WINDOWS EMBEDDED COMPACT 7.0

Authors: Luc Perneel (1, 2), Hasan Fayyad-Kazan (2) and Martin Timmerman (1, 2, 3)

1: Dedicated Systems Experts, 2: VUB-Brussels, 3: RMA-Brussels

Disclaimer

Although all care has been taken to obtain correct information and accurate test results, Dedicated Systems Experts, VUB-Brussels, RMA-Brussels and the authors cannot be liable for any incidental or consequential damages (including damages for loss of business, profits or the like) arising out of the use of the information provided in this report, even if these organisations and authors have been advised of the possibility of such damages.

http://www.dedicated-systems.com

E-mail: info@dedicated-systems.com

EVALUATION REPORT LICENSE

This is a legal agreement between you (the downloader of this document) and/or your company and the company DEDICATED SYSTEMS EXPERTS NV, Diepenbeemd 5, B-1650 Beersel, Belgium.
It is not possible to download this document without registering and accepting this agreement on-line.

1. **GRANT**. Subject to the provisions contained herein, Dedicated Systems Experts hereby grants you a non-exclusive license to use its accompanying proprietary evaluation report for projects where you or your company are involved as major contractor or subcontractor. You are not entitled to support or telephone assistance in connection with this license.
2. **PRODUCT**. Dedicated Systems Experts shall furnish the evaluation report to you electronically via Internet. This license does not grant you any right to any enhancement or update to the document.
3. **TITLE**. Title, ownership rights, and intellectual property rights in and to the document shall remain in Dedicated Systems Experts and/or its suppliers or evaluated product manufacturers. The copyright laws of Belgium and all international copyright treaties protect the documents.
4. **CONTENT**. Title, ownership rights, and an intellectual property right in and to the content accessed through the document is the property of the applicable content owner and may be protected by applicable copyright or other law. This License gives you no rights to such content.
5. **YOU CANNOT**:
   - You cannot, make (or allow anyone else make) copies, whether digital, printed, photographic or others, except for backup reasons. The number of copies should be limited to 2. The copies should be exact replicates of the original (in paper or electronic format) with all copyright notices and logos.
   - You cannot, place (or allow anyone else place) the evaluation report on an electronic board or other form of on line service without authorization.
6. **INDEMNIFICATION**. You agree to indemnify and hold harmless Dedicated Systems Experts against any damages or liability of any kind arising from any use of this product other than the permitted uses specified in this agreement.
7. **DISCLAIMER OF WARRANTY**. All documents published by Dedicated Systems Experts on the World Wide Web Server or by any other means are provided "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. This disclaimer of warranty constitutes an essential part of the agreement.
8. **LIMITATION OF LIABILITY**. Neither Dedicated Systems Experts nor any of its directors, employees, partners or agents shall, under any circumstances, be liable to any person for any special, incidental, indirect or consequential damages, including, without limitation, damages resulting from use of OR RELIANCE ON the INFORMATION presented, loss of profits or revenues or costs of replacement goods, even if informed in advance of the possibility of such damages.
9. **ACCURACY OF INFORMATION**. Every effort has been made to ensure the accuracy of the information presented herein. However Dedicated Systems Experts assumes no responsibility for the accuracy of the information. Product information is subject to change without notice. Changes, if any, will be incorporated in new editions of these publications. Dedicated Systems Experts may make improvements and/or changes in the products and/or the programs described in these publications at any time without notice. Mention of non-Dedicated Systems Experts products or services is for information purposes only and constitutes neither an endorsement nor a recommendation.
10. **JURISDICTION**. In case of any problems, the court of BRUSSELS-BELGIUM will have exclusive jurisdiction.

Agreed by downloading the document via the internet.

DOCUMENT CHANGE LOG

Issue No.	Revised Issue Date	Para's / Pages Affected	Reason for Change
1.00	May 27, 2011	All	Initial Issue: vendor revision draft
1.01		All	Adapted some phrasings
2.0	Sept 13, 2011	ALL	Almost Final
2.10	Sept 19, 2011	ALL	Add fonts
3	Feb 2, 2011	ALL	Modify contents based on MS comments
3.1	Jun 6, 2012		Add MS comments

1 Document Intention

1.1 Purpose and scope

This document presents the qualitative evaluation results of the **Windows Embedded Compact 7** operating system. The testing results of this operating system employed on an x86 processors can be found on our website. (www.dedicated-systems.com).

The layout and the content of this report follow the one depicted in "The evaluation test report definition" [Doc. 3] and "The OS evaluation template" [Doc. 4]. See section 1.4 of this document for more detailed references. Therefore these documents have to be seen as an integral part of this report!

Due to the tightly coupling between these documents, the framework version of "The evaluation test report definition" has to match the framework version of this evaluation report (which is 2.9). More information about the documents and tests versions together with their corresponding relation can be found in "The evaluation framework" see [Doc. 1] in section 1.4 of this document.

1.2 Document issue: the 2.9 framework

This document shows the results in the scope of the evaluation framework 2.9.

1.3 Conventions

Throughout this document, we use certain typographical conventions to distinguish technical terms. Our used conventions are the following:

- ***Bold Italic*** for OS Objects
- **Bold** for Libraries, packets, directories, software, OSs...
- `Courier New` for system calls (APIs...)

1.4 Related documents

These are documents that are closely related to this document. They can all be downloaded using following link:
http://www.dedicated-systems.com/encyc/buyersguide/rtos/evaluations

Doc. 1 The evaluation framework
This document presents the evaluation framework. It also indicates which documents are available, and how their name giving, numbering and versioning are related. This document is the base document of the evaluation framework.
EVA-2.9-GEN-01 Issue: 1 Date: 10/8/2004

Doc. 2 What is a good RTOS?
This document presents the criteria that Dedicated Systems Experts use to give an operating system the label “Real-Time”. The evaluation tests are based upon the criteria defined in this document.
EVA-2.9-GEN-02 Issue: 1 Date: 11/6/2001

Doc. 3 The evaluation test report definition
This document presents the different tests issued in this report together with the flowcharts and the generic pseudo code for each test. Test labels are all defined in this document.
EVA-2.9-GEN-03 Issue: 1 Date: 10/8/2004

Doc. 4 The OS evaluation template
This document presents the layout used for all reports in a certain framework.
EVA-2.9-GEN-04 Issue: 1 Date: 10/8/2004

Doc. 5 Windows Embedded Compact 7 on an x86 platform.
This document presents the results of the evaluation tests done for Windows Embedded Compact 7 on an x86 board (Pentium-II 233MHz).
EVA-2.9-TST-CE-x86-….. Issue: 1 Date: TBD

2 Introduction

2.1 Overview

Releasing a new OS with a different name (changed from **Windows CE** to **Windows Compact 7**) does not mean that we are up with a new OS! Such naming change was mainly done for marketing purposes, as there were no fundamental changes in the OS itself!

Luckily, Microsoft continued using the same criteria for numbering its new releases. In this case, the current release that we are evaluating (**Compact 7**) can be considered as being **CE 7.0** version. This step was a lucky shot from Microsoft, as their current desktop operating system (**Windows 7**) is getting a positive feedback from the market, so they used the same version number for their embedded OS release.

A more confusing fact is the existence of a similar named OS called **Windows Embedded Standard 7**. However, this **Standard 7** OS has nothing in common with **Compact 7**. It is a special version of the **Windows 7** desktop operating system and as such it has nothing to do with real time and so cannot be used for real-time purposes!

Further in the document, the full name "**Windows Embedded Compact 7**" or the short name "**Compact 7**" will be used.

2.1.1 Windows Embedded Compact 7

Fundamentally, looking in the OS internals, these are the major changes between **Compact 7** and its predecessor **CE6R3**:

- **Compact 7** kernel supports now multi-core architectures and thus has SMP functionalities.
- **Compact 7** now supports up to 3GB physical RAM (which was limited to 512MB in CE6R3).
- Microsoft has also redesigned their heap manager to reduce memory fragmentation compared with **CE6**. This is surely important for long time running devices (without reboot), which is typical for embedded systems.
- The possibility to use Address Space Location Randomization: this will adapt randomly the addresses of the DLL loaded functions to the application. As such, a hacker cannot know how to jump to a library routine.

Beside these low-level changes, there were some improvement in the cryptographic support, and a new security loader was created for verifying installed modules (keys, certification…).

Since our previous review of the **CE6R2** version, we see that only few changes in **Compact 7** were performed on the internal behavior (workings) of the RTOS while most of the changes were performed on the application level and tool integration as we can see here:

- **Silverlight**, the new development platform for GUIs (being it web, desktop or smart phone).
- Newer versions of browsers
- Upgrade in (mobile) Office products
- Support for **PDF**
- Multi-touch

Furthermore, the **Platform Builder** toolset uses now **Visual studio 2008 SP1**, where the previous release used **Visual studio 2005**.

Microsoft claims as well that their tools are improved; they mention as example their new remote toolset to access/monitor remote **Compact 7** devices. We will see further in this document what our experiences with these tools are.

2.2 Evaluated (RTOS) product

The RTOS that will be evaluated and tested is **Windows Embedded Compact 7**. This OS was launched by Microsoft Corporation at the beginning of 2011. In fact, this OS "**Windows Embedded Compact**" is the successor of **Windows CE6R3**.

The tests for evaluating this OS were done in March 2011 which is the date when this OS was released as a manufacture release.

Installing **Windows Embedded Compact 7** required the usage of the following products

- **Visual Studio 2008 SP1**
- **.NET Framework 3.5**
- **Windows Embedded Compact 7**

2.3 Supported CPU

The following Instruction set families are supported by **Windows Embedded Compact 7:**

- ARM
- MIPS
- x86

Remark that the SH4 support has been dropped with this release.

