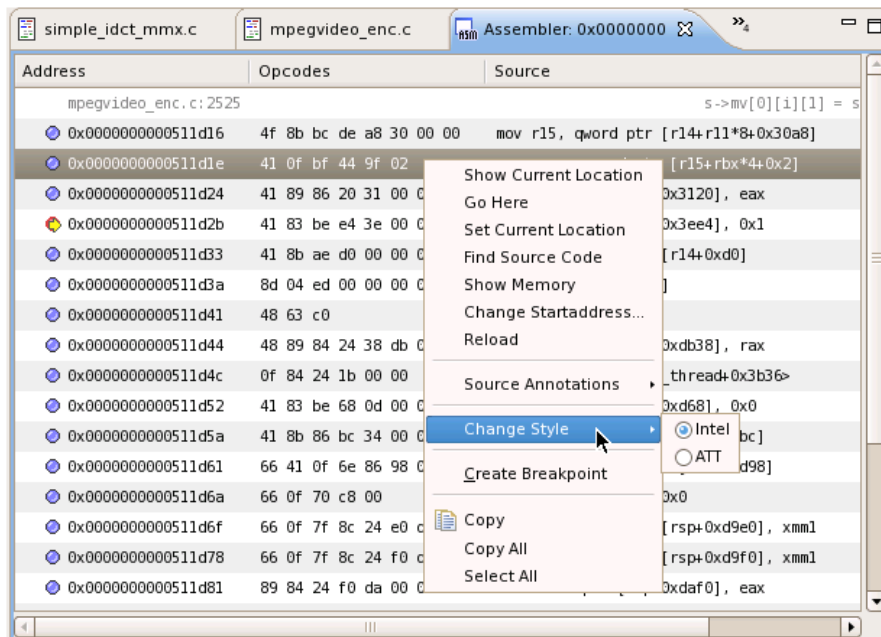
**thawed:** The thread resumes when you resume executing a set of threads in the job.

**uninterrupted:** The thread continues running without being interrupted for events. It basically detaches the thread from the debugger's control.

The thread window is your main 'thread control' window. You can select by double clicking which thread you would like to have in focus. You can even use the context menu to freeze and thaw individual threads

### 5.1.4 Disassembly and Register Views

If during the debugging session you suspend the debuggee at a location where there are no sources available the debugger will automatically open up an assembly window. It allows for pure disassembly as well as disassembly and source interlaced views. It is possible to select the style of assembly mnemonics used by the debugger. The default is to show the Intel style assembly but you can switch to the AT&T style which should be familiar for many Linux\* users. Finally it is also possible to edit code in the disassembly window on the fly making adjustments by replacing assembly code and testing what the outcome of such a change may be.



The **Registers View** is automatically updated whenever the debuggee is halted and shows the register contents for the current debug scope. Register contents that has changed since the last halt is being highlighted in red for easy identification of changes.

Register	Value	Description
\$eax	0xffffdfc	-516
\$ecx	0x4	4
\$edx	0x28	40
\$ebx	0x917cd78	152554872
\$esp [\$sp]	0xbfd46064	(void *) 0xbfd46064
\$ebp [\$fp]	0xbfd46088	(void *) 0xbfd46088
\$esi	0x8f93020	150548512
\$edi	0x47baaff4	1203417076
\$eip [\$pc]	0x804a4dc	(void *) 0x804a4dc
\$eflags	0x200246	2097734
\$cs	0x73	115
\$ss	0x7b	123
\$ds	0x7b	123
\$es	0x7b	123
\$fs	0x0	0
\$gs	0x33	51
\$fctrl	0x37f	895
\$fstat	0x4020	16416
\$ftag	0x0	0
\$fiseg	0x73	115
\$fioff	0x486a3c5c	1214921820
\$foseg	0x7b	123
\$fooff	0xbfd45730	-1076603088
\$fop	0x55d	1373
\$f0	0x0	(unprintable extended double precision float)
\$f1	0x0	(unprintable extended double precision float)
\$f2	0x0	(unprintable extended double precision float)

### 5.1.5 Scripting Language

Create a batch file based on a rich set of powerful debugging script language commands and execute it in non-interactive mode. Results can be logged and analyzed afterwards.

### 5.1.6 Store and Restore of Debug Sessions

The user can store basic debug session settings such as connection information, substitution paths, environment variables, loaded debuggee and load arguments, breakpoints and watchpoints. These settings can then be retrieved by loading the previously stored session settings file. This makes it easy to commence a debug session where the user may have left off earlier.

