Porting BSP to WinCE 6.0

System Analysis

Prepared by: CC Team
Last update: 5th December, 2008

Revision: 1.03

Revision	***Description***	***Date***	***By***
1.01	First Draft	03 October, 2008	CC Team
1.02	Device driver related details added	10 November, 2008	CC Team
1.03	Porting BSP details added	05 December, 2008	CC Team

Table of Contents

1. Introduction

A new version of Windows CE triggers changes in CE projects' design necessary to fit new system requirements. In opposite to previous versions of Windows CE adjustments are significant as a result of changes in kernel architecture.

1.1. Purpose

The purpose of the paper is to present differences between Windows Embedded CE 6.0 and previous version of the Windows CE OS. It also includes information about all potential parts of project/code that should be modified to convert CE 5.0 compliant BSP implementation to a new Windows Embedded CE 6.0.

1.2. Scope

The analysis concerns AMD Alchemy Board Support Package for Windows CE 5.0 delivered by BSQUARE Corporation to be used with RMI Alchemy™ DBAu1550™ Development Board.

1.3. Definitions, Acronyms and Abbreviations

SDB	Standard development board
BSP	A board support package (BSP) is software that implements and supports an operating system (OS) on SDB.
KITL	Kernel Independent Transport Layer
OEM	Original Equipment Manufacturer
OAL	OEM adaptation layer

1.4. References

1.4.1. BSQUARE Site

RMI Board Support Packages
http://www.bsquare.com/amd/bsp/amd.asp
Windows CE 6.0 Blog – *User Mode and Kernel Mode Drivers*, by Dean Ramsier http://www.bsquare.com/blog/default-aug_21.asp

1.4.2. Raza Microelectronics, Inc. Site

MI Alchemy™ Au1550™ Processor
http://www.razamicroelectronics.com/products_alchemy/au1550_overview.htm
The RMI Alchemy™ DBAu1550™ Development Board
http://www.razamicroelectronics.com/products_alchemy/au1550_dev_product_brief.htm

1.4.3. Microsoft MSDN Library for Windows Embedded CE 6.0
Windows Embedded CE
http://msdn2.microsoft.com/en-us/library/ms950422.aspx
Developing a Device Driver
http://msdn2.microsoft.com/en-us/library/aa910457.aspx
Developing an Operating System
http://msdn2.microsoft.com/en-us/library/aa923966.aspx

1.4.4. MSDN Magazine, the Microsoft Journal for Developers
Explore The New Features In Windows Embedded CE 6.0 by Paul Yao http://msdn.microsoft.com/msdnmag/issues/06/12/WindowsCE/

1.4.5. Windows Embedded MSDN Blog
CE 5.0 application Compatibility on CE 6.0, by Mike Hall
http://blogs.msdn.com/mikehall/archive/2006/10/12/ce-5-0-application-compatibility-on-ce-6-0.aspx
CE6 Drivers: What you need to know, by Sue Loh
http://blogs.msdn.com/ce_base/archive/2006/11/09/CE6-Drivers_3A00_-What-you-need-to-know.aspx

1.4.6. MEDC – Microsoft Mobile & Embedded DevCon 2006
Windows CE 6.0 Architecture, by Douglas Boling, President Boling Consulting Inc. http://download.microsoft.com/documents/australia/medc2006/Windows_CE6_Architecture_Boling.ppt

1.4.7. WindowsForDevices site
Differences between Windows CE 5.0 and Windows CE 6.0, by K. Ashok Babu http://www.windowsfordevices.com/articles/AT9457847627.html

2. Overview of Windows Embedded CE 6.0 changes

2.1. API Deployment Scheme

In Windows CE 1.0 – 5.0 versions, the Win32 APIs were deployed using client/server architecture to connect a caller to the target Win32 function through an **interprocess** communication mechanism. So many context switches resulted in performance penalties.

In new CE 6.0 model:

– System APIs are moved out of their own user-mode processes and are placed into kernel-mode DLLs instead that results in some performance improvements.
– Previous limitations on total number of processes (32) and the small virtual address space (32MB) are removed.
– OEM code is separated from operating system code. They are now built into two separate modules: the kernel operating system (kernel.dll) and the OEM Adaptation Layer (nk.exe).

2.2. Kernel-Mode Operation

To run Windows CE, a **CPU must support two privilege levels**:

– Kernel mode – the higher privilege level;
– User mode – the lower privilege level.