3 Evaluation results summary

Following is a summary of the results of evaluating **Windows Embedded Compact 7**, released by Microsoft Corporation, Inc.

3.1 Positive points

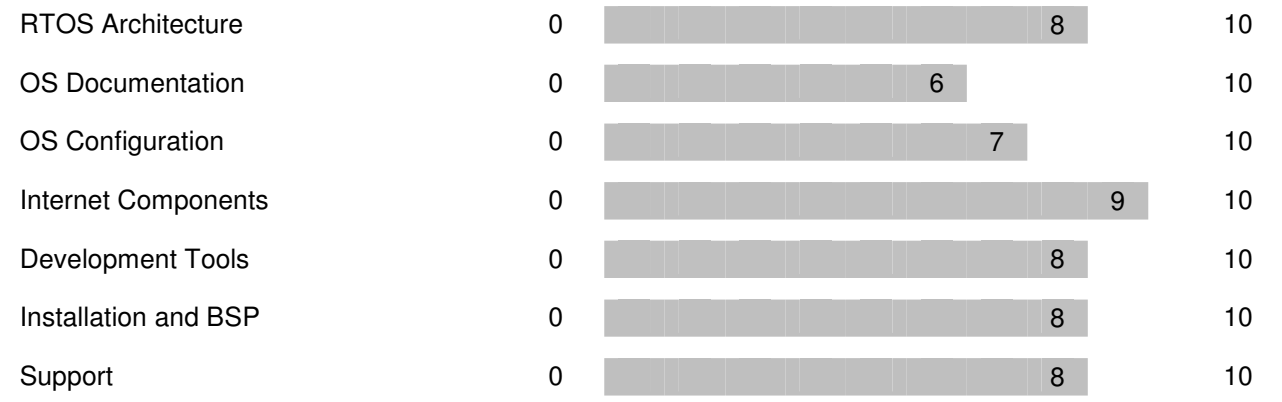
1) All protection primitives use priority inheritance, which is a major plus for achieving real-time behavior
2) Good debugging tools: Available also for kernel/driver debugging.
3) Very easy to install and to set-up a target (from templates).
4) Provides the same flexibility as a 32-bit general purpose OS

3.2 Negative points (see Microsoft's comments in section 3.4)

1) The operating system documentation has taken a step backwards compared with the previous versions. A lot of background information is removed (***see MS comments***).
2) Customizing the kernel and adding custom drivers (BSP) stays a daunting task once you go away from the default configurations.
3) The remote tool has been changed since last version. We noticed two issues, the more important of which is that there is no officially-supported method to include the remote tools within a device image using Platform Builder. Additionally, we noticed during our testing that establishing a connection between the tools and the target took in excess of a minute, which was longer than our expectation (***see MS comments***).

3.3 Ratings

For a description of the ratings, see [Doc. 3].



http://www.dedicated-systems.com
Email: info@dedicated-systems.com

3.4 Vendor Comments

Following are the comments of Microsoft on the negative points:

- **For point 1** Microsoft notes that documentation is a focus for the next release, and the product team plans to bring forward any relevant content from earlier releases, which will be identified as still applicable to the current release.
- **For point 3** Microsoft notes that the ability to add the remote tools to a device image using Platform Builder is by design, as generally a finished device's final image would not normally include debug support. Additionally, because some devices won't have a .CAB installer, making installation of the remote tools a challenge. They are investigating now how to provide this support in a future release of Platform Builder. Microsoft also notes that the Compact Product Team was unable to reproduce the delayed connection time experienced by Dedicated Systems but will continue to investigate whether connection time is a persistent issue.

4 Installation and BSPs

Installation and BSP	0	8	10

Installation of the product tool chain went easy. The **Platform Builder** *is easy to use for configuring a platform as long you stick to the wizards. Customizing your platform remains a daunting task.*

Compact 7 supports a large number of boards and drivers. Most drivers are delivered in source code.

4.1 Installation

4.1.1 Installation on Host

The first step for using **Compact 7** is by installing the **Platform Builder** software, which is now integrated in the **Visual Studio 2008** environment. **Platform Builder** is the set of tools that is used to create a custom **Windows Embedded Compact 7** platform. This software can be obtained by downloading it and it supports ARM, MIPS, SH4 or Intel x86 based platforms. For our evaluations, the Intel x86 and ARM components were installed. It is always a good idea to install the ARM components as an ARM emulator is delivered with **Visual Studio**. This permits to start developing and debugging your application before the real platform is available.

Installing **Platform Builder** is similar to installing any other Microsoft software application, and is pretty straightforward and user-friendly. While installing the support for all supported CPUs, the total installation takes around 100GB disk space, which is a serious increase compared with the **CE6R2** evaluation! The reason for this is that there are binaries included for each BSP (and there are multiple BSPs for each CPU architecture!). Further there are three types of binaries: release builds, checked builds and debug builds for each of them. To avoid consuming a large amount of disk space, you can better only install the BSPs you really need, which of course also reduce the required installation time.

☺ The installation of **Windows Embedded CE Platform Builder** went smoothly. You have to choose the platform you want to build an image for, and after this choice, the installation starts.

On our host machine (**Windows XP**, Pentium 4 running at 2.4 GHz and 2 GB ram) it took around 4 hours to install everything on a fresh machine, when installing all BSPs (thus the 100GB install).

The next step is to use the **Platform Builder** to create, customize and configure a platform image. Configuring the platform image to your requirements can be complicated in review to the expectations. But this is the case for all embedded operating systems; indeed they all have to support many configurations.

☺ The **Platform Builder** integrated development environment (IDE) includes wizards for creating platform images and components. When using the default configurations as proposed by the wizard, it is then possible to create a system image for your device in a short time. Ending up with a running platform could take only one whole day, taking into consideration also the time for the installation procedure of the development tools.

But the moment you start to make changes to the platform image configuration or enable remote debugging and other remote tools, then things become quickly complicated and error prone.

4.1.2 Installation on target

Our target system that we mainly use for evaluating any real time OS, which will be the same for evaluating the **Compact 7** OS, is the Pentium MMX 200 MHz machine. We know that it is very old platform, but we still use it because it is the reference for comparing the RTOS performance evolution over more than 10 years. Moreover, such machine (board) has non-intrusive bios which may run below the kernel and thus falsify our measurements.

The experiences we had while trying our standard evaluation platform was bad. We tried different target options in order to create a platform image that can run on our target machine, but unfortunately none succeeded to boot on our target platform! We moved then for searching in the manuals and online help in order to find a solution. We discovered that this MMX platform seems to be unsupported (at least it is not in the supported list) to run an image of this OS version.

So our goal was to find a substitute platform on which this OS version is supported. We moved for using Pentium II 233 MHz platform. We succeeded to boot a **Compact 7** platform image on this platform (closest to our reference platform) which in turn will be used for our standard evaluation for this OS.

We tried as well an atom based PC, but also this one did not boot. It could be that the Atom platform was not fully compliant with the PC architecture.

Also, we did not succeed in starting the ARM emulation platform which was something that worked very well before. It seems that it is removed, although some things in the GUI still pop-up.

Instead of using the ARM emulation platform, **Virtual PC** can be used now as your target device. Microsoft **Virtual PC 2007** is needed to run the OS image. For the **Virtual PC 2007**, a disk image is delivered to let it boot and wait for an image to be delivered via the network (from the **Platform Builder**).

Booting a stand-alone PC requires also such disk image which is delivered as a floppy image! Hope you still have one of these old goodies… it is incredible that a floppy is still required in 2011! Floppy drives can still be found, but with the presence of USB flash disk, it is difficult to find a floppy. Why such disk image cannot be available on an USB stick instead of a floppy? It is of course possible to create such an USB boot disk, but there are no tools for creating such bootable USB delivered with the **Compact 7** development target…

After contacting Microsoft, it seemed that the tools to create bootable Compact 7 USB sticks exist. So Microsoft said that: “While the tool hasn’t changed there is no dedicated page for CE7 indicating the tool and bios loader support in general, unfortunately the documentation for the tool is only provided for previous versions. This is definitely something we will address."

Luckily the bootable image has been improved since **CE6R2**. You can set now more options (debug ports, screen size and so one).

Once the target connection between the host that is containing the **Platform Builder** and the target device is configured, it is then easy to boot the target. **Platform Builder** has a boot loader which works with Ethernet. There is one annoying feature in the bootable floppy which is that when you boot your target using this bootable floppy, a connection should be established between the target and the host in order to

download the image on the target. But if you do not make a connection within 60s, then the connection is lost and you need to reboot the target again to create a new connection request.

Luckily the source code is provided and thus you can rebuild the image with other timeout settings (maybe they should consider the timeout to be a settable boot option). Because the protocol uses a UDP broadcast to announce its presence, a timeout is normally appropriate to prevent from it from clogging corporate networking with extraneous boot messages. If you are not in a large scale test environments, you may want to consider changing the default timeout to your desired length

4.2 Board support package (BSP)

☺ It is possible to configure BSPs for a specific hardware platform by following the set of configuration wizards. You can create a new BSP by cloning an existing one, modify it and even create global drivers.

Compact 7 is a componentized operating system (OS) where features and drivers are selectable from a graphical IDE catalogue. After the user configures the OS feature set and collection of device drivers from the catalogue, a build dependency checker ensures that the image's feature set is self-consistent with all dependencies being met. We didn't detect problems with the dependency checker which means that consistent builds were made.

We have a complaint here which is the existence of "non-easy" developing options. For instance, enabling the remote tools requires also enabling a lot of components. So if you forget one, it will not work… why Microsoft does not provide a utility or a component named "enable remote (debug) tools access", which allows the programmer to select all the required dependencies automatically. The manual gives only the SYSGEN variables you need to set, but then you have to search these to find which components in the GUI set these variables.