### 5.1.7 Enhanced GDB compatibility

This release features more GDB compatibility features in its command language interface. Included are the following features:

- Support for command history
- Support for info variables, info types and info functions
- Command, filename and variable completion
- Support for user-defined commands

### 5.1.8 Command History

The debugger will read `.gdb_history` by default. You can also use the following commands to control the behavior of command history:

- **Renaming the history file**  
There are 2 ways to rename the history file, which will override the default (`.gdb_history`).
  1. **(idb) set history filename <name>**
  2. Set the environment variable **GDBHISTFILE** to the filename of the history file you want to use. This can be overridden during the debug session by using the `set history file <name>` command.
- **Turning history recording on & off**  
**(idb) set history save [on|off]**
- **Configuring the history size**  
There are 2 ways to configure the size of the history buffer (the number of items retained):
  1. **(idb) set history size <number>**

2. Set the environment variable **HISTSIZE** to the size you want.

If neither of these choices are used, the default value is 256. s

### 5.1.9 info variables, info types, info functions

`info var[iables] [<string>|REGEXP]`

<string> is a simple search string; *REGEXP* is a regular expression. The resulting list is the list of global and static variables in the debuggee that match <string> or *REGEXP*.

`info fu[nctions] [<string>|REGEXP]`

<string> is a simple search string; *REGEXP* is a regular expression. The resulting list is the list of global and static functions in the debuggee that match <string> or *REGEXP*.

`info ty[ypes] [<string>|REGEXP]`

<string> is a simple search string; *REGEXP* is a regular expression. The resulting list is the list of types used in the debuggee that matches <string> or *REGEXP*.

### 5.1.10 Command, filename and variable completion

Completion of commands, filenames, and variables is supported. Simply start typing a command, filename or variable and press the *tab* key. If there is more than one alternative, the debugger sounds a bell; pressing *tab* again causes the debugger to list the alternatives.

Using single and double quotes influences the set of possible alternatives. Single quotes can be used to fill in C++ names, which contains special symbols (":", "<", ">", "(", etc.). Double quotes tell the debugger to look for alternatives among file names only.

The debugger doesn't automatically detect cases where quotes are needed (as GDB does).

### 5.1.11 Support for user-defined commands

The debugger now supports user defined commands. These commands support *if*, *while*, *loop\_break* and *loop\_continue* in their bodies. User-defined commands can have up to 10 arguments separated by whitespace. Argument names are \$arg0, \$arg1, \$arg2,..., \$arg9. The number of arguments is held in \$argc.

The debugger supports the following commands to control user-defined commands:

- **(idb) set max-user-call-depth <depth>**  
Sets max level of recursion for user defined commands. The default value is 1024.
- **(idb) show max-user-call-depth**  
Shows current available level of recursion.

The debugger doesn't yet support the *document*, *help user-defined*, *dont-repeat* commands or user-defined hooks.

### 5.1.12 Support for Pending Breakpoints

If the user tries to set a breakpoint which cannot be resolved to as specific location, the debugger asks the user whether to create a special breakpoint, called a *pending* breakpoint. This breakpoint will be evaluated when shared libraries are loaded and if they can be resolved, that are transformed into a regular breakpoint (where the debugger can stop). The debugger supports specific commands for managing the pending breakpoint support:

- **(idb)set breakpoint pending auto**  
  
This is the default value. If the debugger cannot resolve breakpoint location, the user is asked whether to create a pending breakpoint.
- **(idb) set breakpoint pending on**  
  
Always set pending breakpoints on unresolved locations.
- **(idb) set breakpoint pending off**  
  
Never set pending breakpoint on unresolved locations.
- **(idb) show breakpoint pending**  
  
Show the behavior of the pending breakpoint support.

Normal breakpoint operations are valid for pending breakpoints. Pending breakpoints can be enabled, disabled and deleted. A user condition can be assigned to pending breakpoint and it will be promoted to the breakpoint when the location is resolved.

### 5.1.13 Hardware Watchpoints

The Intel(R) C++ Application Debugger for Linux\* OS Supporting Mobil Internet Devices now supports hardware watchpoints using dedicated target processor registers additionally to software watchpoints.

The syntax for the hardware watchpoint support is

```
(ldb) ldb hawatch <address>:<size>
```

```
(ldb) ldb hwatch <address>:<size>
```

hawatch sets a hardware watchpoint at address with specified size that gets triggered by any access.

hwatch sets a hardware watchpoint at address with specified size that gets triggered by write accesses only.

