Intel® C++ Compiler 2.0.4
For Windows CE, Professional*

Intel® C++ Compiler

For Platform Builder for Microsoft Windows* CE and
Microsoft eMbedded Visual C++*

Release Notes

Document Number: 280124-031US

Contents

Overview

The product *Intel® C++ Compiler 2.0.4 For Windows* CE, Professional* is a set of compiler tools and debugging extensions that integrate into various Microsoft* build environments. It's intended to develop applications optimized for Intel XScale® microarchitecture written in C/C++ and assembly source code.

The product also contains the *Intel® C++ Compiler Build Support for Adaptation Kits* to integrate the Intel® C++ Compiler into Microsoft Adaptation Kits for Windows Mobile* 2003 Software (PocketPC, Smartphone) and Windows Mobile* 5.0.

Besides the highly optimizing Intel® C++ Compiler the package also contains extensions to Microsoft* environments that extend debugging capabilities for the Intel XScale® microarchitecture. This document describes release notes for the code generating components only. For release notes for the debugging extensions see separate document.

The product Intel® C++ Compiler For Windows* CE, Professional consists of two installation packages:

- Intel® C++ Compiler For Platform Builder for Microsoft Windows* CE
  (installs compiler and debugging extensions components for Platform Builder)
- Intel® C++ Compiler For Microsoft eMbedded Visual C++*
  (installs compiler and debugging extensions components for eVC++ and Visual Studio 2005)*

Please refer also to the Product Cross Reference to learn details on the installation packages.

This document describes release notes for the code generating components only. For the debugging extensions see separate release notes.

To receive technical support and updates, you need to register your Intel Software Product. See the Technical Support section.

Product Contents

This product release contains the components as follows:

Component	Description	Version
Intel® Compiler Tools	Intel® C++ Compiler	2.0.37
	Intel® Run-time Library	2.0.32
	Intel® Assembler	2.2.69
Intel® C++ Compiler Build Support for Adaptation Kits	Intel® Adaptation Kit Installation Utility	2.0.7
	Intel® Adaptation Kit Uninstallation Utility	2.0.7
	Intel® Select Compiler Utility	2.0.0.3
Product Documentation	A product documentation index file is provided for easy access of all the documents. It's located at (default): `C:\Program Files\Intel\Intel(R) C++ Compiler 2.0.4 for Platform Builder\doc\doc_intex_PB.htm`. (see also Documentation)	NA

Intel® Debugging Extensions *Intel® Debugging Extensions For JTAG*	*described in separate Release Notes*	

Product Support

This product release provides support for target OS, platform, development environment and target CPU as follows:

Support	Description	Version
Target OS/Platform	Microsoft Windows* CE .NET 4.2	NA
	Microsoft Windows* CE 5.0	NA
	Microsoft* Windows Mobile* 2003 Software (PocketPC, Smartphone)	NA
	Microsoft* Windows Mobile* 5.0	NA
Development Environment	Microsoft* Platform Builder for Windows* CE .NET 4.2	NA
	Microsoft* Platform Builder for Windows* CE 5.0	NA
	Microsoft* Platform Builder for Windows Mobile* 5.0	NA
	Microsoft eMbedded Visual C++* 4.0 (with Service Pack 1, 2 or 4)	NA
	Microsoft* Visual Studio* 2005	NA
Target CPU	Intel® PXA25x Processor Family	all versions
	Intel® PXA26x Processor Family	all versions
	Intel® PXA27x Processor Family	all versions

Product Cross Reference

This product reference is an overview of the distribution product with their respective installation packages and components:

Product	Installation Packages	Components
Intel® C++ Compiler 2.0.4 For Windows* CE, Professional	Intel® C++ Compiler For Microsoft eMbedded Visual C++*	Intel® C++ Compiler For Microsoft eMbedded Visual C++*
		Intel® Debugging Extensions For Microsoft eMbedded Visual C++* *(described in separate Release Notes)*
	Intel® C++ Compiler For Platform Builder for Microsoft Windows* CE	Intel® C++ Compiler For Platform Builder for Microsoft Windows* CE
		Intel® C++ Compiler Build Support for Adaptation Kits
		Intel® Debugging Extensions For JTAG For Platform Builder for Microsoft Windows* CE *(described in separate Release Notes)*

Please refer also to the Product Contents.

Compatibility

The Intel® C++ Compiler 2.0.4 For Platform Builder for Microsoft* Windows CE and Microsoft eMbedded Visual C++* replaces all previous versions of the Intel® C++ Compiler For Windows* CE. Any previous version of the compiler must be uninstalled prior to the Intel® C++ Compiler 2.0.4 installation. For details please refer to the Installation section.

System Requirements

Host System

Same hardware and software requirements as for the installed Development Environments.

Host Software

- Any of the Development Environments properly installed
- Adobe* Acrobat Reader* version 4.0 or later is required to view some of the product documentation
- A Web browser is required to view some of the HTML-based product documentation

What's New

No new features were introduced with this release. For problems resolved in this release, please see section Defects Fixed.

What's Was New in Previous 2.0 Releases

Support for Microsoft* Visual Studio* 2005

The Intel® C++ Compiler is now validated against the released version of Microsoft* Visual Studio* 2005.

Support for Microsoft* Platform Builder for Windows Mobile* 5.0

The Intel® Adaptation Kit Integration Utility integrates the Intel® C++ Compiler into the Adaptation Kit for Windows Mobile* 5.0 environment and enables users to create code for Windows Mobile* based devices using the Intel® C++ Compiler For Windows* CE.

Support for the Intel® Wireless MMX™ 2 Technology

Intel® Wireless MMX™ 2 Technology are the Intel® New Media Technology Extensions to the existing Intel® Wireless MMX™ technology.

The Intel® C++ Compiler uses the new option `/QTPxsc4` to enable the use of Intel® Wireless MMX™ 2 technology by supporting all the instructions via intrinsic functions.
The Intel® Assembler uses the new option `/mcpu 4` to enable usage of Intel® Wireless MMX™ 2 technology instructions.