Making then a remote connection using the remote tools (now in a separate GUI, but not integrated anymore in **Visual Studio** / **Platform Builder**) takes a considerable amount of time (around a minute). At the beginning we thought that the remote connection is not functioning because a network ping normally takes only a couple of ms, so we did not expect to wait for a minute for establishing a network connection! So we had just to wait…

Microsoft replied us that: "Microsoft notes that the ability to add the remote tools to a device image using Platform Builder is by design, as generally a finished device's final image would not normally include debug support. Additionally, because some devices won't have a .CAB installer, making installation of the remote tools a challenge. We are investigating now how to provide this support in a future release of Platform Builder. Microsoft also notes that the Compact Product Team was unable to reproduce the delayed connection time experienced by Dedicated Systems, which may indicate that it was something unique to the development hardware at Dedicated Systems."

Although the **Platform Builder** is stable, we had to restart **Visual Studio** several times before the building process succeeded. Also, the displayed error messages due to an error occurred during the building process were mostly unclear and abstract. For instance, we noticed that if we (re)build a platform image while we are still connected to the target, then the displayed error message was that there is an "if statement" error in a "make-file".

Our conclusion here is that we lost time and faced difficulties to get all things done and working more than what we had in the previous release. We are not saying that it is a bad tool, but if compared with some competitors, who made large steps forwards in user friendliness, then it is considered as a backwards step.

4.2.1 Creating custom BSPs / drivers

Microsoft install their BSPs into the IDE catalogue and thus make them available to customers with similar (or the same) hardware. Other vendors can also provide BSPs for their platforms.

☹ Adding your own drivers is however not possible using the graphical user interface. You need to adapt several configuration files before your driver is compiled and linked-in the system binary. Without the documentation and an example of driver source code, it would be impossible to do this.

In the normal manuals (help delivered with the tools), there is some lacking information there. Luckily, you will find some good whitepapers about this on the Microsoft website. It is a pity that these are not included in the manuals, but you can find them on this URL: http://www.microsoft.com/windowsembedded/en-us/develop/windows-embedded-compact-7-white-papers.aspx

In this whitepaper, they use the stream interface driver template as a start point, as this is the most common driver interface expected to be used by embedded driver developers. Stream interface drivers have the typical driver interface functions: open/close, read/write and I/O control functions as you will find on most other operating systems. The supplied manual could be improved by starting the device driver section with this driver type and with all the information available in the whitepaper.

Remark as well that the previous release required the programmer to implement (write) the stream interface driver by using a prefix of exactly three letters, which is an “undocumented feature”.

One problem that has not been addressed since previous releases is the fact that the graphical user interface is built on-top of a command line based system; such systems are less comprehensive than a pure command line based tool as the internals are hidden by GUIs. But with the GUI alone you will not be able to create your device driver (also there is not a wizard available for helping driver/BSP developers to start their developments). Following the white paper, you have to do a copy/paste of a directory to your BSP. This is thus completely command line based.

☹ This is one of the main problems that remained in the **Platform Builder**. Although it is integrated with **Visual Studio**, it behaves differently than normal **Visual Studio** projects! Once you are working in the **Platform Builder**, compiling, linking and other related tasks are handled completely different compared to the building system of traditional applications using **Visual Studio**. So there is a big difference between building an application and building an entire OS.

For example, you cannot lookup symbols within **Platform Builder** (also you do not have context aware auto member filling for instance)! This prohibits writing code easily. Of course we are focusing here on the BSP developer, device driver writer and component selection for a target platform. Once you are building applications on the platform, **Visual Studio** stays much closer to the native desktop development environment, but this is not our main focus in this document.

In addition to the device drivers, BSP developers will need to create an OEM Adaptation Layer (OAL) to abstract the kernel from the hardware implementation. The OAL is responsible for CPU and system board

initialization and is used routinely during system usage. The OAL handles things like cache operations, system timers, device interrupt handling, etc.

Although the concept is good, it is not an easy task. You have to understand very well the hardware-software border and how the kernel interacts with it. Remark that this statement is in general true for all embedded systems.

If we compare the situation with our previous evaluation, then we are having nothing more than a status quo. Even less as for instance the device emulator, which worked well, is vanished (or replaced with a virtual PC). As some competitors have improved their configuration system significantly the last couple of years, we had the feeling during our evaluation period that the tools took a step backwards.

4.3 Software development kit (SDK)

As said before, the **Visual Studio – Platform Builder** integration cannot be compared. Both use a similar GUI front-end, but they use completely different build systems. So how can you make your applications then?

Here, the SDK system comes in. It is the link between the **Visual Studio** application development environment and the **Platform Builder**.

Once you have created your platform BSP with propriety drivers and combined with applications that will be present on the target, you can easily create an SDK.

The SDK will setup an installation file so that the SDK can be installed on any other **Visual Studio** which has the Smart Device Programmability installed.

Once this is done, you can create a **Visual Studio** project using the **Visual Studio Smart Device templates**.

You can also add a Smart Device project to your solution containing the platform builder project and select the platform builder as target. In such case the application can be deployed to the platform builder target and as such integrated in the platform build.

☺ It has to be said that this system works very well to develop/debug your code for the target. In such case, you have all the advantages as building native **Windows** applications (symbol lookup, editor context awareness …).

It is however the configuring person who set-up an SDK/target which allows easy monitoring, debugging, and updating that target.

If this configuring person has done his job right, then that particular SDK becomes easily usable with the target. This works only with a real target device satisfying the condition that you already added all needed tools in your BSP (and thus SDK) which in turn makes it possible to download your application to the target and debug it. This can cost you a significant amount of time… Of course, once you have a good BSP installed on a device (so remote debug-able) and the linked SDK delivered for application developers, then developing becomes easy.

5 Technical evaluation

This is the qualitative approach of the evaluation. As such, the scores are given in respect of our gathered experiences from evaluating different products, and by definition they aren't based on objective criteria.

Remark: although the technical evaluation is done in this report, the quantitative test results of this operating system on a certain platform can be found in other reports.

☹ The **Windows Embedded Compact 7** has a small amount of available documentation about its internal architecture. As there is not a big change between this release and the **CE6R2** release, we based our discussion in a high extent on the documentation from **CE6** if we couldn't find anything specific in the **Compact 7** manuals. Microsoft acknowledged that that they did not put unchanged information in the Compact 7 documentation. They plan to forward into Compact any relevant content from earlier releases

5.1 OS Architecture

RTOS Architecture	0	8	10

Windows Compact 7 has a modular architecture. The virtual memory and privilege level protection makes the system robust.

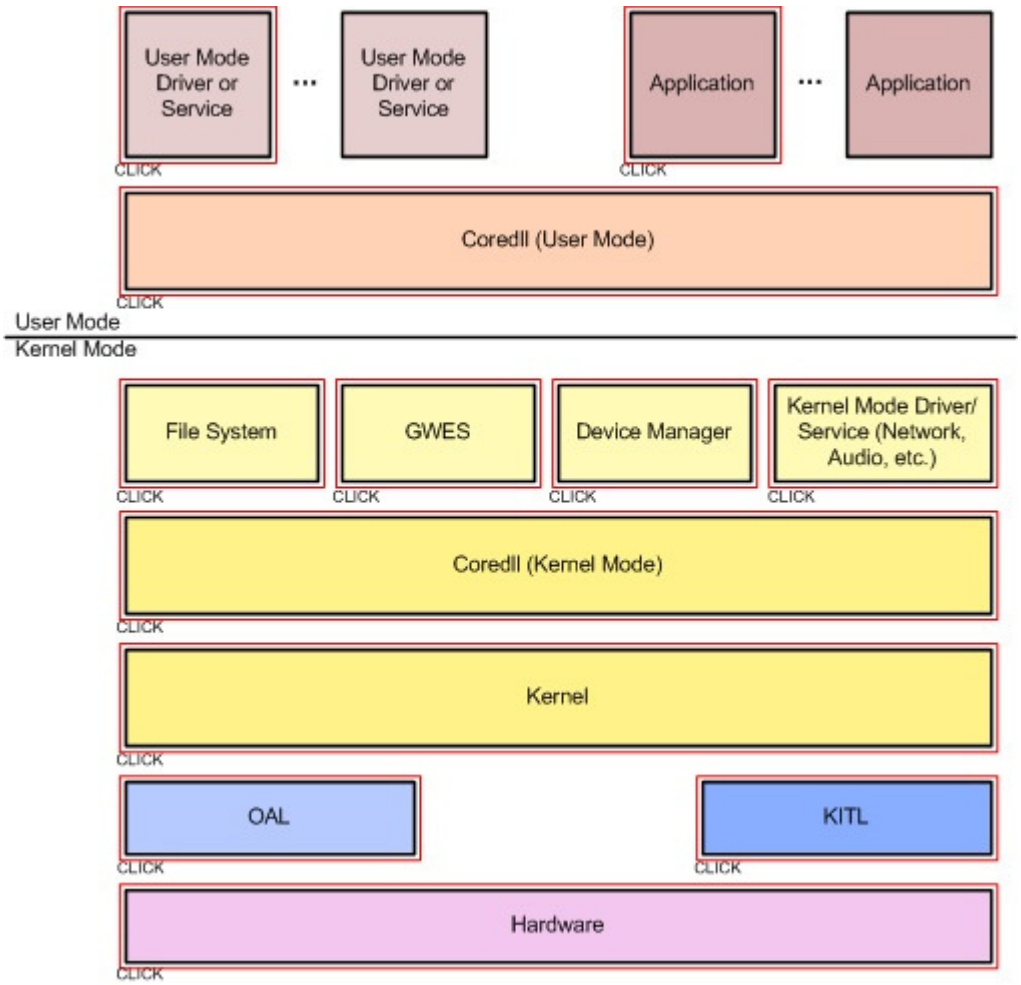
Windows Compact 7 is a priority based pre-emptive real-time operating system with protections between processes and a protected kernel.