Putting code in user mode helps the overall environment run more robustly and more securely. However, mixing user-mode and kernel-mode code is generally slower than running the system entirely in kernel mode.

With CE 6.0, only the mixed operating mode is supported with all applications being loaded into user-mode memory, and all OS components loaded into kernel-mode memory.

To minimize the cost of calling across the privilege boundary some components have both a user-mode and a kernel-mode incarnation. For example *core system library*:

– coredll.dll is user-mode
– k.coredll.dll is kernel-mode.

The expense is a larger operating system image.

2.3. Memory Architecture

Windows CE 1.0 – 5.0 Limits:

– Single virtual address space is divided into the 32 slots, with no overlap between process address spaces,

- 32 processes at any one timed, each of which occupied its own process slot,
- 32 MB Virtual Memory per process,
- Upper half of user space is shared memory – Read / Write by all processes.

Windows CE 6.0 kernel brings a new memory architecture that eliminates the previous limits:

- **Each process** gets its own, **truly private process address space**. As a result no application process can look into the address space of any other application process (standard Win32 shared memory APIs need to be used to enable two processes to read and write to the same memory).
- **32K processes** – a theoretical maximum. The practical maximum is lower because other types of kernel objects occupy space in the kernel handle table.
- **2 GB of Virtual Memory per process** – larger address space per process,

Impact of the new memory architecture on existing software

Well-behaved applications should just run in their unmodified binary form on CE 6.0, since memory is allocated using the same allocation APIs, and data is still stored using 32-bit virtual memory pointers. In CE 6.0 large data blocks allocations can be satisfied with a call to VirtualAlloc instead of shared memory allocation functions (ex. MapViewOfFile) from the earlier CEs.

Generally speaking, **most changes are limited to device drivers**:

- Drivers that still need to access (read/write) data in an application's address space need to run in kernel mode.
- Device driver could be run in user mode through the use of a user-mode device driver process-udevice.exe.

2.4. Device Emulator

Windows CE has seen three generations of emulators:

- **The first generation emulator** supported Windows CE by directly calling the roughly equivalent Windows NT functions. It was a good first start, but left much to be desired.
- **The second generation** emerged as a custom embedded platform using hardware virtualization, running the same low-level instructions as a real device. It was a very capable emulator, but had a drawback for developers who were focused on deploying to non-x86 processors since it was built using the I86 instruction set.
- **A third-generation emulator**, which ships with CE 6.0, provides instruction-level emulation of the ARM V4I instruction set. This emulator ships with the CE 6.0 Platform Builder as the Device Emulator.

This newest generation supports the ARM V4I instruction set. While the x86 instruction set dominates on the desktop, it has not made the same inroads in the embedded world. That honor belongs to the instruction sets designed and licensed by ARM Holdings, Ltd. of the UK. **The benefit of emulation at the machine instruction set level is binary compatibility**. Instead of building the x86 executable files for the emulator and a second set of ARM executables for actual devices, you can create a single set of binaries and run them both. In addition to being more convenient, it also enables a higher degree of testing confidence because you test everywhere with the same executables without worrying about errors in your setup scripts, or compiler or linker bugs.

2.5. Cell Technologies

To enable machine-to-machine communication, CE 6.0 includes the interfaces needed to connect to mobile phone networks, support for making phone calls and sending SMS (text) messages. CE 6.0 provides:

- **cellcore.dll** – extends the Win32 API to support a variety of mobile phone services such as initiating a data connection, sending an SMS message, and so on,
- **ril.dll** – the driver for the Radio Interface Layer (RIL). It is a low-level interface for connecting an application layer to the mobile phone hardware.
- Low-level components for the Wireless Application Protocol (WAP), including a kernel-mode driver (**wapdrv.dll**) and a user-mode API (**wap.dll**).
- HTML-only browsers (no support for WAP browsing).
- Rich set of SMS service providers to enable the receipt from the mobile phone network of SMS broadcast messages, notification messages (for services like voice mail, fax, and e-mail), and status messages.

2.6. Location-Based Programming

Microsoft introduced support for Global Positioning System (GPS) APIs with Windows Mobile 5.0. Those same APIs, and the drivers to support them, are now available with CE 6.0.