New Compiler Options

The following table lists compiler options that provide new functionality in this release:

Option	Description	Default
/EHa	Enables asynchronous C++ exception handling	OFF
/EHc	Assume external "C" functions do not throw exceptions	OFF
/EHs	Enables synchronous C++ exception handling	OFF
/Qexpand_env [text]	Expands environment variables in the given text text and processes the resulting text as command line options or file specifications	OFF
/Zi[-]	New parameter added to existing option /Zi: /Zi- disables generation of symbolic debugging information	OFF

Additional Mnemonics for Intel® Wireless MMX™ Technology Configuration/Status Registers

The Intel® Assembler now also accepts Intel® Wireless MMX™ technology Configuration/Status Register names in the format of wC<*x*>, where <*x*> represents the register number of Coprocessor 1. The mnemonics which can be used synonymously are shown in the following table:

Register Name	Coprocessor Register Number	Alternative Mnemonics	
Coprocessor ID, Revision, Status	0	wCID	wC0
Control	1	wCON	wC1
Saturation SIMD flags	2	wCSSF	wC2
Arithmetic SIMD Flags	3	wCASF	wC3
Reserved	4...7	-	-
General Purpose Register 0	8	wCGR0	wC8
General Purpose Register 1	9	wCGR1	wC9
General Purpose Register 2	10	wCGR2	wC10
General Purpose Register 3	11	wCGR3	wC11
Reserved	12...15	-	-

Integer Vector Class Library Support

The Intel® C++ Compiler For Windows* CE supports the Integer Vector Classes Ivec with intrinsic functions. A new header file ivec.h is added to this release.

C Structured Exception Handling (SEH) Support

Microsoft* C Structured Exception Handling is now supported. Please refer to SEH Limitations to see the limitations in using Structured Exception Handling with the Intel® C++ Compiler For Windows* CE.

Natural Alignment in Structures changed

The default behavior for data alignment in structures (natural alignment) was changed for structure elements of 64bit data types (long long):
- Previous versions of the Intel compiler aligned 64bit data to 4byte boundaries.
- The new Intel compiler aligns 64bit data to 8byte boundaries to be compatible with the Microsoft* compiler (v4.x, 5.x).

If you want to restore the default alignment as in previous versions, use the option /Zp4

This change in default alignment should not have any effect on your application unless you are using 64bit data types (__int64 or __m64). If you are using these data types in objects that were built with a previous version of the Intel compiler, but you can not recompile the code, please make sure to set the option /Zp4.

It is strongly recommended to recompile your existing application using the new default setting, especially if you link in code/libraries that are built with the Microsoft* compiler.

Assembly Inline Support Additions

The Intel® C++ Compiler For Windows* CE now supports labels in inlined code as well as register lists.

Additional Warnings and Diagnostics

This version of the Intel C++ Compiler has improved diagnostic capabilities which may result in new informational, warning or error messages when your application is recompiled.

Defects Fixed

Wrong result on complex calculation using optimization

Under certain conditions and with optimization the subtraction of two large values did not result in zero as expected.

Loss of debug information

No debug information was generated for specific applications built with /O2 in the Microsoft* Visual Studio* 2005 environment. The problem did not occur in the Microsoft eMbedded Visual C++* 4.x build environment.

Assembler generates error when MACRO definition includes a space

An empty line between the MACRO keyword and the start of the macro body caused the assembler to stop with an error message.

Intel assembler requires complete expression in preprocessor

For a conditional expression defined in the assembler, a test for this expression produced an error if the condition was not explicitly specified in the test statement.

Example:

```
A_1               EQU  0x0
A_2               EQU  0x01
B_1               EQU  0x0
SELECT            EQU  B_1
SELECT_IS_A_x     EQU (SELECT = A_1):LOR:(SELECT = A_2)
; Changing the line below to 'IF SELECT_IS_A_x <> 0'
; was necessary in the previous version to avoid an error
IF SELECT_IS_A_x
SOMETHING         EQU 1
ENDIF
```

Decimal immediates resolved incorrectly with MOV command

An immediate value in a MOV command starting with zero was interpreted as an octal value. This was incompatible to the interpretation of the Microsoft assembler.

Defects Fixed in Previous 2.0 Releases

Assembler not switched with Selection Tool

Note: Defect is fixed for Platform Builder 5.0 and Platform Builder for Windows Mobile* (Diagnostic Kit) 5.0 only.

The Intel® Selection Tool didn't switch the assembler with the compiler.

The Intel® Selection Tool, called via `Tools / Select Compiler` from the Microsoft* IDE, now also selects the assembler together with the compiler:

- If the `Intel® C++ Compiler` checkbox is checked, the Intel® C++ Compiler and Intel® Assembler are used for the build process.
- If the `Intel® C++ Compiler` checkbox is unchecked, the Microsoft compiler and assembler are used for the build process.

Corrupt symbol table generated by the assembler

The assembler created corrupt symbol table entries for static symbols, for example

```
006 00000000 UNDEF  notype       Static       | .text
007 00000000 UNDEF  notype       Static       | .pdata
```

Static symbols however cannot be UNDEF. They should be associated with the section they are assigned to. This is solved now.

Assembler issues error on unbalanced quotes in comments

The Intel® Assembler issued an error `unterminated string` if unbalanced quotes were used inside comments, for example

```
; "This is a comment
```

This is solved now. The assembler completely ignores contents of comments.

Assembler issues error on large comment `;/* ...*/`

The Intel® Assembler issued an error `Line too long on input file` if large comments were included in `/* */`, for example

```
; /* This is a comment
; <comment lines following>
; end of comment */
```

This is solved now. The assembler completely ignores contents of comments.

Data misalignment

When passing objects by value in C++ code the Intel Compiler produced data misalignments which caused runtime errors. This problem is fixed with this release.

Syntax checker defect on intrinsic `_mm_extract_pi8`

The compiler issued an error when intrinsic function `_mm_extract_pi8` used. This is solved now.

Unresolved externals on `afxtempl.h`

The compiler issued an unresolved external error when `afxtempl.h` header was included. This is solved now.

Wrong Stack Alignment on Big Local Arrays

Local arrays bigger than 3.5 kB with size not dividable by 4 caused unaligned stack generation together with optimization switched on (`/O1, /O2, /O3` or `/Qip`). This is fixed with this release.

Wrong Offset for Function Pointer within Structures

The compiler calculated wrong offsets for function pointers within structures. It was a problem with pointer members of incomplete type. This is fixed with this release.