For the people that still have doubts about it, we can confirm that **Compact 7** has all the features to make it a valid real-time operating system (this statement is true since **Windows CE version 3.0**).

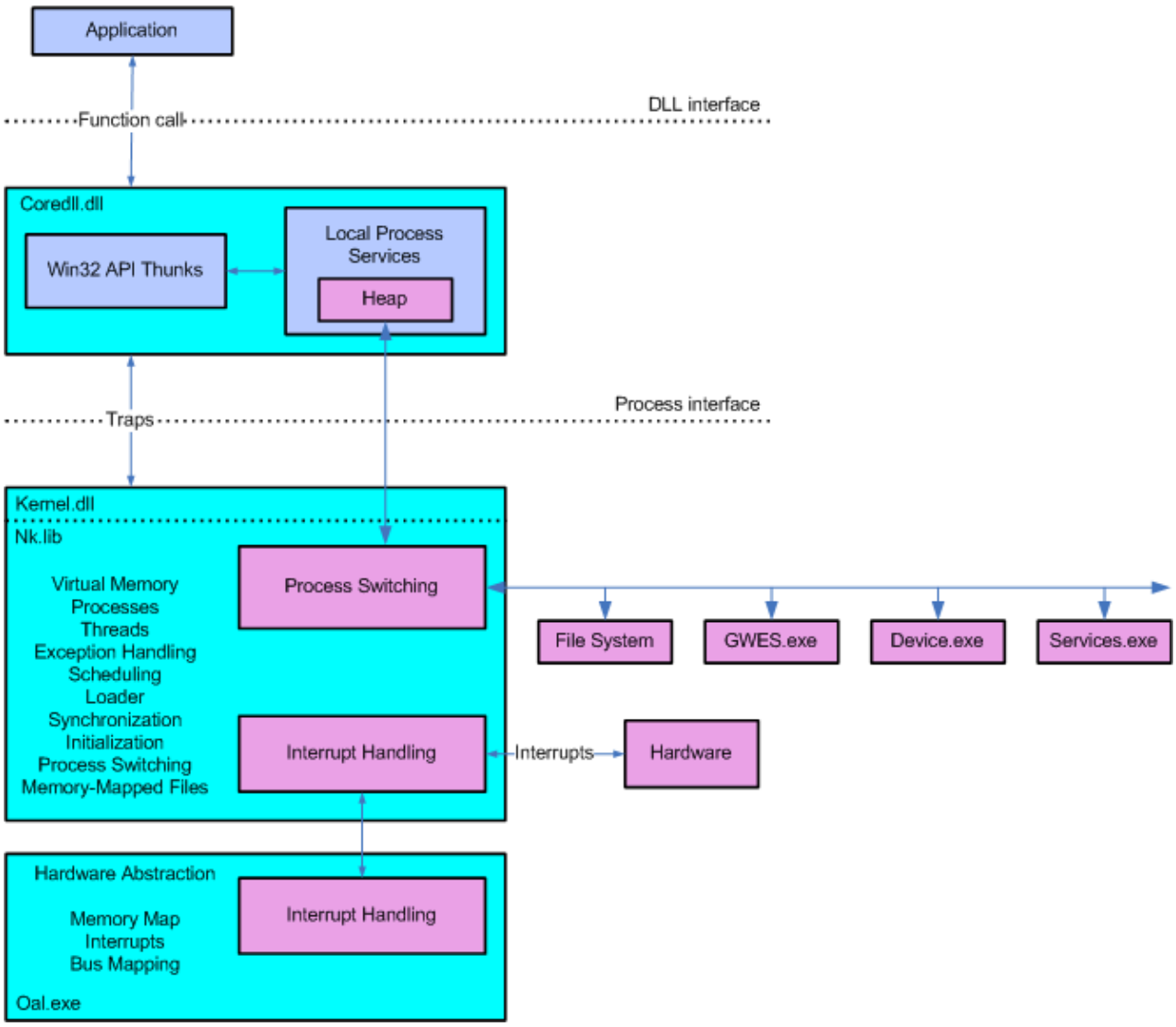
As mentioned before, the differences between **Compact 7** and **Windows CE 6R2** are minor. There is only one fundamental change which is the support for multiple processors.

Of course, the main advantage of **Windows Compact** stays the interoperability with the general purpose Windows environment (with technologies like **.NET**) and the ease to make windowing applications just like in the general purpose Windows operating system. Therefore, you can use a large developer base for the non-real-time part of your system. Of course, the real-time part is better designed by developers with a good understanding of real-time behavior.

Although it has a modular design in order to suit small systems with a minimal configuration, it is very complex and has a very large footprint to be a good solution for deeply embedded systems.



CE Architecture



CE API system

5.1.1 Task Handling Method

5.1.1.1 Processes and threads

Windows Compact is a classical pre-emptive multitasking system with support for both processes and threads within a process. **CE** uses a priority based scheduler to decide which ready-to-run thread it will activate. Remark that priority number 0 is the highest priority. Threads with the same priority are scheduled using a fair ***round-robin*** time slice mechanism. This time-slice quantum can be set for each thread separately.

The number of running processes in **Windows CE** is unlimited (there is a theoretical limit of 32K, but in practice this means unlimited). Each process can access a full 2GB private memory region.

The number of running threads within a process is only limited by the available system resources (RAM)

.

5.1.1.2 Process protection ⇔ full kernel mode

Since **CE 6.0**, each process has its own virtual address space.

5.1.1.3 Thread priorities

Compact 7 uses 255 priority levels which are large enough to accommodate complex real-time systems.

Compared with other RTOS, **Windows Compact 7** is built to be compatible as much as possible with its cousins, the general purpose operating system: **Windows 7**. Nevertheless, different changes (listed below) were done in the kernel to make **CE** suitable for real-time systems:

- To be compatible with GPOS Windows, the lower 8 priorities are the same as used in the normal Windows `SetThreadPriority` calls. Thus ported Windows applications to CE will not affect the real-time behavior of the system.
- The 248 highest priority levels can be set only by the specific `CeSetThreadPriority` call.

An OEM can protect the top most 248 priority levels, so that only the lowest 8 levels are available for applications. This makes it possible to add third-party software without compromising real-time performance.

Microsoft recommends using the priority levels ranges as follows:

- 0 through 96: Reserved for real-time above drivers
- 97 through 152: Used by the default **Windows CE**–based device drivers
- 153 through 247: Reserved for real-time below drivers
- 248 through 255: Mapped to non-real-time priorities

The system designer can adapt this according to his needs. The developer is free to use any priority that fits his application, at least if he has the rights for it.

In previous versions, Microsoft did a good job by documenting the default priority of all system and driver threads. However in **Compact 7**, less documentation about this was found.

It is a pity that it isn't possible to create a thread with a predefined priority level: each thread is created with the default priority level (`THREAD_PRIORITY_NORMAL`: 251). Therefore you will need to change the priority of the thread after its creation. You can however create a thread in suspended mode, then set its priority, and finally start it.

Remark that an interrupt run as a thread, and as such they use the same priority levels (see the interrupts section further in this document).

5.1.1.4 Fibers

Compact 7 supports also "***Fibers***". ***Fibers*** are also thread of executions but NOT managed by the kernel. Each ***fiber*** runs in the thread context of the thread that created it.

So ***fibers*** are useful to create your own scheduler within a thread.

This can probably be used for some exotic applications or for legacy systems, but normally it is not a good programming concept in real-time systems.

	OS under evaluation
Supported memory models	☺ - Processes protected between each other, with multiple threads in one process. - Flat memory model: not supported anymore.
Thread/process priority levels	☺ 256 levels
Max. number of processes	32K processes
Max. number of threads	The maximum number of *threads* in a process is only limited by the amount of memory available.
Scheduling policies	***Priority based scheduling*** ***Round-robin*** between threads at the same priority level, with adjustable time-slice (quantum). When the quantum is set to zero, the thread runs to completion.
Number of documented thread states	5 states (running, suspended, sleeping, blocked, and terminated).
Number of undocumented thread states	

5.1.2 Memory Architecture

In **Windows Embedded Compact 7**, each process has a 2GB address space, which is the same as for most other 32-bit operating systems with virtual memory support.

The concept: “when the virtual memory is actually physically mapped and thus available” is not clearly documented in the manuals. So if you want to be sure, you should use `VirtualAlloc` with the flag to commit it directly.

Also, the concept: “how a program is loaded and whether it is fully loaded to the RAM when execution starts” is not available in the documentation.

If you want to avoid any page faults for your application, then special precautions have to be taken. Your executable should be seen by the system as a “driver” module. This can be done by following mechanisms:

- You create your real-time part of the application as a DLL and load it using the “LoadDriver”. The LoadDriver function will call LoadLibrary, but after loading the library, it will ensure that everything is loaded into RAM (code and data sections) and that it stays there (cannot be swapped out).
- You set in the platform.bib file, that the executable is a “Module”. Again, this signals the operating system that it should be handled as a driver and thus all code/data sections have to be loaded in RAM before starting the application. It also ensures that it will never be swapped out.

Of course, you will still have impact of caches and TLB exceptions, but this is the case for all operating systems using virtual paged memory.

5.1.2.1 Boot ROM and XIP

It is still possible to configure **Compact 7** in a way that it can be executed (partially) from ROM, which is called ***eXecute*** In Place (XIP).