2.7. Security Enhancements

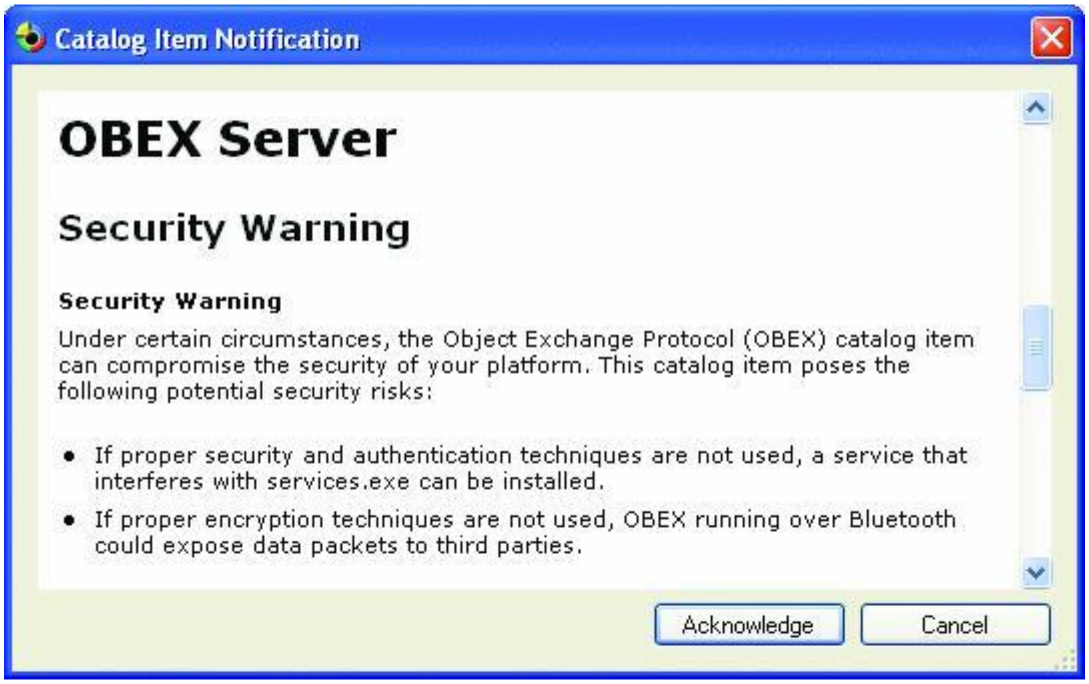
The security features that already exist in Windows CE:

- Ability for a Windows CE-powered device to maintain **tight control** over **which applications and DLLs are allowed to load and run** (it is the centerpiece of Windows CE security). Through the OEMCertifyModule function, a device can prevent any unauthorized code from running. A common method for identifying authorized modules is through the use of digital certificates. Device security can be set up in a variety of different configurations. For example, in one configuration any unknown modules-those without valid certificates-can be denied any system access. Alternatively, this mechanism can be turned off so that all modules have complete access to all system services.
- Another core security feature is the **Cryptographic API**, which allows applications to encrypt and decrypt blocks of data using a variety of encryption algorithms. Encryption algorithms are identified by symbolic names with the prefix CALG_, such as CALG_DES and CALG_AES.
- Support for **secure sockets layer (SSL)**, which provides secure HTTP connections.
- **Virtual Private Networking (VPN)** is supported through the point-to-point tunneling protocol (PPTP).
- To enable secure connections with server systems, Windows CE provides support for a variety of **authentication mechanisms**, including the **Windows NT LAN Manager** protocol and the more robust **Kerberos authentication** protocol.
- The **Credential Manager**.
- Support for **Smart Cards**.
- The **Data Protection API** (DPAPI).
- Support for **public key certificates** (PKI).
- The **Local Authentication Subsystem** (LASS).

The new security features with CE 6.0:

- There is a **strict separation** of **user-mode code** from **kernel-mode code**. Enhanced validation checks of the Protected Server Library (PSL) and I/O Control's (IOCTL) parameters that are passed from user mode to kernel mode are performed, improving kernel mode security and stability.