Missing Vendor Signature in Compiler Run-time Libraries

The first invocation of the dynamic linked runtime library DLLs `x0__ar10dll.dll` (ARM* mode) or `x0__ar10dllt.dll` (Thumb* mode) issued a warning message on Windows Mobile* 5 devices. The message asked to user to confirm the loading of a DLL from an unknown vendor. The reason for this was a missing vendor certificate in the runtime DLLs. This is fixed with the update. The runtime DLLs are now signed with the need certificates. There are no functional differences to the former version of the libraries.

Incorrect Alignment with `LDR` Instructions using `-QTPxsc3`

The incorrect alignment with `LDR` instructions using `-QTPxsc3` led to runtime application crashes. This problem is solved now.

Optimization Alignment Problem with `-QTPxsc3 -O3`

In very rare cases the compiler does not detect unaligned structure alignment correctly together with options `-QTPxsc3 -O3` and enabled Vectorizer. This has been resolved.

Vectorizer generated incorrect code with `for`-loop

The Vectorizer (when `/QTPxsc3` used) generated incorrect code on for-loops that led to runtime crashes due to a misaligned memory access.

`int32` treated as unaligned

The Compiler treated `int32` data type as unaligned and converted a simple word-memory move into a series of byte-writes.

Handling predicated instructions

Predication and return/epilogue inlining is now improved.

Function return code sequences not optimal

Function return code sequences (especially for functions with multiple exit points) were not optimal. Support for predicated returns/epilogues is now implemented and epilogue

inlining improved.

Inlining with `/Ob1`

When `/Ob1` was set, the compiler did inline functions although they ware marked with a pragma to be excluded from inlining. Option `/Ob1` is now working correctly as specified in the documentation.

Unnecessary `memcpy()` operations instead of register moves on small structures

The Compiler now generates ESP frames and forwards struct params stores to improve performance on operations with small structures. However, the compiler is not able to remove redundant parameter stores and does not free the stack slot for a struct parameter.

Incompatibility in modulo operation for constant-propagated shifters

The Compiler performed an implicit modulo-32 operation for constant-propagated shifters. This behavior was different than from the Microsoft* compiler and is now corrected.

Misalignment exception when application built with `/QTPxsc3` (Vectorizer enabled)

If the following code:

```
int test (short *a)
{
  int b; int i; b = a[0];
  for ( i=1; i<60; i++)
  {
    if (a[i] < b)
    {
      b = a[i];
    }
  }
}
```

was called with a 4byte aligned short pointer it caused a misalignment exception. This is fixed with this release.

Macro redefinition caused `internal error 0_0`

Macro redefinition caused `internal error 0_0`: Compiling a precompiled `*.i` file worked with warning on the precompilation: `incompatible redefinition of macro "__cdecl".` This is fixed now.

Runtime Error on Release Build

In special circumstances runtime error occurred on Release Builds with using compiler option `/Osx`. This is fixed now.

Installation

Before Installation

- You can install the compiler for existing Microsoft* environments only.
- The debugging extensions are always being installed into a separate directory from where they are linked to the Microsoft* environments.
- The compiler and debugging extensions are being installed within one setup. In the custom setup dialog you can deselect components you don't want to have installed.
- If `Intel® C++ Compiler for Platform Builder for Windows Mobile* 5.0` is selected in the `Custom Setup` the installer copies the compiler tools into a base directory. The integration of the compiler into Adaptation Kits have to be done separately as documented in Adaptation Kit Integration.
- For the Intel® C++ Compiler for eMbedded Visual C++* 4.0 installation it is not possible to perform an "All User" installation (see Known Issues).
- You must have administrator privileges to install the Intel® C++ Compiler For Windows* CE.
- Install the corresponding Microsoft* development environment (see Development Environments) and make sure to install all required SDKs, Service Packs and platforms (ARMV4I or ARMV4T). Start the IDE at least once (to make registry entries effective) and make sure that all platforms are closed before installation. Otherwise the installation will abort.
- You must have a valid license file in order to install and use the product.
- Uninstall any previous version of the Intel® C++ Compiler for Windows* CE .NET /CE (see Uninstall Previous Compilers).

License File

The Intel® C++ Compiler For Windows* CE 2.0.4 uses Macrovision Corporation's FLEXlm* electronic licensing technology. Before installing any component, the installer checks for a valid license. If there's none, you will be prompted to browse for a valid license file. The license file must be in place in order to use the compiler and debugger components.

The license file must have an extension "`.lic`".

The license directory is the location the environment variable `INTEL_LICENSE_FILE` points to (default `C:\Program Files\Common Files\Intel\Licenses\`). If it does not exist, the installation program will create it.

Electronic Download

If you have received the product by electronic download, the license will be sent to you via email. Please follow the instructions in the email to install the license file.

Product CD

If you have received the product on CD-ROM, a valid license is included on the CD and the installation program can locate it automatically. But, in order to obtain access to technical support and to be able to download and execute product updates, you must do the following:

1. **Register your product:** First, locate the serial number found on the inside flap of the product box. Then, visit the web site https://registrationcenter.intel.com and follow the instructions. After the registration you will receive an email within 24 hours containing a new license.
2. **Install the new license:** The new license in the email typically entitles you to one year of support services that allow you to download and execute product updates and obtain full technical support. The email also contains the instructions on how to install the license. Please follow the instructions to finish the new license installation.

Uninstall Previous Versions

You need to uninstall any previous Intel® C++ Compiler For Windows* CE prior to the Intel® C++ Compiler For Windows* CE 2.0.4 installation. Otherwise the installation will terminate and alert you to uninstall previous versions.

Choose `Add / Remove Programs` from the Windows* `Start / Control Panel` and click on the `Remove` button on any of the following or similar entries you may have installed on your system:

```
Intel® C++ Compiler 2.0.3 for eMbedded* Visual C++
Intel® C++ Compiler 2.0.3 for Platform Builder
Intel® C++ Compiler 2.0 for eMbedded* Visual C++
Intel® C++ Compiler 2.0 for Platform Builder
Intel® C++ Compiler For Microsoft* eMbedded* Visual C++*
Intel® C++ Compiler For Platform Builder for Microsoft Windows* CE .NET
```

Install Compiler

By default, compiler installation includes also installation of the corresponding debugging extensions, if not explicitly disabled in the `Custom Setup`.