The optional size parameter defines the range of memory that gets watched. Only 1,2,4 are currently valid sizes.

This info also accessible via `help hawatch` or `help hwatch`.

The debugger's watchpoint dialog also provides GUI support for this command and the debugger's symbol selector can be used to select a variable for the hardware watchpoint address.

Since hardware watchpoints require explicit addresses, please make sure that you specify a memory address if you add the watchpoint location in this dialog manually, either by typing in an address or by specifying `&<variable name>`.

## 6 Usage Notes

### First Steps

#### 6.1.1 Enabling the Intel Debugger Remote Server

Before you can start the debugger, you need to first start the Intel Debugger Remote Server on the target system. The remote server, `idbserver`, is installed with the debugger installation on the host system.

##### 6.1.1.1 To enable the Intel(R) Debugger Remote Server:

1. Copy the Intel(R) Debugger Remote Server from the host system to the shared drive. Open a shell and use the following command:

```
cp <install-dir>/atom/idb/2.3.xxx/bin/ia32/idbserver /mnt/shared/
```

Notice that the path includes the full version number including update number.

2. Start the remote server from a shell. Use the following command:

```
/mnt/shared/idbserver
```

Leave this shell open while the remote server is running.

#### 6.1.2 Starting the Debugger

To start the remote debugger on the host enter the following command in the shell:

```
<install-dir>/atom/idb/2.3.xxx/bin/ia32/idb_remote
```

This is in addition to the `idbserver` debug handler that needs to be launched on the target device.

The debugger is now running.

#### 6.1.3 Connecting to a Remote Debugging Target

Before you can open an Embedded to debug, you must connect to the system on which it is running, the target.

##### 6.1.3.1 To connect to the target:

1. Get the IP address of the target by using the `ifconfig` command on the target system. Against `eth0` look for an `inet` entry, such as `10.100.15.78`. Note this address to use it as the IP address.

2. Select **File > Connect....**  
The **Open Connections** dialog box opens.
3. Enter the IP address of the target from step 1.
4. Enter the port number of the target. The target shell displays the port number when you start the Intel(R) Debugger Remote Server.
5. Click **OK**.

The host connects to the target system. The Intel® Debugger remote server command shell on the target indicates that it is connected.

The debugger is now ready to be used and to be attached to a debuggee process. Please refer to the Getting Started document at [/opt/intel/atom/documentation/idb/Getting\\_Started.html](/opt/intel/atom/documentation/idb/Getting_Started.html) for more details.

## Debugging Yocto Project\* Applications with Intel® Debugger

### 6.1.4 Debugging an application running on a QEMU\* based virtual machine

1. Ensure the Application Development Toolkit (ADT) environment variables have been exported to the development machine

```
$ . /opt/poky/1.1/environment-setup-i586-poky-linux
```

2. Now you can launch QEMU\* with the appropriate Yocto Project\* OS image :

```
$ poky-qemu qemu86 bzImage-2.6.37-qemu86-1.0.bin yocto-image-lsb-  
qemu86-1.0.rootfs.ext3
```

```
Continuing with the following parameters:
```

```
KERNEL: [bzImage-2.6.37-qemu86-1.0.bin]
```

```
ROOTFS: [yocto-image-lsb-qemu86-1.0.rootfs.ext3]
```

```
FSTYPE: [ext3]
```

```
Setting up tap interface under sudo
```

```
[sudo] password for user:
```

```
Acquiring lockfile for tap0...
```

```
Running qemu...
```

```
/opt/poky/1.1/sysroots/i686-pokysdk-linux/usr/bin/qemu -kernel bzImage-  
2.6.37-qemu86-1.0.bin -net nic,vlan=0 -net  
tap,vlan=0,ifname=tap0,script=no,downscript=no -hda yocto-image-lsb-  
qemu86-1.0.rootfs.ext3 -show-cursor -usb -usbdevice wacom-tablet -vga  
vmware -enable-gl -no-reboot --append "vga=0 root=/dev/hda mem=128M  
ip=192.168.7.2::192.168.7.1:255.255.255.0 oprofile.timer=1 "
```

3. Please note which IP address is assigned to your virtual machine. In this example it is highlighted as 192.168.7. The actual IP address of your virtual machine is needed for remote debugging.

**Note:**

When using an OS image it is recommended to have ssh enabled. This allows to copy files easily to and from the target, and start the remote debug agent.