There is no limitation as booting can also be done by flash card, compressed ROM and so on.

5.1.2.2 Physical memory

Another major difference between **Windows Embedded Compact 7** and **Windows CE 6.**0 is the amount of physical memory that can be managed by the kernel. In **Windows CE 6.0,** this was limited to 512MB. Now it is possible to use up to 4GB physical addressable memory space (devices included). Concerning RAM, it is limited to 3GB.

The RAM in **Compact 7** based device is divided into two areas: the object store area and the program memory area. The object store in RAM is optional, so the whole physical RAM can be allocated to the program memory.

- The object store looks like a permanent virtual RAM disk. Data in the object store is retained when you suspend or perform a soft reset operation on the system (if the device has a backup power supply for the RAM). When operation resumes, the system looks for a previously created object store in RAM and uses it, if one is found. Devices that do not have battery-backed RAM can use the hive-based registry to preserve data during multiple boot processes.

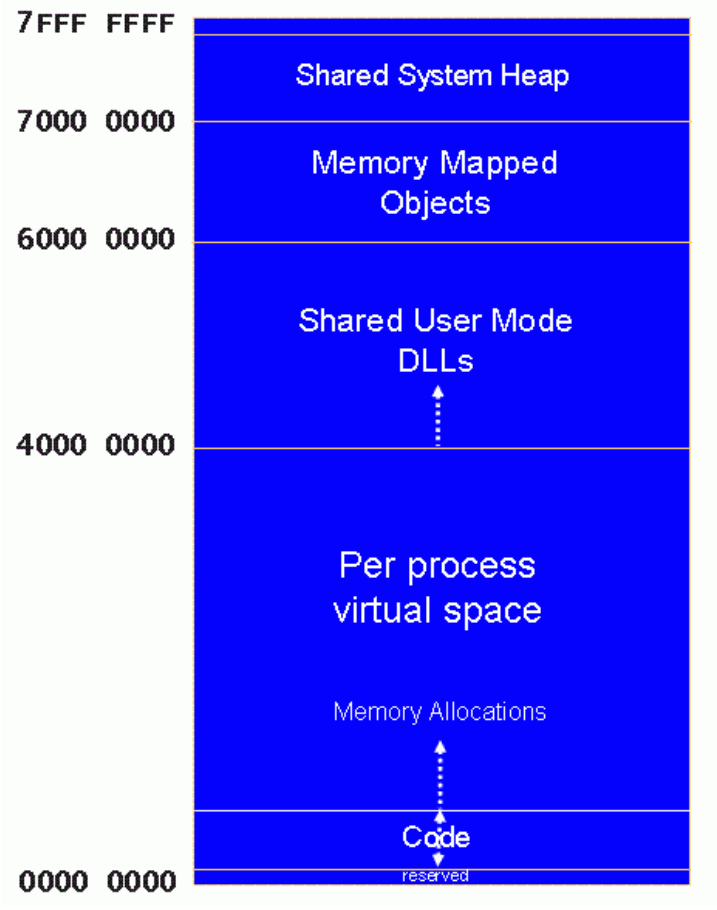
- The program memory consists of the remaining RAM. Program memory works like the RAM in personal computers — it stores the heaps and stacks for the applications that are running.

Dynamic moving memory allocation from object store RAM to program RAM is possible (via a control panel application). Some applications might indeed require this feature.

5.1.2.3 Virtual memory

In **CE6R3,** each process had its own private region of 2 gigabyte virtual address space. We suppose that this is unchanged in **Compact 7**. Probably, the virtual memory space of the kernel has changed (as the amount of supported RAM increased); however we cannot find anymore documentation on this.

Remark that in previous versions such things were documented…



User virtual address space (within a process)

5.1.2.4 Paging

Windows CE implements a paged virtual memory management system similar to other Microsoft Windows–based desktop platforms. The page size is on most platforms 4,096 bytes (4 KB).

Windows Embedded Compact 7 does not have support for large pages.

Important thing in real-time systems is that it is possible to disable demand paging (done by changing the ROMFLAGS register settings). This can be done for both, the module and the whole system. In this mode, a module is fully loaded into the memory before it runs. So paging faults cannot occur during the run of the application. This is important to provide predictable performance. If you did not disable demand paging, you have to explicitly indicate that your application is a module, or you have to use the LoadDriver function and have the real-time code in a DLL, so that the operating system will handle this code differently and keep it in RAM.

For security reasons, the Translation Look aside Buffer (TLB) is flushed on an x86 when going back from the kernel to the application space. For real-time purposes, this can be disabled as well, but it introduces a security hole.

5.1.2.5 Memory protection

Under normal configurations, the MMU is used to protect processes from each other and to protect the kernel space. This is the same as currently used in most General Purposes operating systems.

This protection helps a lot for making reliable and robust systems.

5.1.2.6 Heap

The heap in **Windows CE** is optimized for small objects. However, just like in any real-time system, long-time usage can lead to a fragmented heap causing extra lookup delays and even failing allocation requests.

As this was the case in previous versions which led sometimes to problems with long time running systems (after months without reboot), Microsoft addressed this problem in **Compact 7** and reworked their heap manager so that fragmentation should be reduced compared with the previous versions.

You can add your own heaps in a process. This can then be used for instance for fixed size allocations (preventing any fragmentation) in applications with real-time constrains. If you setup your own heap, you can as well replace the `alloc` and `free` routines, so that you have full control.

	OS under evaluation
MMU support	Yes
Physical Page Size	4KB (depending on CPU)
Swapping/Demand Paging	Supported, but can be disabled to achieve real-time performance.
Virtual memory	YES
Memory protection models	☺ Full virtual memory protection. Each process runs in its own virtual private memory space.

5.1.3 Interrupt Handling

Interrupt handling is not changed since the previous version of **Windows CE**.

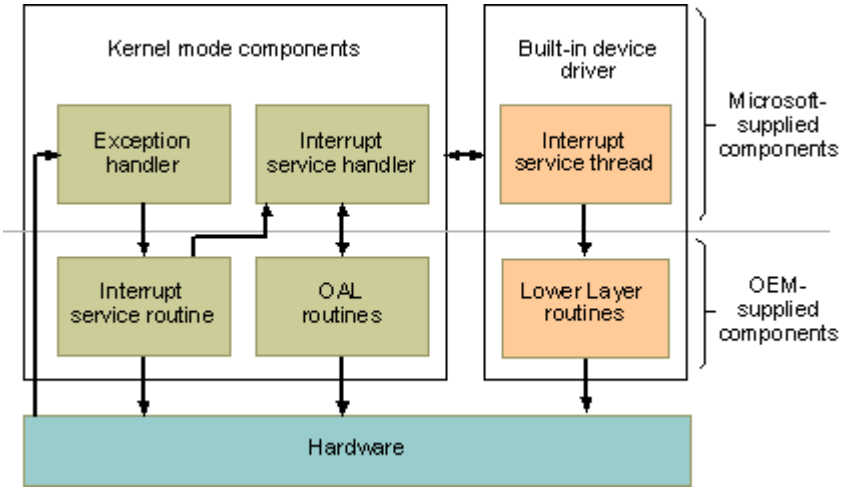
Interrupt handling in **Windows CE** is somewhat different if compared to more traditional RTOS. Microsoft uses the same terminology as we do for the interrupt handling:

- The ***Interrupt Servicing Routine (ISR):*** low level routine in the kernel with limited system calls.
- The ***Interrupt Servicing Thread (IST):*** thread at application level handling the interrupt with access to all system calls.

The difference is that in **Windows CE**, the ***ISR*** is meant to be implemented in the OAL (OEM Abstraction Layer) just for handling the lowest level of the interrupt (interrupt controller). It is not easy to add quickly an ***ISR***… You can only do it in the OAL layer, or by creating a ***driver DLL*** just for the handling the ***ISR***! It is not possible to have the ***ISR*** integrated with the rest of your driver!

Therefore device drivers will in general run their interrupt handler in the ***IST***. Interrupt service threads act just like any other thread: so they use the same priority system like normal application threads! This means that the system designer can end up in giving an ***IST*** a lower priority than some application thread, which is not a good design for the real-time part of your application, but which might be interesting for the non-real-time part (thus for interrupts that do not have any real-time requirement).

This system is shown in the figure below (which still comes from the CE 6.0 manuals, as this information is lacking in the **Compact 7** documentation):



Interrupt handling in Windows CE 6.0

This approach has its pros and cons. The main negative point is the extra delay before the ***IST*** is activated. One advantage is that it makes interrupt handlers simpler (all system calls available) and gives the system designer a high degree of freedom. Another advantage is that the hardware interrupt levels do not affect the ***IST*** priorities. We consider the freedom to choose some interrupt service threads with a lower priority than some more important threads as a plus. Not all device interrupts will have real-time requirements, and as such you can prevent to be delayed by these by using higher priorities.

If however for some real-time constraints shorter latencies are required, it is possible to make a custom ***ISR*** by adapting the OAL (in fact by adapting the BSP) or by creating a specific ***ISR driver DLL***. As said before: this is not an easy task. We do not see immediately cases where the small extra overhead of using an ***IST*** would make you miss deadlines. Also, this is not possible on all types of hardware's, as for instance the ARM has only one interrupt.

Windows Compact 7 kernel exception handler is a prioritized pre-emptive one if possible, so higher level interrupts can interrupt lower level interrupts in their ***ISR***. Remark that this depends on CPU interrupt handling and ***Priority Interrupt Controller (PIC***), thus not all platforms support this.