1. Start the `autorun.exe` from the product CD or - if you downloaded the installation fileset from the web - run the self-extracting installation file `w_xwoem_*.exe`.
   The installation home page appears.
2. If you haven't registered your serial number yet, go to the registration link on the installation home page for product registration.
3. To install the **Compiler (including Debugging Extensions for JTAG) for Platform Builder for Windows* CE**:
   Click on the `Intel® C++ Compiler For Platform Builder for Microsoft Windows* CE` link on the left hand navigation bar of the installation home page.
   To install the **Compiler (including Debugging Extensions) for eMbedded Visual C++*/Visual Studio* 2005**:
   Click on the `Intel® C++ Compiler For Microsoft eMbedded Visual C++*` link on the left hand navigation bar of the installation home page.
4. Click on the `Install Now` button following all instructions.
5. Carefully read and accept the license agreement.
6. Optionally de-select sub-components in the `Custom Setup` dialog you DO NOT want to have installed (all available sub-components are selected by default).

Please refer also to the Default Installation Directories to learn where the components are being installed.

New Menu Entries in the Microsoft* Environments

The installation program of the Intel® C++ Compiler For Windows* CE 2.0.4 adds new menu items in the `Tools` menu of the Microsoft eMbedded Visual C++* / Visual Studio* 2005 and the Platform Builder for Windows* CE (NOT the Platform Builder for Windows Mobile* 5.0) environment as follows:

```
Intel(R) C++ Compiler Options
Select Compiler
Intel(R) C++ Compiler Help
```

Notes:

- The installation does NOT create compiler menu entries into the Platform Builder for Windows Mobile* 5.0 since this is a debug environment for Windows Mobile* 5.0 based platforms only. The code producing environment for Windows Mobile* is the Adaptation Kit with its own makefile build environment. See the compiler documentation for further information.
- The installation adds only the first two menu items for the compiler in the `Tools` menu of the *Microsoft* Platform Builder 5.0*, if both the Compiler and the Debugging Extensions are installed simultaneously. The third menu item `Intel(R) C++ Compiler Help` is not added due to restricted number of menu entries for external tools.

Default Installation Directories

The Intel® C++ Compiler For Windows* CE 2.0.4 installs default to:

Development Environment	Default Installation Directory for Intel® C++ Compiler

eMbedded Visual C++* 4.0 with Service Pack 1	`C:\Program Files\Microsoft eMbedded C++ 4.0 \EVC\WCE410\BIN`
eMbedded Visual C++* 4.0 with Service Pack 2	`C:\Program Files\Microsoft eMbedded C++ 4.0 \EVC\WCE420\BIN`
eMbedded Visual C++* 4.0 with Service Pack 4	`C:\Program Files\Microsoft eMbedded C++ 4.0 \EVC\WCE500\BIN`
Visual Studio* 2005	`C:\Program Files\Microsoft Visual Studio 8 \VC\ce\bin\x86_arm\`
Platform Builder for Windows* CE .NET 4.2	`C:\WINCE420\SDK\BIN\I386\ARM`
Platform Builder for Windows* CE 5.0	`C:\WINCE500\SDK\BIN\I386\ARM`
Adaptation Kit for Windows Mobile* 2003 Software	*Manual integration required. See Integrate Compiler / Windows Mobile* 2003 Software*
Platform Builder for Windows Mobile* 5.0	`C:\Program Files\Intel\Intel(R) C++ Compiler 2.0.4 for Platform Builder\PBM50` *(Additional steps are required to make the compiler work within the Windows Mobile* 5.0 environment. See Integrate Compiler / Windows Mobile* 5.0)*

where the name of the path, `C:\Program Files\` , may be different depending on your Windows* operating system.

Uninstall Compiler

Uninstall the Complete Product

To completely uninstall the Intel® C++ Compiler with corresponding Intel® Debugging Extensions from your computer go to `Add or Remove Programs` from the Windows* `Start / Control Panel.` Highlight the following you want to completely remove:

```
Intel(R) C++ Compiler 2.0.4 for eMbedded Visual C++
Intel(R) C++ Compiler 2.0.4 for Platform Builder
```

Click on the `Remove` button following all instructions.

Uninstall Individual Components

To uninstall components of the Intel® C++ Compiler and/or Intel® Debugging Extensions go to `Add or Remove Programs` from the Windows* `Start / Control Panel.` Highlight the following packages you want to uninstall components from:

```
Intel(R) C++ Compiler 2.0.4 for eMbedded Visual C++
Intel(R) C++ Compiler 2.0.4 for Platform Builder
```

Click on the `Change` button.
Click on the `Next` button.
Check the `Modify` checkbox from the `Program Maintenance` dialog.
From the `Custom Setup` dialog deselect the Compiler/Debugging Extensions you want to uninstall.

This will re-install the components you selected to install and remove components you deselected.

Integrate Compiler into Microsoft* Adaptation Kits

After installation of the Compiler the integration into Microsoft* Adaptation Kits has to be performed in a separate process as described below.

Windows Mobile* 2003 Software (Pocket PC*, Smartphone*)

There is no setup to integrate the compiler into Adaptation Kits. The integration has do be done manually as follows:

1. Copy Intel® C++ Compiler files from the Platform Builder* SDK folder to the Adaptation Kit SDK folder.
2. Customize the build script `makefile.def` from the Adaptation Kit SDK

Copy Intel® C++ Compiler files from the Platform Builder* SDK directory to the Adaptation Kit SDK directory

Copy the files

```
asxscce.exe
ccxscce.cfg
ccxscce.exe
ccxsccet.cfg
ccxsccet.exe
isacc42.cfg
isacc42.exe
isacc42a.cfg
mcpcomce.exe
txicl42.exe
x0__ar10.lib
xicl42.exe
xilink42.exe
```

from source directory <*PB_Root*>`\sdk\bin\i386` to target directory <*AK_Root*>`\sdk\bin\i386`

<*PB_Root*> is the path depending on the installation location of the Platform Builder*. The default directory is `C:\WINCE420`
<*AK_Root*> is the path depending on the location of the Adaptation Kit Integration. The default directories are `C:\PGrine\` (for Pocket PC*) and `C:\SLove\(`for Smartphone*).

Customize the Build Script

***Note**:* Make a backup copy of the `makefile.def` prior to any modification! You need the original version if you want to switch back to using the Microsoft* compiler.