4. Copy the remote debug agent to the target:

```
$ scp /opt/intel/atom/idb/2.3.xxx/bin/ia32/idbserver  
root@192.168.7.2:/home/root/
```

5. In the console window you are building your applications and starting the debugger from you need to source ADT as well:

```
$ . /opt/poky/1.1/environment-setup-i586-poky-linux
```

6. Launch the debugger.

```
$ /opt/intel/atom/idb/2.3.xxx/bin/ia32/idb_remote &
```

7. After launching the debugger you can then start the remote debug agent, idbserver, on the target (e.g via ssh).
8. Connect to it from the debugger using the connect button and the IP address provided at QEMU launch time. The TCP/IP port defaults to 2000.

### 6.1.5 Debugging an application running on a physical device

1. Ensure the Application Development Toolkit (ADT) environment variables have been exported to the development machine

```
$ . /opt/poky/1.1/environment-setup-i586-poky-linux
```

2. Please note which IP address is assigned to your device in the network.

**Note:**

When using an OS image it is recommended to have ssh enabled. This allows to copy files easily to and from the target, and start the remote debug agent.

3. Copy the remote debug agent to the target:

```
$ scp /opt/intel/atom/idb/2.3.xxx/bin/ia32/idbserver  
root@192.168.7.2:/home/root/
```

4. In the console window you are building your applications and starting the debugger from you need to source ADT as well:

```
$ . /opt/poky/1.1/environment-setup-i586-poky-linux
```

5. Launch the debugger.

```
$ /opt/intel/atom/idb/2.3.xxx/bin/ia32/idb_remote &
```

6. After launching the debugger you can then start the remote debug agent, idbserver, on the target (e.g via ssh).
7. Connect to it from the debugger using the connect button and the IP address of your target device. The TCP/IP port defaults to 2000.

## Debugging MeeGo\* Applications with Intel® Debugger

### 6.1.6 Setup of Debugger Target Environment

Please be aware of the necessity to have the Intel® Debugger Remote Server for IA32 Linux\* present on your MeeGo\* based target installation. It can be found in the

```
<install-dir>/atom/idb/2.3.xxx/bin/ia32
```

directory. where the default installation directory is `/opt/intel`. To use this remote server simply copy the binary `idbserver` onto your target system via `sftp` and launch it on your target environment. You will need to ensure that the `ssh` demon has been installed and started on the MeeGo\* platform first:

```
$ zypper install openssh-server
$ chkconfig --add sshd
$ /etc/init.d/sshd start
```

If you are connecting to a QEMU based virtual machine that has been launched from Qt Creator and the MeeGo\* SDK, the command to establish and sftp connection for debug server download is

```
$ sftp -P 6666 meego@127.0.0.1
```

If your target is a physical device you can use the standard ssh ports without specifying the `-P` option using the actual IP address of your target device.

To launch the idbserver Debugger Remote Server you would establish an ssh connection.

QEMU target:

```
$ ssh meego@127.0.0.1 -p 6666
```

Physical target:

```
$ ssh <device ip address>
```

Once the ssh connection is established you can then launch the idbserver via the command

```
$ ./idbserver &
```

The server will listen to debugger communication requests on port 2000

```
meego@[meego-tablet-sdk]::~~$ Intel(R) Debugger Remote Server for IA32
Linux*, Build 76.xxx.x
Copyright (C) 2006-2010 Intel Corporation. All rights reserved.
Waiting for connection with "tcpip:2000"...
```

You may want to ensure that the Debugger Remote Server runs in the same type of user space as you intend for the application to run in, whether it is with standard user rights or with super user (root) rights.

**1st Note on QEMU Usage:** To be able to set up communication between the application launched inside a QEMU and the Remote Debug Server using TCP/IP port 2000, you need to modify the information file for the QEMU image you want to debug. In the information file you will add port forwarding to the Remote Debug Server (idbserver). If, for example, the image you would like to enable is “meego-tablet-ia32-qemu--sda-runtime” you would change to the `/usr/lib/madde/linux-i686/runtimes/meego-handset-ia32-qemu--sda-runtime/` directory and edit the information file adding the part in red below:

```

qemu_args='-name MeeGo -m 1024 -boot c -hda meego-handset-ia32-qemu-
1.1.20101031.2201-sda.raw -enable-kvm -vga std -enable-gl -device
virtio-gl-pci -skin /opt/meego/qemu-
gl/share/qemugl/meego/skin/handset/skin.xml -usbdevice tablet -soundhw
ac97 -net nic -net user,hostfwd=tcp:127.0.0.1:6666-
:22,hostfwd=tcp:127.0.0.1:13219-:13219,hostfwd=tcp:127.0.0.1:14168-
:14168,hostfwd=tcp:127.0.0.1:2000-:2000'

```

**2nd Note on QEMU Usage:** If your application does not launch correctly and does not get downloaded to the target platform this could be due to strict SSH key checking enabled. The easiest way to get around this is to edit the global SSH configuration file (/etc/ssh/ssh\_config) or the user specific SSH configuration file (~/.ssh/config) on your target device. This will help avoid access denial due to key checking.