- The perimeter security of a system can be vastly improved by supporting a **secure loader**. A secure loader ensures that only trusted code gets to run on a system. In Windows CE 5.0, the operating system provided a loader hook but OEMs had to write their own secure loader. CE 6.0, in contrast, ships with a **built-in secure loader**. The trust decisions of the loader are certificate-based-this means that all code that runs on the system has to be signed. The secure loader, if enabled, inspects the code signature and if the signature is signed by a trusted certificate then that code is allowed to run. If not, the module load fails. The OEM has control over what certificates are trusted and thus has control over the code that gets to run on the system.
- Another area in which **security improvements** have been made is the **Windows CE OS Design (or the New Platform) Wizard.** When a platform is configured with a feature that may compromise the security of the device, a security warning is issued. For example, the following Security Warning from the Design Wizard shows the notification displayed when the Object Exchange (OBEX) Protocol is included in a platform. Details of the potential compromise are provided, to help platform developers address the potential problems early in the platform development process.



OBEX Server

Security Warning

Security Warning

Under certain circumstances, the Object Exchange Protocol (OBEX) catalog item can compromise the security of your platform. This catalog item poses the following potential security risks:

- If proper security and authentication techniques are not used, a service that interferes with services.exe can be installed.
- If proper encryption techniques are not used, OBEX running over Bluetooth could expose data packets to third parties.

- **Support for a secure boot loader** – this feature ensures that downloaded operating system (NK.BIN) images contain valid digital signatures before allowing the OS images to be installed and run on a system.

2.8. Development Tools

Prior to CE 6.0, the Windows CE team shipped a standalone product called Platform Builder. With CE 6.0, the platform development tools are integrated into Visual Studio® 2005. Visual Studio is, of course, the premier development tool for both client and Web development; now this same tool is available to support Windows CE. Features that have benefited application developers-like IntelliSense®, syntax checking, syntax highlighting, outline view, and function completion-can be used in the development of custom Windows CE platforms.

Some Platform Builder terms have changed to better fit into the Visual Studio paradigm:

- A Windows CE 5.0 Workspace is now a Solution in CE 6.0.
- A Windows CE 5.0 Project (an IDE-defined application or DLL) is now a Subproject in CE 6.0.

2.9. Shared Source and Windows CE Development

With CE 6.0, Microsoft continues a program that began with Windows CE 3.0 – the shared source program. Taking a cue from the open source movement, Microsoft began by making available a significant portion of the source code for Windows CE to anyone willing to download the Evaluation edition of the Windows CE Platform Builder.

Developers who build applications to run on Windows CE have gotten tangible benefits from other open source projects. Perhaps the best known is the OpenNETCF project (available for download), an extension to the .NET Compact Framework.

Microsoft has directly sponsored several source code projects such as the Bluetooth project, which provides managed code classes to simplify the use of Bluetooth in Windows CE-based applications. Another is the webcam driver used to kick-start designs with webcams when no drivers were publicly available. A third project sponsored by Microsoft is the digital video recorder (DVR) project, to support the creation of Windows CE-powered video recorders.

3. Device Drivel related details

3.1. OS Layout

In all Windows CE 1.0 – 5.0 versions, the Win32 APIs were deployed using client/server architecture similar to that found in early versions of Windows NT.

In Windows NT 3.5, for example, all GUI functions reside in a single process: csrss.exe, the client-server runtime subsystem. When functions like CreateWindow or TextOut are called, the CSRSS process is triggered through an **interprocess** communication mechanism called local-procedure calls (LPCs). Owing to performance penalties of so many context switches between the heavyweight Windows NT processes, this architecture was formally abandoned in Windows NT 4.0 (released in 1996). At the time, all the formerly user-mode API libraries were transferred into kernel mode.

A similar change is taking place with the release of Windows Embedded CE 6.0. For all previous versions, Windows CE also used a client/server mechanism to connect a caller to the target Win32 function. There are similarities, but it is important to note the differences between the Windows CE implementation and the one on Windows NT. One difference is that Windows CE used a much more lightweight process than Windows NT, and so the performance cost was not as high. Unlike Windows NT, Windows CE was designed to be configurable. And the use of processes to deploy system APIs allowed the demands for ROM and RAM to be gated by how much of the overall operating system was needed for a specific device. For example, a device could be configured to run with just one API process, the system kernel (nk.exe). Other devices that need GUI support would also need to run the GUI (gwes.exe) process.

Although the new model does provide some performance improvements, the real motivating factor for making the change is to remove limitations that could bottleneck development on the next generation of devices. Those limitations, which are well known to Windows CE programmers, include the limit on total number of processes (32), and the small virtual address space (32MB) of previous generations of Windows CE kernels.