	OS under evaluation
Handling	☺ Nested and prioritized **Compact 7** uses a thread interrupt model to encourage the use of threads to handle most of the interrupt service work. OEMs can access a kernel ***ISR*** also to perform a minimal amount of work. **Windows CE** also provides installable ***ISRs***.
Context	The ISR runs in a special context and uses virtual addresses statically mapped by the OEM. The ***IST*** is a normal application thread and has its own context and priority like any other thread in the system.
Stack	The ***IST*** is a normal application thread and has its own stack.
Interrupt-to-task communication	An event is used from within the ***ISR*** to signal the ***IST***. No other ***API*** is accessible from within the ***ISR***. OEM can create a shared memory region by statically mapping a memory region into the ISR's address space.

5.1.3.1 System timer

The OAL implements the system timer interrupt (in fact that is the only required ***ISR*** in the kernel) that should be activated each millisecond. This interrupt has to check if any time-out is reached and if this is the case, then it signals a "reschedule" event to the kernel. In the other case, a "NOP" is signaled to the kernel and no rescheduling occurs.

Since **Windows CE 6.0** the lower level OEM adaption layer (OAL) receives the number of milliseconds it may sleep before a timer event is needed. This is done before going to the idle mode and allows slowing down the timer interrupt if no threads are running. Such feature is very useful for saving power (important for handheld devices). This mechanism can be compared with the "tickless **Linux** kernel", although the reasoning why it is introduced is different: in **Windows CE** it was introduced for enhanced power savings, while **Linux** uses it to limit the CPU load in virtual machine clusters (as otherwise each virtual machine instance wakens up to handle the tick).

Remark however that we noticed that some BSP (in this case from another vendor, thus not from Microsoft) use this mechanism in a bad way. Their implementation made the timer interrupt much longer and as such had a serious impact on real-time behavior! Be aware of this when selecting a certain BSP. In general you have to be very careful to use power saving mechanisms in real-time systems…

The clock timer never runs at the thread level and is very compact (at least on BSPs provided by Microsoft). As long as no time-outs are detected, the delay will be small. A short clock timer interrupt improves latencies.

5.1.3.2 High resolution timers

An OEM can add high resolution timers for more accurate timing or performance analysis. This is not mandatory as this will depend on the used hardware platform.

These timers are only to be read from an application or from drivers and serve the purpose of a measurement tool only: they are never used by the kernel in scheduling threads and calculating time-outs.

5.1.4 Synchronisation mechanisms

In **Windows Embedded Compact 7**, the synchronization mechanisms can be divided into two distinctive categories:

- Protection mechanisms against simultaneous access: ***critical sections, mutexes and semaphores***.
- Communication mechanisms: events and message queues.

These two categories are fundamentally different in their implementation so that you should not use protection mechanisms as a communication mean. This is not a negative point! On the contrary, as will be explained in the sections below.

The main difference with other RTOS is the use of generic wait functions that are (like `WaitForSingleObject`) independent of the object type (***mutex, semaphore***, events but also process and threads). This is similar to what is done in a General Purpose Windows Operating System. As an advantage, it is possible to wait on multiple objects (until one of them is signaled). For the experienced real-time systems developer (with other RTOS), this system might be perceived as somewhat strange.

5.1.4.1 Protection mechanisms

Two main mechanisms are foreseen in **Windows Compact** to do this:

- ***Critical sections***: Can be used only within a process. We suggest using this mechanism if you are using it for protection within a process only. As it cannot be used between processes, it has some optimizations by using atomic instructions of the CPU. So it avoids round-trips to the kernel if there is no contention on the lock. Therefore this is the method to use if you want optimal multi-threading performance.
- ***Mutex***: like the counting **semaphore** but without counting

☺ Fundamentally, it is here that **Windows CE** uses priority inheritance to avoid priority inversion cases for all owned locks (thus ***mutex*** and ***critical sections***)! Even better: it is always there, it is impossible to disable priority inheritance! This clearly shows that Microsoft is very aware on how to achieve real-time behavior in a system and prevents the developer from wrong designs!

This is a feature that we didn't find in any other RTOS we evaluated so far. In our opinion, having priority inheritance always enabled is what should be done in all RTOS! Remember that the system crash on the first Mars Pathfinder was caused by a ***mutex*** used in its default settings without priority inheritance enabled. The wrong behavior (=bug) was fixed by just enabling the priority inheritance for this ***mutex***. This example shows you, how important this feature is!

Of course, some people will argue that this causes the protection mechanisms to be slower. This is true in the average case, but in real-time systems the average is of no importance: the worst-case is the only time we are interested in. In such scenarios, priority inheritance does exactly what it is required to do: it improves worst-case situations!

In our opinion, it doesn't make sense to disable the priority inversion whatever was the case, just like it is useless to disable the brakes of a car…

One remark here: priority inheritance works only at one level. When using a second protection mechanism while the priority is already increased, this will not cause a second inheritance. This is not a main problem: using a second protection while already in a protected area is just an example of bad design and should never happen in any well designed real-time system.

The counting ***semaphore*** can also be used as a protection mechanism, but this is not its real purpose. Indeed, a ***semaphore*** can be used to wake-up one thread by another thread (= signaling function) which cannot be done by a ***mutex***. It is said that a ***mutex*** lock is "owned" by a thread, which is not the case for a ***semaphore***.

Remark that you can use a named **mutex** to allow inter-process protection.

5.1.4.2 Communication mechanisms

The following communication mechanisms are foreseen in **Windows CE**:

- ***Semaphores***: are used to synchronize access to some resources, or between processes. For the attentive reader: remark that this not the same as a ***mutex***! Although on most educational institutes, it is educated that a ***mutex*** is just a binary ***semaphore***, this is not true. The main difference is the "ownership" of such an object. A ***mutex*** can only be released by the thread that has taken the ***mutex*** (he received ownership during the "`P()`" operation), while with a ***semaphore*** any thread can release it!
  This difference is crucial, as ownership is required to implement priority inheritance algorithms!
- Events: are used to signal that something occurred and can be used between processes.
- Message queues: are to pass data between threads (not usable between processes).

Not much to explain here: these mechanisms behave the same as in most other RTOS.

If there is a need to pass data between processes, then shared memory can be used to do so.

5.1.4.3 Interlocked mechanisms

Windows CE has also some `InterLockedXxx` API calls available to perform atomic operations like incrementing and decrementing a counter.

On most processors, these can be implemented by special atomic CPU instructions.

5.2 API richness

Windows Compact 7 *uses a subset of the Win32 API. This API provides the most commonly used kernel features. Further it has some extensions compared with the Win32 API, typically needed for embedded real-time systems*

It should be noted by the reader that the tables below have the sole purpose of making an inventory of the kernel related APIs. The full API includes a very large amount of interfaces that provide features that are beyond the scope of this study.

The tables below make an inventory of basic real-time objects and features present in the `POSIX` and OS proprietary interfaces.

While interpreting these results, the reader should keep in mind that these tables cover a strictly defined set of system calls only. As it is very hard to compare the different API for the different operating systems, no score is given for this section. The reader should use this section to check if the API calls he needs for his application are available or not.

In general, the more system calls are available, the easier it is for the programmer to find the correct call for his application, without writing an own library with for instance wrapper calls…

5.2.1 Task Management

Thread management	YES
Get stack size	-
Set stack size	✓
Get stack address	✓
Set stack address	-
Get thread state	-
Set thread state	-
Get TCB	✓[1]
Set TCB	✓
Get priority	✓[2]
Set priority	✓
Get thread ID	✓

[1] This structure is called a thread context in Compact 7
[2] This will return the current priority, which may be increased by the priority inheritance mechanisms in place.

Thread management	YES
Thread state change handler	-
Get current stack pointer	✓
Set thread CPU usage	-
Set scheduling mechanism	✓[3]
Lock thread in memory	-[4]
Disable scheduling	✓

5.2.2 Clock and Timer

Clock	YES
Get time of day	✓
Set time of day	✓
Get resolution	✓
Set resolution	✓[5]
Adjust time	✓
Read counter register	✓
Automatically adjust time	✓

Interval timer	YES[6]
Timer expires on an absolute date	-
Timer expires on a relative date	✓
Timer expires cyclical	-
Get remaining time	-
Get number of overruns	-
Connect user routine	-

[3] Only the round-robin quantum can be set. Scheduling is always priority based, with quantum based round-robin scheduling for threads at the same priority.
[4] All threads can be locked in memory at configuration time. Executable modules can be loaded and locked into memory with the "LoadDriver" API. It is also possible to disable demand paging for the whole system.
[5] This depends on the OAL. Default is 1000Hz.
[6] Beside "sleep" and "wait" on an object with timeout, there is no timer support.

5.2.3 Memory Management

Fixed block size partition	NO
Set partition size	-
Get partition size	-
Set memory block size	-
Get memory block size	-
Specify partition location	-
Get memory block – blocking	-
Get memory block - non blocking	-
Get memory block - with timeout	-
Release memory block	-
Extend partition	-
Get number of free memory blocks	-
Lock/unlock partition in memory	-

Non-fixed block size pool	YES[7]
Set pool size	✓
Get pool size	✓
Make new pool	✓
Get memory block size	✓
Get memory block – blocking	-
Get memory block - non blocking	✓
Get memory block - with timeout	-
Release memory block	✓
Extend pool	✓[8]
Extend block	✓
Get remaining free bytes	-

[7] By the HeapCreate and related calls. Remark that it is also possible to create a heap with your own allocation/free handlers.
[8] The heap cannot be extended with an API, but the system automatically extends it when necessary.