To enable the Intel® C++ Compiler instead of the Microsoft* C++ Compiler the following changes are necessary in <*AK_Root*>`public\common\oak\misc\makefile.def:`

- Replace `HOSTCOMPILER=clarm` with `HOSTCOMPILER=ccxscce`
- Remove the line `CFLAGS=$(CFLAGS) -WX`
  (Background: The Intel® C++ Compiler performs warning checks more restrictively and -WX treats any warning as an error. This would cause the build process to fail if the -WX option is set.)
- Replace `CCOMPILER=clarm -nologo` with `CCOMPILER=ccxscce -nologo`
  Optionally set additional Intel® C++ Compiler options such as /O3, /QTPxsc3 (please refer to the compiler documentation)
- Replace `ASSEMBLER=armasm -coff` with `ASSEMBLER=asxscce -mcpu 3 -coff`

- Search for the end of the TARGETLIBS list generation and add the following line outside of all IF-ELSE-ENDIF constructs:
  TARGETLIBS=$(_SDKROOT)\bin\i386\x0__ar10.lib $(TARGETLIBS)
  (Background: The Intel® C++ compiler uses an additional runtime library, x0__ar10.lib, which must be specified as the very first entry in the linker library list)

Windows Mobile* 5.0

If the Intel® C++ Compiler for Platform Builder for Microsoft* Windows Mobile* 5.0 was selected in the Custom Setup of the compiler installation, a base directory

```
C:\Program Files\Intel\Intel(R) C++ Compiler 2.0.4 for Platform Builder\PBM50
```

and Windows* Start menu entries

```
All Programs
   Intel(R) Software Development Tools
      Intel(R) C++ Compiler Build Support for Adaptation Kits 2.0
         Add Adaptation Kit Integration
         Remove Adaptation Kit Integration
         Select Adaptation Kit Compiler
```

were created to integrate the compiler into Adaptation Kits.

Please refer to chapter 3 'Building a Microsoft* Windows Mobile* 5.0 Platform' of the Compiler documentation (see Documentation) for usage of the Intel® C++ Compiler Build Support for Adaptation Kits.

Note: This integration has to be performed for all of the Adaptation Kits installed on your computer which shall use the Intel® C++ Compiler.

Known Limitations

Restrictions in C/C++ Language Support

The ISO/IEC 9899:1999 (E) features are not fully supported

Resolution of duplicate symbols in runtime libraries

In debug builds, linking the Intel runtime library x0__ar10.lib before the Microsoft libraries may lead to an ambiguity in symbol resolution if the same symbol is found in both libraries.
If the symbol is **first** found in the Intel library and later inadvertently included from a Microsoft library, you can resolve the "duplicate symbol" error by adding the option /FORCE:MULTIPLE to the link-options in C:\WINCE500\PUBLIC\COMMON\OAK\MISC\makefile.def

Change:

```
!IF "$(RESOURCEONLYDLL)" == "1"
DEBUG_LINK_CMD_LINE=
!ELSEIF "$(WINCEPROFILE)" == "1"
DEBUG_LINK_CMD_LINE=-debug -debugtype:both -incremental:no
```

```
CDEBUG_DEFINES=$(CDEBUG_DEFINES) -DWINCEPROFILE
!ELSE
DEBUG_LINK_CMD_LINE=-debug -debugtype:cv -incremental:no
!ENDIF
```

to

```
!IF "$(RESOURCEONLYDLL)" == "1"
DEBUG_LINK_CMD_LINE=/FORCE:MULTIPLE
!ELSEIF "$(WINCEPROFILE)" == "1"
DEBUG_LINK_CMD_LINE=-debug -debugtype:both -incremental:no
CDEBUG_DEFINES=$(CDEBUG_DEFINES) -DWINCEPROFILE /FORCE:MULTIPLE
!ELSE
DEBUG_LINK_CMD_LINE=-debug -debugtype:cv -incremental:no /FORCE:MULTIPLE
!ENDIF
```

As a result the **first** occurrence of the symbol will prevail over later occurrences.

Breakpoints not activated

Breakpoints remain inactive when debugging some C++ DLLs as part of the Windows* CE operating system build. Once you managed to stop in the DLL code (e.g. by using a `DebugBreak()` statement) it is however possible to single step through the code.

Compiler does not build Intel® IPP samples

The problem is due to a linker path, which is added, and contains spaces. As a workaround you may use the library including the path, and set it in quotes (`" "`) or use the short name for the setting.

Unresolved Externals in Visual Studio* 2005 Projects

Under the Microsoft Visual Studio* 2005 environment, the usage of the Intel® C++ Compiler may lead to an `unresolved external` error when specific runtime libraries are missing in the project settings or option `/Za` (standard C language) is not being used while `OLDNAMES.lib` is not being excluded.

The compiler runtime library options which require specific runtime libraries are defined under `Projects / Properties / Configuration Properties / C/C++ / Code Generation / Runtime Library`. Depending on the option setting add the corresponding runtime library under `Projects / Properties / Configuration Properties / Linker / Additional Dependencies` as follows:

Compiler Option used	Runtime Library to be included
/MD	`MSVCRT.lib`
/MDd	MSVCRTd.lib
/MT	LIBCMT.lib (default setting)
/MTd	LIBCMTd.lib

An unresolved external error also may occur when the compiler option `/Za` (strict standard C language usage) is not being set and the `OLDNAMES.lib` library is not being included under `Projects / Properties / Configuration Properties / Linker / Ignore Specific Library`

In this case add the `OLDNAMES.lib` under `Projects / Properties / Configuration Properties / Linker / Ignore Specific Library`

You can also use the `#pragma comment(linker," ")` on source code level to include/exclude libraries, for example:

```
#pragma comment(linker, "/defaultlib:libcmt.lib")
#pragma comment(linker, "/disallowlib:oldnames.lib")
```

Character / Wide String Literals don't have the Type of array of `N const char/wchar_t`

This is a violation of the ANSI C++ Standard ISO/IEC 14882 (1998) ch. 2.13.4.
As a result the incorrect overloaded function is called:

```
long func(wchar_t *p) { /* 1 */
... }
long func(const wchar_t* p) { /* 2 */
... }
func(L"1"); /* should call 2 but calls 1 */
```

Argument Dependent Name Lookup not Supported

This is a violation of the ANSI C++ Standard ISO/IEC 14882 (1998) ch. 3.4.2.
As a result an error message is issued:

```
namespace myNamespace {
struct myStruct {
...
};
int myFunc() {
... };
}
int func() {
myNamespace::myStruct X;
myFunc(X); // => error: identifier "myFunc" is undefined
```

Corrupt Symbol Table Generated by Assembler

The Assembler generates incomplete section headers in some cases. The result are undefined static symbols.