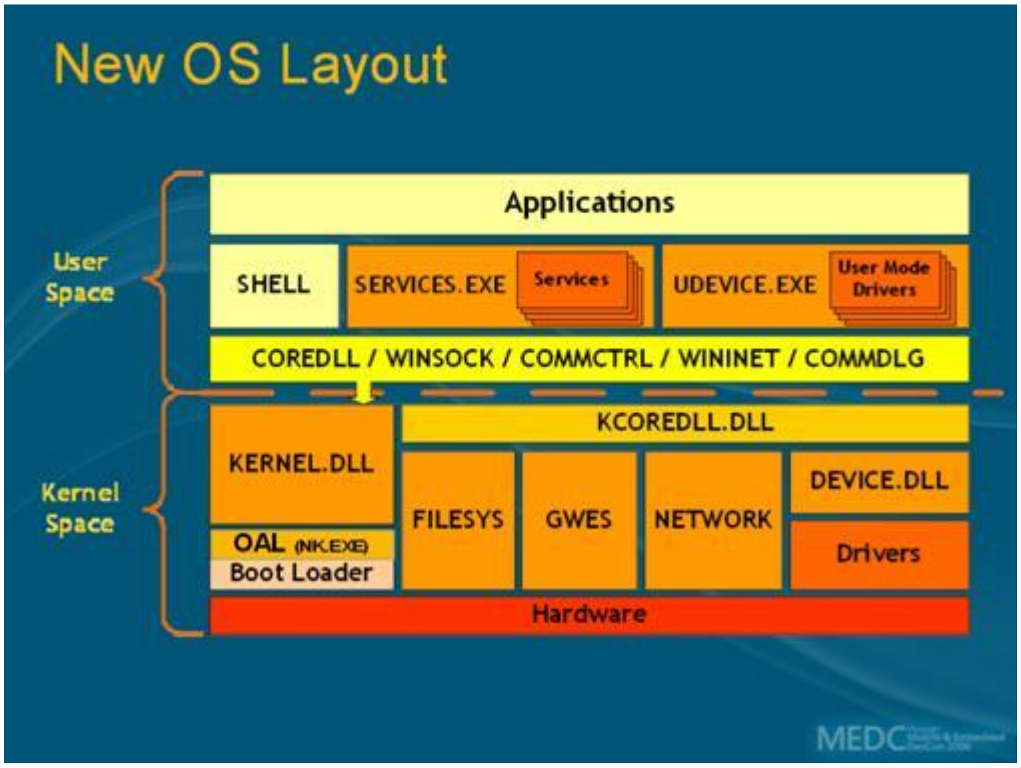
Many things have changed in the decade since the first kernel was deployed. Devices today can have more capable hardware: faster CPUs, more storage, a color LCD display screen, and so on. Today's designs don't need the tiny memory footprint of yesterday's models, and so a redesign was in order. With CE 6.0, system APIs are moved out of their own user-mode processes and are placed into kernel-mode DLLs instead.

Changes to Major Modules in Windows Embedded CE 6.0

Windows CE 5.0 Process	Windows Embedded CE 6.0 DLL	Description
nk.exe (OAL + kernel)	nk.exe (OAL) kernel.dll (kernel)	OEM code is getting separated from CE kernel code starting with CE 6.0
filesys.exe	filesys.dll	Registry, file system, and property databases
device.exe	device.dll	Manages kernel-mode device drivers

device.exe	udevice.exe	New to CE 6.0, a separate process to manage user-mode device drivers
gwes.exe	gwes.dll	Graphical and windowing event subsystem
services.exe	servicesd.exe	Host process for system services
services.exe	services.exe	Command-line interface to configure services

While this reorganization of the operating system was underway, another important change was made: OEM code was separated from operating system code. Previously, a hardware designer created a low-level set of routines called the OEM Adaptation Layer (OAL), and that component was statically linked to the operating system kernel. The OAL and the kernel appeared as a single executable, nk.exe. In CE 6.0, these two components are now built into two separate modules: the kernel (kernel.dll) and the OEM Adaptation Layer (nk.exe).



Kernel-Mode Operation

To run Windows CE, a **CPU must support two privilege levels**:

- Kernel mode – the higher privilege level;
- User mode – the lower privilege level.

Any memory that is allocated for either code or data is assigned to one of these two modes. Putting code in user mode helps the overall environment run more robustly and more securely. However, these benefits are not free, because mixing user-mode and kernel-mode code is generally slower than running the system entirely in kernel mode.