Non-fixed block size pool	YES[7]
Lock/unlock pool in memory	-
Lock/unlock block in memory	-

5.2.4 Interrupt Handling

Interrupt handling	YES[9]
Attach interrupt handler	✓
Detach interrupt handler	✓
Wait for interrupt - with timeout	✓
Raise interrupt	-
Disable/Enable hardware interrupts	✓
Mask/Unmask a hardware interrupt	-
Interrupt sharing	✓

5.2.5 Synchronization and Exclusion Objects

Counting semaphore	YES
Get maximum count	-
Set maximum count	✓
Set initial value	✓
Share between processes	✓
Wait - blocking	✓
Wait - non blocking	✓
Wait - with timeout	✓
Post	✓
Post – Broadcast	✓[10]
Get status (value)	-

[9] An Interrupt Service Thread is used. As such, you can wait on an Event with a WaitForSingleObject function, which allows time-outs.
[10] By incrementing with more than one.

Binary semaphore	NO[11]
Set initial value	-
Share between processes	-
Wait – blocking	-
Wait - non blocking	-
Wait - with timeout	-
Post	-
Get status	-

Mutex	YES
Set initial value	✓
Share between processes	✓
Priority inversion avoidance mechanism	✓
Recursive getting	✓
Thread deletion safety	-
Wait – blocking	✓
Wait - non blocking	✓
Wait - with timeout	✓
Release	✓
Get status	-
Get owner's thread ID	-
Get blocked thread ID	-

[11] Binary semaphores are not explicitly provided, but can easily be simulated by a counting semaphore with maximum count of one. Also it is possible to use the provided critical section API.

Conditional variable	NO
Pend non-blocking	-
Pend with timeout	-
Pend in FIFO / priority order	-
Broadcast	-
Priority inversion	-

Event flags	YES
Set one at a time	✓
Set multiple	-
Pend on one	✓
Pend on multiple	✓
Pend with OR conditions	✓
Pend with AND conditions	-
Pend with AND and OR conditions	-
Pend with timeout	✓

POSIX signals	NO
Install signal handler	-
Detach signal handler	-
Mask/unmask signals	-
Identify sender	-
Set destination ID	-
Set signal ID	-
Get signal ID	-
Signal thread	-
Queued signals	-

5.2.6 Communication and Message Passing Objects

Queue	YES
Set maximum size of message	✓
Get maximum size of message	✓
Set size of queue	✓
Get size of queue	✓
Get number of messages in queue	✓
Share between processes	✓
Receive – blocking	✓
Receive – non blocking	✓
Receive – with timeout	✓
Send - with ACK	-
Send - with priority	✓
Send – OOB (out of band)	-
Send - with timeout	✓
Send – broadcast	-
Timestamp	-
Notify	-

Mailbox	NO[12]
Set maximum message size	-
Get maximum message size	-
Share between processes	-
Send - with ACK	-
Send - with timeout	-
Send – broadcast	-
Receive – blocking	-
Receive – non blocking	-
Receive – with timeout	-
Get status	-

[12] A mailbox is in fact nothing more than a message queue that can store no more than one message. It is included for the sake of completeness, but most operating systems do not explicitly support it anymore.

5.3 Documentation

OS Documentation	0	6	10

The delivered documentation with **Compact 7** *(MSDN) is considered as a step backwards compared with* **Windows CE 6.0***. It contains mostly only the documented API. It lacks overview documentation or architectural discussions which existed in the CE 6.0 documentation.*

Luckily you can find more information on the Microsoft website, where there are some good white papers.

However you will not find any architectural documentation on the kernel internals anymore, something that is important for real-time system engineers.

Microsoft offers paid premier support for OEMs. Online support can be freely obtained from an extensive online knowledge base on the MSDN website.

Windows Embedded Compact 7 comes with MSDN documentation set, which is also available online.

☹ This documentation set contains a lot of reference information, but it lacks more fundamental documentation on internals or “howto’s”. As a result it is not appropriate as a tutorial.

Newcomers will have a hard time acquiring an overview of the system if this is the only documentation they have at their disposal.

Luckily you can find more on the Microsoft website, where there are some good white papers on some topics (for instance on how to make a device driver).

☹ In the **Windows Embedded Compact 7**, all architectural documentation is vanished… This is the main problem of the documentation. If you want to know something about the internal workings (behavior) of the system, you have to look in the **CE 6** documentation and cross fingers that nothing changed!

Microsoft have taken our complains seriously: “Microsoft notes that documentation is a focus for the next release, and the product team plans to bring forward any relevant content from earlier releases, which will be identified as still applicable to the current release.”

Microsoft provides an extensive knowledge base on its MSDN website, which can be freely consulted. Premier support for OEMs is available against payment.

5.4 OS Configuration

OS Configuration	0	7	10

The configuration system is not improved in this release although it is now integrated in the **Visual Studio 2008***. But the integration is more cosmetic than real. Both* **Platform Builder** *and* **Visual studio** *projects use a different mechanism to build the source files and as such behave totally different.*

As long as you stay within the standard set-ups, everything works very well now. Once you have to add your own drivers and BSPs, things become complex.

5.4.1 OS boot options

The boot loader follows a semi-chronological order of tasks, but there are some options you can set yourself. For example, it is possible to relocate the run-time image in RAM as its initial place is in flash. **Compact 7** can obtain an IP address from a DHCP server; the alternative is to assign static IP addresses. For debugging purposes you can add a passive KITL connection which won't disturb the OS.

All the tasks of the boot loader are supported by libraries which make it easier to develop your boot loader.

There are 2 types of binary images that will be used by the boot loader: ***.bin*** file and the ***.nb0*** file.

- The most common format for a **CE** image is the ***.bin*** file. It is optimized to minimize the amount of data that needs to be transferred to the device.
- The ***.nb0*** file is useful to place the boot loader image on the target device. The format is a RAW binary image of the boot loader. Most of the time, the board manufacturer provides a program to do this, but alternatively you can do the same thing through a JTAG connection.

5.4.2 OS configuration options

The next step is to use the **Platform Builder** to create, customize and configure a platform. Configuring the platform in accordance to your requirements is a complicated process.

Let's start with the good news: if you use a standard BSP to develop your software, then the wizard will help you to quickly set-up an environment from scratch. You can be up-and-running in one day from the start of installation and the platform also booting up which is very fast.

With the current version of **Platform Builder,** we had sometimes stability issues; it did not crash but sometimes a restart of the **Visual Studio** was required in order to successfully build the image.

Another issue is about the unclear terminologies used (build, SYSGEN, BSP) which makes it difficult to know what is going on. It happened too often that a rebuild of the whole system is performed while our aim was just to rebuild our own driver. It is not the first time we complain on this issue; we hope that Microsoft will improve such used terminologies one day, so that it becomes more intuitive.

Once you go further and start to change options in the registry, things become more complex. Parts of the registry are scattered around… it is not easy to detect in what part an adaptation should be done, and how the registry will look like at the end on the booted platform. Such things still need to be done manually!

We would like to have a "properties" box for each component where all the properties (registry) of this component can be edited in a user friendly way (a form with real options…). The fact that this is internally handled by registry settings is really not our concern. As you know, the registry is a huge collection of settings; so if you do not have any extensive knowledge of all kind of parameters, it becomes cumbersome to solve the puzzle.

The main problem is that the **Platform Builder** is a GUI built on top of a "makefiles" based build system. This makes it difficult to understand.

Our conclusion is that starting-up a platform directly from the wizard is an easy task, but doing anything beyond that stays difficult, which makes the configuration process a difficult task for newcomers.

5.4.3 BSPs

Windows Embedded Compact 7 provides a BSP for each platform category. It is possible to create your own BSP by cloning an already existing BSP and modify it. The option to create global drivers for all BSPs is an advantage.

It is possible to create new BSP's, add a boot loader and drivers to it. All this can be done through the wizards of the Microsoft **Platform Builder** IDE tools. However, in previous release, it was possible to create/export/import BSP while the current version supports only cloning of an existing BSP.

The difficulty of this process is that you need to configure your run-time image configuration files in order to use the new BSP (the registry as explained in the section before). By changing these files, you can easily make mistakes which will raise the possibility that the build process is disturbed and in turn fails to be successfully built due to your inadvertently changes. Errors will occur through the building process which in turn will display "unclear" error messages without any clue of the error. You have then to spend some time searching for the error. We consumed a lot of time in some situations to figure out what was the problem.

As it is really easy to screw things up, be sure to take backups of the BSP directories before changing anything.

Building the whole platform takes a while, so it is important to use the available processors. You can configure the number of threads used for a build from within the IDE: in the Tools | Options then Platform Builder\OS Design and Build in the "Multi-Processor Build" group you can use the default to automatically detect the number of processors or you can manually specify the number of threads you want to run. Our tests were inconclusive, as the CPU load seemed to be only covering one CPU for a significant part of the build, while for other parts it indeed used them all.

5.4.4 Device drivers

As **Compact 7** did not have any fundamental change in its kernel internals, developing a driver for **Windows Compact 7** can be performed in the same way as in **CE 6.0**. As a result, drivers should be compatible between these two versions.

Different device drivers' source codes are delivered which makes it easy to take one and adapt it to your needs. It provides also a good debugging help if something goes wrong (you have access to almost all the code from the kernel debugger). This can save a lot of your time.

Integrating a new driver in a custom BSP is still a daunting task! You have to create registry entries for it and try to integrate this in the BSP registry without screwing up the other BSPs… Also you have to create some custom SYSGEN variables that can be used then by the **Platform Builder.**

The way of doing such thing is error-prone: it is still not an easy solution to be used. The worst issue is the error messages that are not always clear if something goes wrong. There is a total lack of documentation in the delivered support documents. However there is a good white paper on this issue, it is strange why such useful information is not included within the MSDN documentation set.