Limitations on C Structured Exception Handling

`Try` in an `except` or a `finally` block causes an exception to be caught within the first `except` block.

Nested `finally` constructs are not supported with the Intel® C++ Compiler: In case of need you may use subroutines.

```
void func1 (void)
{
    __try {
…
    }
    __finally {
```

```
…       /* exception will be caught in this block */
        __try { /* an exception thrown in this try block will only be caught in the top level block*/
…
        }
        __finally {
…                   /* exception will not be caught here */
        }
…
    }
}
```

As workaround use the following Instead:

```
void sub()
{
    __try {
…
    }
    __finally {
…
    }

}

void func1(void)
{
    __try {
…
    }
    __finally {
…
        sub();
    }
}
```

Intel® Run-time Library DLL required on target when dynamically linked (e.g. using `/O1`)

The Intel® Run-time Library dll `x0__ar10dll.dll` (ARM* mode) and/or `x0__ar10dllt.dll` (Thumb* mode) is a requirement on the target when dynamically linked. This especially happens when code is produced with the compiler code size optimization switch `/O1`. Therefore this library needs to be explicitly copied to the same location on the target as the calling executable.

`__fastcall` and `__stdcall` Calling Conventions are ignored

The Intel® C++ Compiler doesn't support `__fastcall` and `__stdcall` calling conventions. It ignores these keywords without any warning and defaults to the default calling convention.

Issues with Using Compiler Option `/O3`

The Intel® C++ Compiler option `/O3` cannot be used directly when using the default project settings from the Microsoft* environment.

Workaround:

Remove any occurrence of options `/Od /O1, /O2, /Os, Ot` from the project settings and set option `/O3` in the `Intel(R) C++ Compiler Options` dialog.

Using Compiler Options `/GX` and `/GR`

The Intel® C++ Compiler option `/GX` can be used together with `/GR` only if C++ Exception Handling is used. Using option `/GX` without `/GR` may result in unexpected error messages.

No 64-bit WMMX register loads

The compiler doesn't generate a 64-bit register load. It generates 2 loads in 32 bits ARM registers followed by a transfer instruction.

Within the Microsoft Windows* CE environment neither stack nor data are guaranteed to be 8 byte aligned. Only 4 byte alignment is enforced. Especially for callback-functions the behavior cannot be calculated. Therefore the enabling of the access will improve performance, but causes runtime-problems.

Intel® Assembler generates incorrect warning

The Intel® Assembler generates a warning, if a function exceeds 64K:
`BUILD: [01:0000006171:INFO ] ASXSCCE-W-WARNING[0017]:C:\DOCUME~1\<user>\LOCALS~1\Temp\<tempname>.asm:37516:Truncating value to 16 LSB`

This warning is incorrect and can be ignored:

Indirect Recursion in Macros

Indirect recursion in macros is not detectable yet, it may cause the Intel® Assembler to go into an infinite loop.

Cross-references in listings

The Intel® Assembler listing does not contain cross-references.

Incompatibility of Assembler options `-list` and `-j`

If options `-list` and `-j` are set an empty list file is generated.

Missing debug line information

Debug line information is missing if a line containing the closing brace "}" of a function immediately follows a return statement.

Example:

```
test(){
    if (x)
         return 0;
         } // debug line info will not be generated here
```

Header/Libraries Re-build

If you create a new platform/project under Microsoft eMbedded Visual C++* 4.0 / Visual Studio* 2005, a new set of headers and libraries is generated. If you have changed headers and libraries manually, you have to redo these changes each time you create a new platform/project.

Intel® Run-time Library

The compiler adds the Intel® Run-time Library x0__ar10.lib as default library. If default libraries are disabled and if the Intel® C++ Compiler is used, the Intel® Run-time Library has to be added manually. The Intel® Run-time Library contains the run-time support functions as well as math functions and some highly optimized string and memory functions. The run-time support functions are always taken from the Intel® Run-time Library; the other functions are either taken from the Windows* CE libraries or from the Intel® Run-time Library, whichever is placed first. To make sure that the Intel Run-time Library is used, place the library in the linker project settings at the first place.

Inconsistencies in header paths

There may be some inconsistencies with the header paths. Each time a new platform is generated, the system is generating a new set of libraries and headers. Since the Microsoft* installation process cannot be changed, the new library and include paths may have to be added manually.

Under eMbedded Visual C++*

Under eMbedded Visual C++* these headers can be entered project-related under:
`Project / Settings / C/C++ / Category / Preprocessor / Additional Include Directories (header)`
(e.g.: c:\Program Files\Windows CE Tools\wce410\STANDARDSDK_410\Include\Armv4i)
and
`Project / Settings / Link / Category / Input / Additional Library Path (libraries).`

Under Platform Builder

These entries can be found globally under
`Tools / Options / Directories / Show Directories / Include Files`
or
`Tools / Options / Directories / Show Directories / Library Files`
or project-related under
`Project / Settings / C/C++ / Category / Preprocessor / Additional Include Directories (header)`
and
`Project / Settings / Link / Category / Input / Additional Library Path (libraries)`

ASCII Support using `stdlib.h`

For Platform Builder 4.2 and eMbedded Visual C++* 4.0 only: There might be problems with ASCII support for Windows* CE / CE .NET using the Microsoft* header `stdlib.h`. Some functions, e.g., `isalpha(), isupper()`, may deliver partly incorrect results when used with characters greater 127, and with character 9. You can correct these problems by replacing parts of `stdlib.h`.