In previous versions, Windows CE could be configured for all kernel-mode operation, or for mixed-mode operation using both kernel mode and user mode. With CE 6.0, only the mixed operating mode is supported with all applications being loaded into user-mode memory, and all OS components loaded into kernel-mode memory.

User-Mode and Kernel-Mode System Libraries

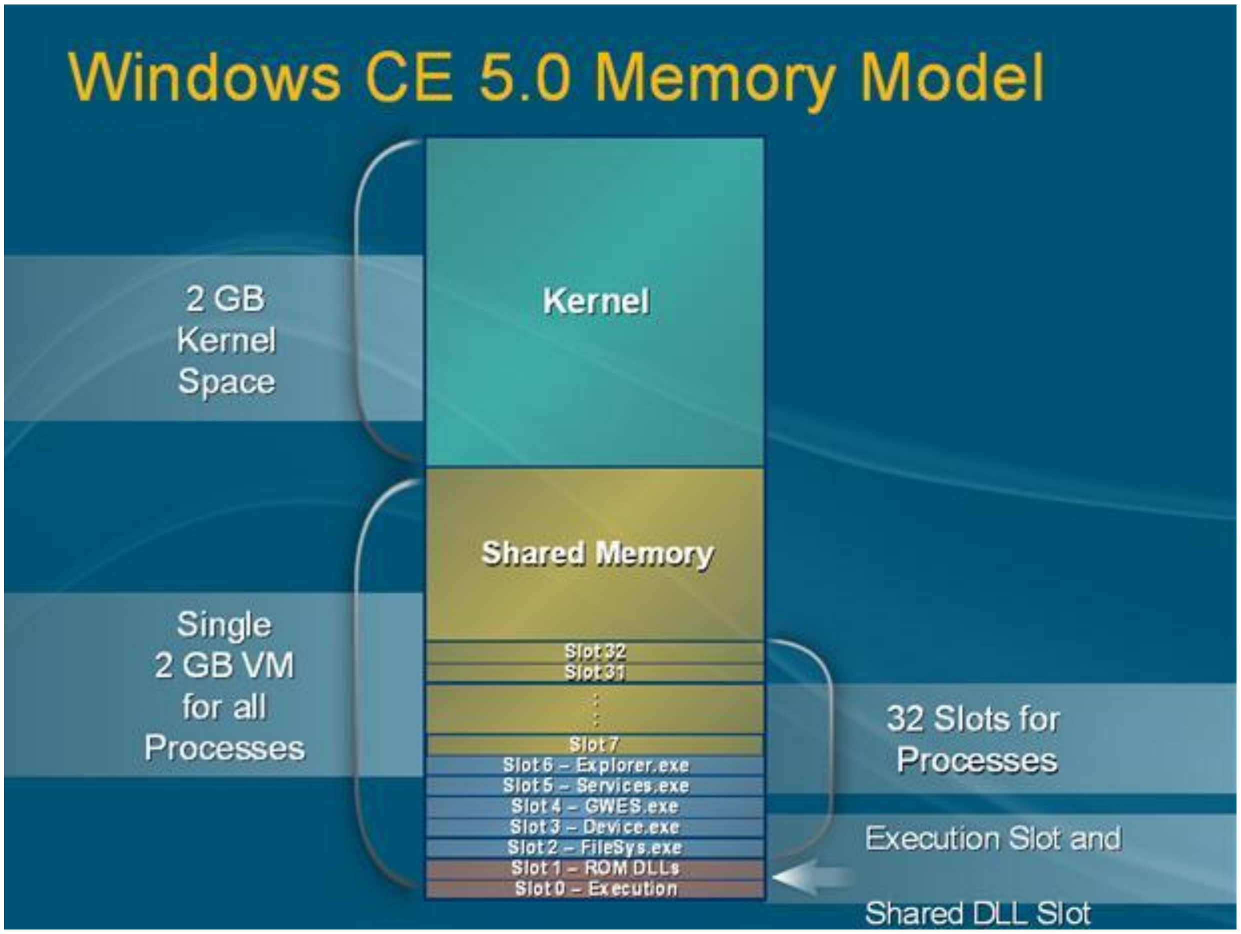
Component	User-Mode	Kernel-Mode
Core system library	coredll.dll	k.coredll.dll
Device driver helpers	ceddk.dll	k.ceddk.dll
Credential provider	credsrv.dll	k.credsvr.dll
IP network helper API	iphlpapi.dll	k.kphlpapi.dll
Multimedia timer	mmtimer.dll	k.mmtimer.dll
Process, thread, and memory helpers	toolhelp.dll	k.toolhelp.dll
Windows Sockets	ws2.dll	k.ws2.dll
Windows Sockets service provider interface	wspm.dll	k.wspm.dll

As shown in table, some components have both a user-mode and a kernel-mode incarnation. This deployment helps minimize the cost of calling across the privilege boundary, at the expense of a larger operating system image.