See link: http://www.microsoft.com/windowsembedded/en-us/develop/Building-Testing-Compact-7-Device-Driver.aspx

Moreover, this document still does not have all the information you need. For instance, it does not talk about the prefix entry points of the stream driver, which can be found in another white paper. Also it is not mentioned that if you use prefixes, it should not have more than three characters. By using the DEVFLAGS_NAKEDENTRIES there is no need to use anymore prefixes for these exported calls. The sample drivers and BSPs don't generally use them anymore.

Microsoft notes that Prefix entry points for CE drivers is an old, legacy approach to implementing drivers it was required in the CE1.x and 2.x but hasn't been required for several major OS releases.

As an experiment, we made a special BSP for our tests (see complementary document), where our custom drivers are included for hardware time and event tracing (not intrusive measurements with dedicated hardware). This is an example of the list of required and unexpected things, which makes you frustrated; all these things for just creating a simple PCI device driver…

- Copy the StreamDriver example.
- Create SYSGEN variables for the new driver.
- Create registry settings for the driver so that the PNP mechanism can work (via device manager).
- Integrate these registry settings in your BSP registry and finally in the registry as it will be used on your platform. Why is this so difficult? We would expect that if you create a driver with a registry file, then this registry setting is automatically included when the driver is selected in the **Platform Builder**! This is not the case: you need to create manually your entries in the main BSP registry file.
- Detecting that you need to set not only the vendor ID and the product ID, but at least also the Class, SubClass and ProgID before the device manager will load automatically your driver. This was detected with a debug session…

- Detecting that your device driver specific entry functions may only use a prefix of three letters (again only detected with a debug session…); anything more is discarded (so the symbol lookup in your driver DLL fails…).
- Displaying an error “Please Check your SYSGEN variable” meant finally that you need to clean your build and try again, which solved the problem.

Our suggestion for Microsoft is to hire some experienced embedded developers (and BSP developer) who had never used **CE** before in order to help Microsoft and let them observe how they need to struggle with the system, in order to let them understand and have a clear view of how embedded developers think about systems. Then they could improve the **Platform Builder** and make it really intuitive!

Conclusion is that the product has not improved since previous version, so it is still far from where we would like to see it. But as always we have to remark that setting up a BSP is not an easy task. It is not easy for **Compact 7** as it is not easy for any other RTOS. Trying to hide this complexity in a nice GUI seems to be too complicated at this time, although we are seeing the last years some better tools for doing this for other RTOS.

When building Compact 7 applications however, then all good things about Visual Studio apply! The platform builder should therefore only be used for development of the OAL and drivers, leaving application development to the Visual Studio smart device projects.

One interesting tool that was added since the previous release is the “**Windows Embedded Compact Test Kit**” (CTK) included, which can be used for creating a complete test framework for you platform. Again, good documentation on this toolset is lacking. This time you can find also some documentation on the Technet wikis.

5.5 Internet components

Internet components	0	9	10

Windows Embedded Compact 7 has very extensive Internet support.

The internet support is divided into modules and components. These can be found in a structured way among the other catalogue items by just clicking to add them to your OS design. Adding catalogue items is pretty straightforward.

Compact 7 comes with an extensive set of Internet products and tools. It includes, among others:

- An HTTP server to post information. The server supports active server pages, ISAPI extensions and filters.
- A web browser for viewing information. The browser is a miniature version of Internet Explorer. It supports frames, tables, Javascript, as well as JPEG, static and animated GIF and WAV files, and flash.
- A telnet server to remotely administer devices, or to administer devices that do not have displays.
- Networking protocol support for communicating across the internet/intranet.
- VOIP related drivers and applications
- Cell (mobile phone) related services (like WAP browser, SMS functionality…)
- Firewall, parent-control, …
- RAS, PPP, VPN, …

☺ So almost anything you need for a networked device is delivered with **Compact 7**.

5.6 Development tools

Development tools	0	8	10

Platform Builder *has not improved over its predecessor in terms of user friendliness. So, the product is still far from perfectness.*

Its integration with **Visual Studio 2008** *is new, but it does not add new functionality. The kernel debugger is very easy to use and works very well in debugging applications including the kernel and device drivers! However, it is not easy to get this up and running.*

The remote tools are now available in separate application. However, besides some nice GUI stuff, we did not have a good experience with it.

The ARM emulator, which worked very well in previous release, is gone. Instead you can run a virtual machine using **VirtualPC***. However this works less smoothly.*

Microsoft provides development tools to cater the needs of two developers' categories: the platform developers and the application developers.

The **Platform Builder** is now integrated with **Visual Studio 2008**.

The **Platform Builder** can be used to create a custom **SDK (Software Development Kit**) based on the **Compact 7** OS, in order to allow developers to write applications that run on the target platform. An **SDK** is a set of library, header, and "help" files that developers use to write applications for a specific platform. The **SDK** is used in conjunction with the **Visual Studio** for creating, debugging and running custom applications.

The debugging tools for debugging a remote target (using a network connection) works very well as long you are enabling the KITL (Kernel Independent Transport Layer) your kernel. The tools are intuitive and can be used for debugging the kernel, device drivers and applications, at least when you are using the **Platform Builder**. Remark that it is not simple to create a build having these debug features enabled.

☺ This makes it simple to debug even a device driver (which is otherwise a daunting task)! The integration with **Visual Studio** clearly improves these debugging tools. They can save a lot of your costly developing time!

😐 The thing that is still lacking in the editor is the symbol cross referencing. This feature works for the applications (when using the **SDK**), but not for device drives, services and other kernel related source code. This is a pity…

☹ The remote tools, which are completely reworked with this release, are now delivered in a separate toolset. However this is not an improvement. Engineers are not interested in better GUI tools; On the contrary, they want a tool that is easy to be used and works perfectly. They need working tools not nice looking tools with amazing GUI. The problems we had with the remote tools are:

- First, there is no easy way in the **Platform Builder** where you can select the option of enabling the remote tools. You need to adapt your target build image so that all the required components are included (at least, this time we found a list of SYS_GEN variables that need to be included in the documentation, but to translate this to module names of **Platform Builder** is a difficult task…)
- Making the connection between the remote tool and the target takes so long (a minute or so); at the beginning, we thought that this tool is not working. Hey, why does this need to take so much time? And if it takes so much time, show us what is happening. If today any website does not respond within a minute, then for us it is unreachable and we will not wait longer… a minute is a very long time on a LAN!

When using the **SDK** approach, then developing applications comes very close to the way applications are developed in the normal Windows developing environment as they all use **Visual Studio** features.

Then, it is a good idea to create as soon as possible an **SDK** for application development, which in turn will save your development time.

The table below makes an inventory of the most commonly used tools and features available:

	Present (Yes/No)	Integrated in IDE	Standalone	Command based	GUI based
Editor					
Color highlight	**Yes**	✓			✓
Integrated help	**Yes**	✓			✓
Automatic code layout	**Yes**	✓			✓
Compiler					
C	**YES**	✓	✓	✓	✓
C++	**YES**	✓	✓	✓	✓
Ada	**No**				
Assembler	**Yes**	✓	✓	✓	✓
Linker					
Incremental	**Yes**	✓	✓	✓	✓
Symbol table generation	**Yes**	✓	✓	✓	✓
Development tools					
Profiler	**Yes**		✓	✓	

	Present (Yes/No)	Integrated in IDE	Standalone	Command based	GUI based
Project management	**Yes**	✓			✓
Source code control	**Yes**	✓			✓
Revision control	**Yes**	✓			✓
Debugger					
Symbolic debugger	**Yes**	✓			✓
Thread sensitive debugging	**Yes**	✓			✓
Mixed source and disassembly	**Yes**	✓			✓
Variable inspect	**Yes**	✓			✓
Structure inspect	**Yes**	✓			✓
Memory inspect	**Yes**	✓			✓
Register inspect	**Yes**	✓			✓
Target connection					
JTAG	**Depends platform**				
BDM	**No**				
Serial (via ROM-Monitor or app?)	**Yes**				✓
Network (via ROM-Monitor or app?)	**Yes**				✓
System analysis tool					
Tracing	**Yes**	✓	✓		✓
Thread information	**Yes**	✓	✓		✓
Interrupt information	**Yes**		✓		✓
Loader					

http://www.dedicated-systems.com
Email: info@dedicated-systems.com

	Present (Yes/No)	Integrated in IDE	Standalone	Command based	GUI based
TFTP boot loader	**Yes**	✓	✓		✓
Serial boot loader	**Yes**	✓	✓		✓
Load separate modules	**Yes**	✓			✓

5.7 Support

Support	0	8	10

During installation, customers have free support. They have many fee-based support options with standard SLAs...

Whenever urgent issues exist, Microsoft use a process called QFE Quick Fix Engineering) which first delivers a hotfix privately to the impacted customer (solving the immediate need) and then patches the OS for all other customers. The Windows Embedded Developer Update (WEDU) is a Visual Studio component installed by Compact 7 which delivers these updates directly to the developer

6 Appendix A: Vendor comments

All vendor comments were integrated within the document as there were no disagreements.

7 Appendix B: Acronyms

Acronym	Explanation
API	Application Programmers Interface: calls used to call code from a library or system.
BSP	Board Support Package: all code and device drivers to get the OS running on a certain board
DSP	Digital Signal Processor
FIFO	First In First Out: a queuing rule
GPOS	General Purpose Operating System
GUI	Graphical User Interface
IDE	Integrated Development Environment (GUI tool used to develop and debug applications)
IRQ	Interrupt Request
ISR	Interrupt Servicing Routine
MMU	Memory Management Unit
OS	Operating System
PCI	Peripheral Component Interconnect: bus to connect devices, used in all PCs!
PIC	Programmable Interrupt Controller
PMC	PCI Mezzanine Card
PrPMC	Processor PMC: a PMC with the processor
RTOS	Real-Time Operating System
SDK	Software Development Kit
SoC	System on a Chip