Please replace the following lines in `stdlib.h`

```
#define isalpha(_c)    (_isctype(_c,_ALPHA) )
#define isupper(_c)    (_isctype(_c,_UPPER) )
#define islower(_c)    (_isctype(_c,_LOWER) )
#define isdigit(_c)       (_isctype(_c,_DIGIT) )
#define isxdigit(_c)     (_isctype(_c,_HEX) )
#define isspace(_c)      (_isctype(_c,_SPACE)
#define ispunct(_c)      (_isctype(_c,_PUNCT) )
```

```
#define isalnum(_c)      (_isctype(_c,_ALPHA|_DIGIT) )
#define isprint(_c)      (_isctype(_c,_BLANK|_PUNCT|_ALPHA|_DIGIT))
#define isgraph(_c)      (_isctype(_c,_PUNCT|_ALPHA|_DIGIT) )
#define iscntrl(_c)      (_isctype(_c,_CONTROL) )
#define __isascii(_c)    ((unsigned)(_c) < 0x80)
#define isascii          __isascii

#define iswalpha(_c)     ( iswctype(_c,_ALPHA) )
#define iswupper(_c)     ( iswctype(_c,_UPPER) )
#define iswlower(_c)     ( iswctype(_c,_LOWER) )
#define iswdigit(_c)     ( iswctype(_c,_DIGIT) )
#define iswxdigit(_c)    ( iswctype(_c,_HEX) )
#define iswspace(_c)     ( iswctype(_c,_SPACE) )
#define iswpunct(_c)     ( iswctype(_c,_PUNCT) )
#define iswalnum(_c)     ( iswctype(_c,_ALPHA|_DIGIT) )
#define iswprint(_c)     ( iswctype(_c,_BLANK|_PUNCT|_ALPHA|_DIGIT)) )
#define iswgraph(_c)     ( iswctype(_c,_PUNCT|_ALPHA|_DIGIT) )
#define iswcntrl(_c)     ( iswctype(_c,_CONTROL) )
#define iswascii(_c)     ( (unsigned)(_c) < 0x80 )
#define isleadbyte(_c)   ( IsDBCSLeadByte(_c))
```

with the lines below:

```
static _CRTIMP int __cdecl _isctype1(int c, int t)
{
  if ((c & 0x80) == 0){
       if ((c == 0x09)&&( t == (_BLANK|_PUNCT|_ALPHA|_DIGIT)))
            return(0);
       return(_isctype(c,t));
 }
 else {
        return(0);
      }
 }
static _CRTIMP int iswctype1 (wint_t c, wctype_t t){
  if ((c & 0x80) == 0){
       if ((c == 0x09)&&( t == (_BLANK|_PUNCT|_ALPHA|_DIGIT)))
            return(0);
       return(iswctype (c,t));
 }
 else {
        return(0);
      }
 }

#define isalpha(_c)      (_isctype1(_c,_ALPHA) )
#define isupper(_c)      (_isctype1(_c,_UPPER) )
#define islower(_c)      (_isctype1(_c,_LOWER) )
#define isdigit(_c)      (_isctype1(_c,_DIGIT) )
#define isxdigit(_c)     (_isctype1(_c,_HEX) )
#define isspace(_c)      (_isctype1(_c,_SPACE) )
#define ispunct(_c)      (_isctype1(_c,_PUNCT) )
#define isalnum(_c)      (_isctype1(_c,_ALPHA|_DIGIT) )
#define isprint(_c)      (_isctype1(_c,_BLANK|_PUNCT|_ALPHA|_DIGIT))
#define isgraph(_c)      (_isctype1(_c,_PUNCT|_ALPHA|_DIGIT) )
#define iscntrl(_c)      (_isctype1(_c,_CONTROL) )
```

```
#define __isascii(_c)    ((unsigned)(_c) < 0x80)
#define isascii          __isascii

#define iswalpha(_c)     ( iswctype1(_c,_ALPHA) )
#define iswupper(_c)     ( iswctype1(_c,_UPPER) )
#define iswlower(_c)     ( iswctype1(_c,_LOWER) )
#define iswdigit(_c)     ( iswctype1(_c,_DIGIT) )
#define iswxdigit(_c)    ( iswctype1(_c,_HEX) )
#define iswspace(_c)     ( iswctype1(_c,_SPACE) )
#define iswpunct(_c)     ( iswctype1(_c,_PUNCT) )
#define iswalnum(_c)     ( iswctype1(_c,_ALPHA|_DIGIT) )
#define iswprint(_c)     ( iswctype1(_c,_BLANK|_PUNCT|_ALPHA|_DIGIT)) )
#define iswgraph(_c)     ( iswctype1(_c,_PUNCT|_ALPHA|_DIGIT) )
#define iswcntrl(_c)     ( iswctype1(_c,_CONTROL) )
#define iswascii(_c)     ( (unsigned)(_c) < 0x80 )
#define isleadbyte(_c)   ( IsDBCSLeadByte(_c))
```

No "All Users" installation under eMbedded Visual C++* 4.0

It is not possible to install the Intel® C++ Compiler under Microsoft eMbedded Visual C++* 4.0 with a "All users" profile.

Workaround:

1. Start eVC++ once which configures the user-dependent tools entries.
2. Install the Intel® C++ Compiler (see Installation) to make your user dependent entries valid for your user profile.

If more users should work on a machine and therefore have their own profiles respectively, any subsequent compiler installation after the first one has to be performed as a "Repair" installation.

Documentation

The Compiler documentation is available from `Tools / Intel(R) C++ Compiler Help` of the Microsoft* Platform Builder or eMbedded Visual C++* / Visual Studio* 2005 environments.

The Compiler documentation as well as the Assembler Reference Manual help files `*.chm` are located in the directories as follows:

Documentation	Environment	Default Installation Directory
`ccxscce.chm` Compiler `asxscce.chm` Assembler On-line Help files	eMbedded Visual C++* 4.0 (any Service Pack)	`C:\Program Files\Microsoft eMbedded C++ 4.0\cepb\help`
	Visual Studio* 2005	`C:\Program Files\Microsoft Visual Studio 8\VC\ce\bin\x86_arm`
	Platform Builder for Windows* CE .NET 4.2	`C:\WINCE\420\SDK\BIN\I386\ARM`
	Platform Builder for Windows* CE 5.0	`C:\WINCE500\SDK\BIN\I386\ARM`

	Platform Builder for Windows Mobile* 5.0	`C:\Program Files\Intel\Intel(R) C++ Compiler 2.0.4 for Platform Builder\PBM50`

where the name of the path, `C:\Program Files\`, may be different on your Windows* operating system.

Documentation Addendum

This section discusses useful additions or topics missing in the current product documentation.

Usage of Intel® Wireless MMX™ Technology Intrinsics

To use Intel® Wireless MMX™ technology intrinsics you need to include
`#include "mmintrin.h"`
to use the intrinsics.