Memory Architecture

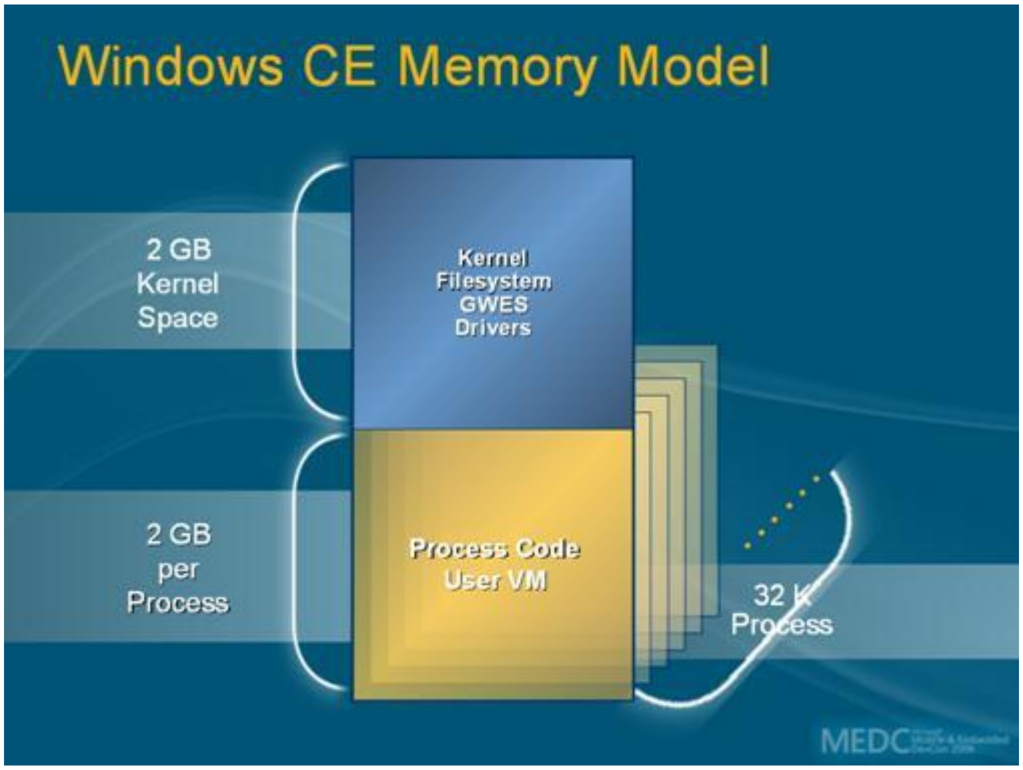
Windows CE 1.0 – 5.0 Limits:

- Single virtual address space is divided into the 32 slots, with no overlap between process address spaces,
- 32 processes at any one timed, each of which occupied its own process slot,
- 32 MB Virtual Memory per process,
- Upper half of user space is shared memory – Read / Write by all processes.



Windows CE 6.0 kernel brings a completely new memory architecture that eliminates the previous limits:

- **Different structure of process address spaces** – each process gets its own, truly private process address space (it appears very similar to the process address space in desktop versions of Windows, such as Windows XP).
- **32K processes** – this is a theoretical maximum, based on a limit on the storage of kernel handles. CE 6.0 can accommodate a maximum of 64K kernel handles, and a process requires a minimum of two handles-one for the process itself, and one for the process's thread. The practical maximum is lower because other types of kernel objects occupy space in the kernel handle table.
- **2 GB of Virtual Memory per process** – larger address space per process,



The new memory architecture impacts existing software
Generally speaking, **most changes are limited to device drivers**. While the size of the process address space is much larger, well-behaved applications shouldn't notice, since memory is allocated using the same allocation APIs, and data is still stored using 32-bit virtual memory pointers. The new memory architecture does make some things easier, particularly for applications that needed to allocate very large blocks of memory-greater than 10MB, for example-such as might be needed to hold a high-resolution image read in from a digital camera sensor. Previously the need for large data blocks had to be satisfied by allocating with the shared memory allocation functions (e.g. MapViewOfFile). Starting with CE 6.0, large allocations can just as easily be satisfied with a call to VirtualAlloc.