Add the following lines to the beginning of the SSH configuration file.

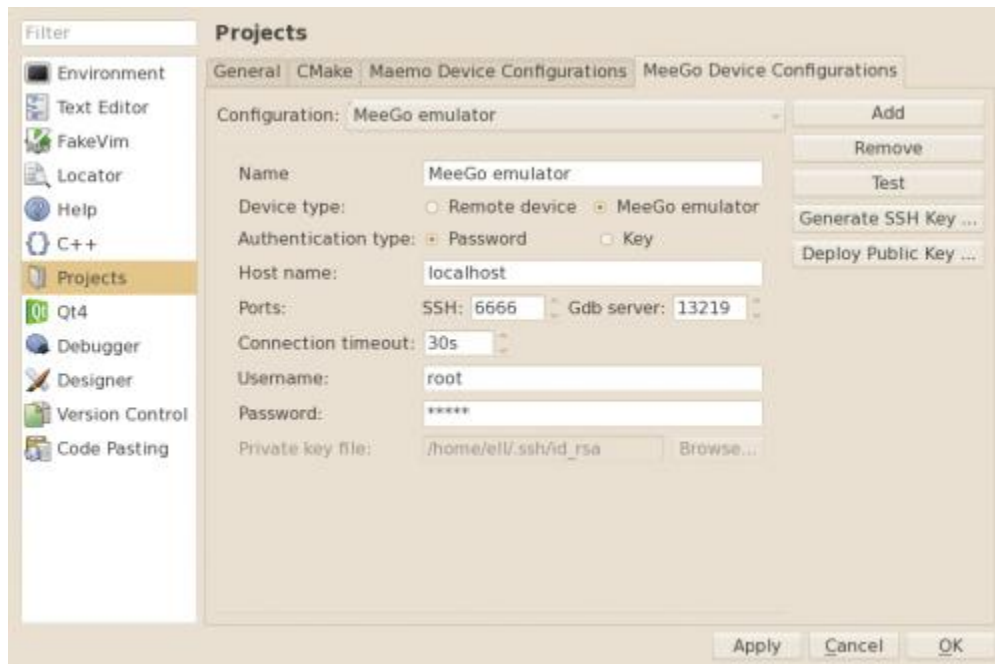
```

Host 172.0.0.1
StrictHostKeyChecking no
UserKnownHostsFile=/dev/null

```

### 6.1.7 Configure Device in Qt Creator

Before you can launch an application that has just been built in QtCreator\* and have it deployed and run on your MeeGo\* target you will first need to define a MeeGo\* Device Configuration for your Qt Creator project. In QtCreator select the **Tools>Options pull-down menu**. There pick **Projects** from the menu bar on the left. Select the **MeeGo\* Device Configurations** tab. For a QEMU virtual machine based target the setting should be similar to the ones shown below:



For a physical target you may choose the settings that best fit your host-target connection.

### 6.1.8 Launch the application from Qt Creator

In Qt Creator select the **run** button



### 6.1.9 Launching the Debugger

Ensure the environment variables for the Intel® Debugger are configured correctly. This can be done by running `<Composer_installdir>/bin/compilervars.sh ia32` or `<IDB_installdir>/bin/idbvars.sh ia32`.

The default for `<Composer_installdir>` is `/opt/intel/composerxe/` and for `<IDB_installdir>` the default is `/opt/intel/atom/idb/2.3.xxx/`.