Additonally the compiler option `/QTPxsc3` must be set in order to enable intrinsics instructions.

Intel Compiler Macros `__INTEL_COMPILER, __ISACC and __XSCALE__`

The Intel compiler defines the macros `__INTEL_COMPILER` and `__ISACC` which are useful for conditional compilation and for identification of which compiler is being used at compile time. The macro `__XSCALE__` may be used to identify the target type.

Documentation Errata

There are the following errors in the product documentation (see Product Contents):

Assembler Option `-abi[=`*`spectype`*`]`

The Assembler option `-abi[=`*`spectype`*`]` as referenced in the Intel® Assembler Reference Manual is not supported for the Intel® C++ Compiler For Windows* CE.

Wrong Syntax for Intrinsics `_mm_align_si64(), _mm_getwcx(), _mm_setwcx()`

The *Intel® Wireless MMX™ Technology General Support Intrinsics* documentation section of the Intel® C++ Compiler User's Manual describes the following functions incorrectly:

```
__m64 _mm_align_si64(__m64 m1, __m64 m2, int count)
```
should read
```
__m64 _mm_align_si64(__m64 m1, __m64 m2, int count)

int __mm_getwcx(int number)
```
should read
```
int _mm_getwcx(int number)

__mm_setwcx(int number)
```
should read
```
void _mm_setwcx(int number)
```

As for all intrinsics there is only one underscore in front of the function name.

ActiveX Bug Workaround

When opening help topics that contain Related Topics button links, you may see an Internet Explorer* warning message that reads: "An ActiveX control on this page might be unsafe to interact with other parts of the page. Do you want to allow this interaction?". You can safely click "Yes" to continue.
This problem occurs due to registry errors caused by installing a Windows* Service pack.
To avoid seeing the warning, you need to register and then reregister the HTML Help ActiveX control. To do this, open a command prompt and type:

```
regsvr32 /u %windir%\system32\hhctrl.ocx
regsvr32 %windir%\system32\hhctrl.ocx
```

Technical Support

Registration

To receive technical support for this product and product updates, you need to be registered for an Intel® Premier Support account on our secure web site, https://premier.intel.com/. If not yet done, please register your product at http://www.intel.com/software/products/registrationcenter/.

If you cannot register your product or you cannot access your account, please contact https://registrationcenter.intel.com/support/contact.aspx and submit your problem.

Note: If your distributor provides technical support for this product, please contact them for support rather than Intel.

Startup Support

For initial startup support such as installation, licensing, support account issues, please visit https://registrationcenter.intel.com/support/contact.aspx.

Product Support

If you need help or if have problems with the product, submit your issues via the Intel® Premier Support at https://premier.intel.com.

Steps to submit an issue:

1. Go to https://premier.intel.com/.
2. Type in your Login and Password. Both are case-sensitive.
3. Click the "`Submit`" button.
4. Read the Confidentiality Statement and click the "`I Accept`" button.
5. Click on the "`Go`" button next to the "`Product`" drop-down list.
6. Click on the "`Submit Issue`" link in the left navigation bar.
7. Choose "`Development Environment (tools,SDV,EAP)`" from the "`Product Type`" drop-down list.
8. If this is a software or license-related issue, choose "`Intel C++ Compiler, Windows* CE, Pro`" from the "`Product Name`" drop-down list.
9. Enter your question and complete the fields in the windows that follow to successfully submit the issue.

Guidelines for problem report or product suggestion:

1. Describe your difficulty or suggestion.
   For problem reports please be as specific as possible, so that we may reproduce the problem. For compiler problem reports, please include the compiler options and a small test case if possible.
2. Describe your system configuration information.
   Run the compiler (`ccxscce`) from the command window and provide the compiler version.
3. Copy the "Package ID" (e.g. `w_xwoem_pc_2.0.xxx`) into the corresponding Premier Support field. Please include any other specific information that may be relevant to helping us to reproduce and address your concern.
4. If you were not able to install the compiler or cannot get the Package ID, enter the filename you downloaded as the package ID.

General Support Information

For information about the Intel C++ Compiler's Users Forums, FAQ's, tips and tricks, and other support information, please visit: http://support.intel.com/support/performancetools/xsc/cpb/index.htm.

For general support information please visit http://www.intel.com/software/products/support/.

Additional Information

Related Products and Services

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

Some of the related products include:

- The Intel® Compilers for Embedded Application Development are an important part of making software run at top speeds with full support for the latest Intel XScale® application processors.
- The Intel® C++ and Fortran Compilers are an important part of making software run at top speeds with full support for the latest Intel IA-32 and Itanium® processors.
- The VTune™ Performance Analyzer enables you to evaluate how your application is utilizing the CPU and helps you determine if there are modifications you can make to improve your application's performance.
- The Intel® Performance Library Suite provides a set of routines optimized for various Intel processors. The Intel® Math Kernel Library, which provides developers of scientific and engineering software with a set of linear algebra, fast Fourier transforms and vector math functions optimized for the latest Intel Pentium® and Intel Itanium processors. The Intel® Integrated Performance Primitives consists of cross-platform tools to build high performance software for several Intel architectures and several operating systems.
- The Intel® Software College provides training for developers on leading-edge software development technologies. Training consists of online and instructor-led courses covering all Intel architectures, platforms, tools, and technologies.

Disclaimer and Legal Information

The information in this manual is subject to change without notice and Intel Corporation assumes no responsibility or liability for any errors or inaccuracies that may appear in this document or any software that may be provided in association with this document. This document and the software described in it are furnished under license and may only be used or copied in accordance with the terms of the license. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted by this document.

The information in this document is provided in connection with Intel products and should not be construed as a commitment by Intel Corporation.

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. Intel products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

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 software described in this document may contain software defects which may cause the product to deviate from published specifications. Current characterized software defects are available on request.

Intel, the Intel logo, Intel SpeedStep, Intel NetBurst, Intel NetStructure, MMX, i386, i486, Intel386, Intel486, Intel740, IntelDX2, IntelDX4, IntelSX2,Celeron, Intel Centrino, Intel Xeon, Intel XScale, Itanium, Pentium, Pentium II Xeon, Pentium III Xeon, Pentium M, and VTune are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

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

Copyright © Intel Corporation 2004-2006.