Change to the `<IDB_installdir>/bin/ia32/` directory and enter

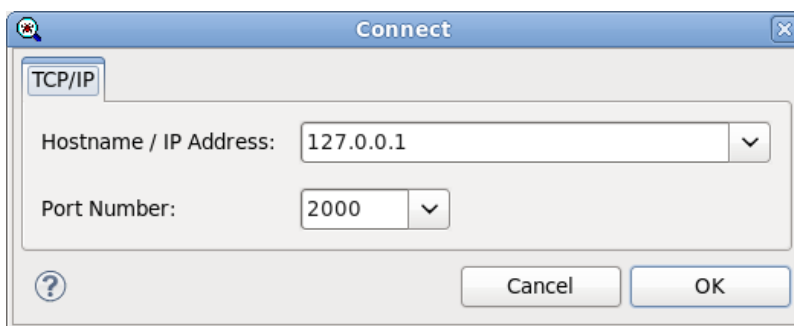
```
$ ./idb_remote
```

### 6.1.10 Loading and Attaching Debugger to Application

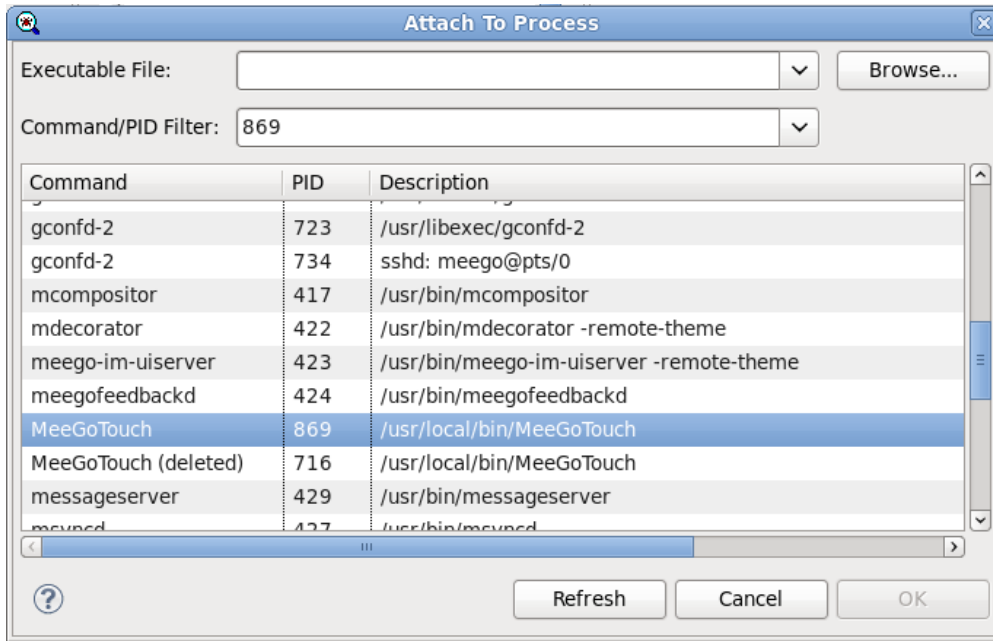
There are two alternative ways of launching a debug session for your application running on a virtual or physical MeeGo\* device.


#### *Attach to Target Application*

In the Debugger GUI select the **Connect** button . After the connection to the target is established using parameters similar to the ones shown below



You can then select the running target process using the **Attach to Process**  button.



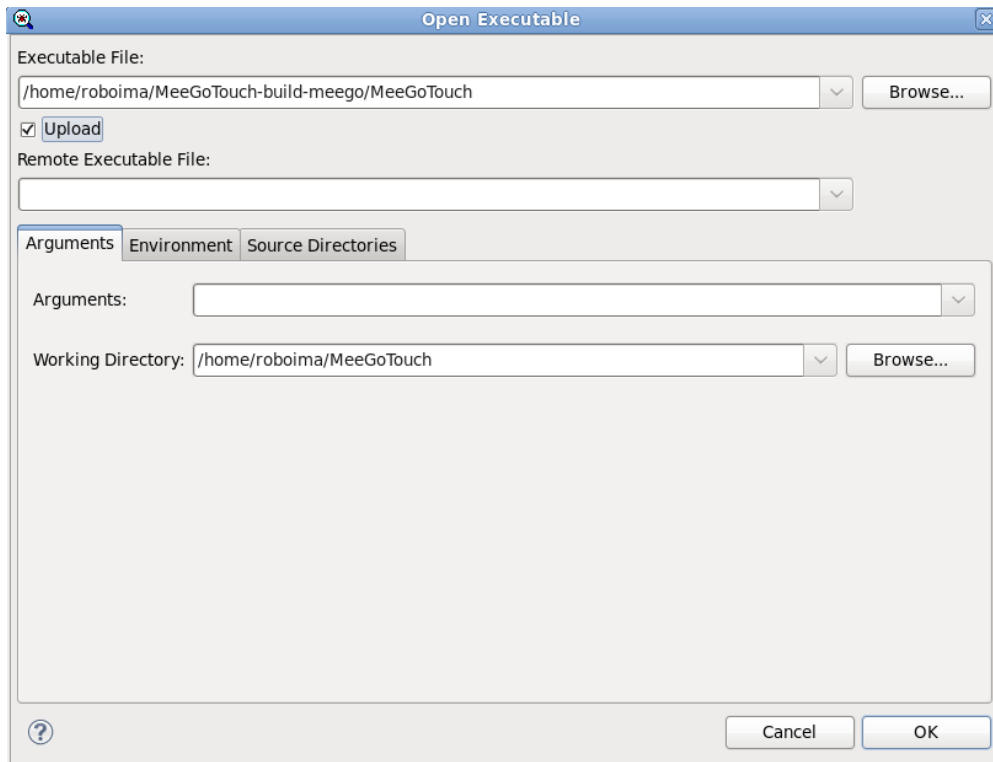
The path to the executable binary on the host should be filled in and found automatically. If this is not the case you can navigate to it using the **Browse**  button.

### *Launch Application from Debugger*

Alternatively you can also download an application that was built inside or outside of Qt Creator by using the Load button



in the debugger GUI and uploading the executable to the connected virtual or physical target using the dialog depicted below:



Finally after the application is downloaded or you attached to a running application on your target device (whether virtual or physical) you are able to take advantage of the debugger's rich feature set and you can start debugging your application.

## 7 Issues and Limitations

### Known Issues and Limitations

#### 7.1.1 Online Help for Intel® Debugger has not been updated for this release

The context sensitive help available from within the Intel® Debugger graphical user interface reflects the status prior to the Intel® Embedded Software Development Tool Suite 2.3 release. Please consult these release notes and the Getting Started Guide for the latest up-to-date information.

#### 7.1.2 Application Upload to Target from within Debugger

It currently is only possible to upload an application to the MeeGo\* target from within the debugger once per debug session. For re-uploading the debuggee application from within the same debugger instance, first disconnect the debugger from the remote debug server idbserver and then kill and restart idbserver. Afterwards you can reconnect and upload the application again.

#### 7.1.3 Signals Dialog Not Working

The Signals dialog accessible via the GUI dialog Debug / Signal Handling or the shortcut Ctrl+S is not working correctly. Please refer to the Intel® Debugger (IDB) Manual for use of the signals command line commands instead.

#### 7.1.4 Resizing GUI

If the debugger GUI window is reduced in size, some windows may fully disappear. Enlarge the window and the hidden windows will appear again.

#### 7.1.5 Problems Communicating with Remote Debug Server launched inside QEMU

If you encounter problems communicating with the Remote Debug Server launched inside a QEMU you may need to modify information file for the QEMU image you want to debug. In the information file you will add port forwarding to the Remote Debug Server (idbserver).

If, for example, the image you would like to enable is “**meego-tablet-ia32-qemu-<version>-sda-runtime**” you would change to the `/usr/lib/madde/linux-i686/runtimes/meego-handset-ia32-qemu-<version>-sda-runtime/` directory and edit the information file adding the part in red below:

```
qemu_args='-name MeeGo -m 1024 -boot c -hda meego-handset-ia32-qemu-1.1.20101031.2201-sda.raw -enable-kvm -vga std -enable-gl -device virtio-gl-pci -skin /opt/meego/qemu-gl/share/qemugl/meego/skin/handset/skin.xml -usbdevice tablet -soundhw ac97 -net nic -net user,hostfwd=tcp:127.0.0.1:6666-:22,hostfwd=tcp:127.0.0.1:13219-:13219,hostfwd=tcp:127.0.0.1:14168-:14168,hostfwd=tcp:127.0.0.1:2000-:2000'
```

### 7.1.5 \$cdir, \$cwd Directories

\$cdir is the compilation directory (if recorded). This is supported in that the directory is set; but \$cdir is not itself supported as a symbol.

\$cwd is the current working directory. Neither the semantics nor the symbol are supported.

The difference between \$cwd and '.' is that \$cwd tracks the current working directory as it changes during a debug session. '.' is immediately expanded to the current directory at the time an entry to the source path is added.

### 7.1.6 info stack Usage

The GDB mode debugger command info stack does not currently support negative frame counts in the optional syntax below:

```
$ info stack [num]
```

A positive frame count num will print the innermost num frames. A negative or zero count will print no frames rather than the outermost num frames.

### 7.1.7 \$stepg0 Default Value Changed

The debugger variable \$stepg0 changed default to a value of 0. With the value "0" the debugger will step over code without debug information if you do a "step" command. Set the debugger variable to 1 to be compatible with previous debugger versions as follows:

```
(idb) set $stepg0 = 1
```

### 7.1.8 Thread Syncpoint Creation in GUI

While for plain code and data breakpoints the field **Location** is mandatory, thread syncpoints require both **Location** and **Thread Filter** to be specified. The latter specifies the threads to synchronize. Please note that for the other breakpoint types this field restricts the breakpoints created to the threads listed.

### 7.1.9 Data Breakpoint Dialog

The fields **Within Function** and **Length** are not used. The location to watch provides the watched length implicitly (the type of the effective expression is used). Also **Read** access is not working.

### 7.1.10 Stack Alignment for IA-32 Architecture

Due to changes in the default stack alignment for the IA-32 architecture, the usage of inferior calls (i.e. evaluation of expressions that cause execution of debuggee code) might fail. This can cause as well crashes of the debuggee and therefore a restart of the debug session. If you need to use this feature, make sure to compile your code with 4 byte stack alignment by proper usage of the --falign-stack=<mode> option.

### 7.1.11 GNOME Environment Issues

With GNOME 2.28, debugger menu icons may not being displayed by default. To get the menu

icons back, you need to go to the **System->Preferences->Appearance, Interface** tab and enable, "**Show icons in menus**". If there is not **Interface** tab available, you can change this with the corresponding GConf keys in console as follows:

```
(idb) gconftool-2 --type boolean --set /desktop/gnome/interface/buttons_have_icons true
```

```
(idb) gconftool-2 --type boolean --set /desktop/gnome/interface/menus_have_icons true
```

## 8 Attributions

This product includes software developed at:

The Apache Software Foundation (<http://www.apache.org/>).

Portions of this software were originally based on the following:

- software copyright (c) 1999, IBM Corporation., <http://www.ibm.com>.
- software copyright (c) 1999, Sun Microsystems., <http://www.sun.com>.
- the W3C consortium (<http://www.w3c.org>) ,
- the SAX project (<http://www.saxproject.org>)
- voluntary contributions made by Paul Eng on behalf of the Apache Software Foundation that were originally developed at iClick, Inc., software copyright (c) 1999.

This product includes updcrc macro,  
Satchell Evaluations and Chuck Forsberg.  
Copyright (C) 1986 Stephen Satchell.

This product includes software developed by the MX4J project  
(<http://mx4j.sourceforge.net>).

This product includes ICU 1.8.1 and later.  
Copyright (c) 1995-2006 International Business Machines Corporation and others.

Portions copyright (c) 1997-2008 Cypress Semiconductor Corporation.  
All rights reserved.

This product includes XORP.  
Copyright (c) 2001-2004 International Computer Science Institute

This product includes software from the book

"Linux Device Drivers" by Alessandro Rubini and Jonathan Corbet,  
published by O'Reilly & Associates.

This product includes hashtab.c.  
Bob Jenkins, 1996.

## 9 Disclaimer and Legal Information

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to:  
<http://www.intel.com/design/literature.htm>

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. Go to:  
[http://www.intel.com/products/processor\\_number/](http://www.intel.com/products/processor_number/)

BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino Inside, Cilk, Core Inside, i960, Intel, the Intel logo, Intel AppUp, Intel Atom, Intel Atom Inside, Intel Core, Intel Inside, Intel Inside logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel Sponsors of Tomorrow., the Intel Sponsors of Tomorrow. logo, Intel StrataFlash, Intel Viiv, Intel vPro, Intel XScale, InTru, the InTru logo, InTru soundmark, Itanium, Itanium Inside, MCS, MMX, Moblin, Pentium, Pentium Inside, skool, the skool logo, Sound Mark, The Journey Inside,

vPro Inside, VTune, Xeon, and Xeon Inside are trademarks of Intel Corporation in the U.S. and other countries.

\* Other names and brands may be claimed as the property of others.

Microsoft, Windows, Visual Studio, Visual C++, and the Windows logo are trademarks, or registered trademarks of Microsoft Corporation in the United States and/or other countries.

Java and all Java based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Copyright (C) 2008 - 2011, Intel Corporation. All rights reserved.