Aside from the increase in size of the process address space, the other change introduced with this memory architecture is the truly private process address space. Prior to CE 6.0, it was generally quite easy for one process to look into the address space of another process. In the now defunct slotted address scheme, the currently running process is mapped into slot zero. This was done by fixing up all pointers to reference memory in the range of zero to 32MB. By calling a function like MapPtrProcess, the currently running process can take a pointer from another process and access that memory directly in the process slot (in the range 32MB-64MB for slot 1, 64MB-96MB for slot 2, and so forth). With the implementation of a private process address space, no application process can look into the address space of any other application process. Instead, you will need to use the standard Win32 shared memory APIs to enable two processes to read and write to the same memory.

One reason that this new architecture is likely to have the biggest impact on device drivers is that existing Windows CE device drivers often use functions like MapPtrProcess to read or write directly in an application's address space. In CE 6.0, drivers that still need to access data in an application's address space need to run in kernel mode. This is accomplished by linking to the appropriate set of kernel libraries (k.coredll.dll) instead of their user-mode counterparts.

In some cases, it might be necessary for a device driver to run in user mode instead of in kernel mode. For example, there might be a driver for which you don't want to grant full kernel mode privileges. Such a driver can be run in CE 6.0 through the use of a user-mode device driver process-udevice.exe.

3.2. Device Drivers model

Windows Embedded CE 6.0 introduces new **third-generation kernel architecture** to support kernel mode drivers.

In Windows CE 1.0 – 5.0 device drivers were executed in the device.exe that was a user-mode process like other applications with the following shortcomings:

- User-mode memory limitations.
- Decreased performance due to inter-process calls between the application, driver, file system and kernel.
- Risk of shutting down the entire process with all drivers by a corrupt driver (all drivers were located in the same process space).

CE 6.0 introduces two different device driver models to increase system performance or robustness and security:

- **Kernel-Mode Drivers** – functionality of the old device.exe moved into the kernel,
- **User-Mode Drivers** – introduced a new user device process (udevice.exe).

3.2.1. Kernel-Mode Drivers

In CE 6.0, Filesys.exe (file system), device.exe (device drivers), and GWES.exe (Graphics, Windowing, and Events Subsystem) have been moved from user mode to kernel mode. As a result, user-mode process initiating a driver access does not need to be switched out, because it coexists (is resident in memory at the same time) with the kernel.
Kernel-mode drivers are loaded inside of the kernel by **device.dll** that resembles deprecated device.exe model. Elimination of expensive inter-process calls increases performance.
However, as a part of the kernel, kernel-mode drivers have full privileges for the entire kernel address space. Therefore they must be very well tested and robust, because some memory corruption caused by defective driver could easily crash the kernel and entire system.

3.2.2. User-Mode Drivers

User-mode drivers are still supported through a user mode driver manager called udevice.exe, interestingly; with CE 6.0 you can have multiple instances of udevice.exe, each hosting a single user-mode driver (or a group of user-mode drivers). Such drivers running in their own processes are isolated from the kernel and the rest of the system, so in case of failure only their copies of udevice.exe are affected while the rest of the system remains untouched.

In this mode drivers loose some performance, but overall system robustness and security is increased.

To load a driver into user-mode instead of the default kernel-mode, a new bit defined in the Flags registry key must be set. User-mode driver model has very few differences to kernel-mode drivers so user-mode driver can be loaded into kernel-mode with no changes. To hide from applications how a particular driver was loaded, communication between user-mode applications and the user-mode driver pass through the user-mode driver reflector (a new kernel component), which helps with buffer marshalling. The reflector also assists the user-mode driver with operations that user-mode code is not normally allowed to make, like mapping physical memory.

Drivers running inside a user-mode process follow their restrictions in use of certain APIs such as VirtualCopy (ex. limited to use it for only the memory space defined in the registry to restrict arbitrarily accessing any hardware resource in the system).

Besides, SetKMode function and setting process permissions (to access other processes), are no longer supported. Consequently, drivers using such techniques have to be revised.

Summing up, BSP developers could now choose between:

- High performance kernel-mode driver,
- More protected user-mode driver – suitable for untrusted third party drivers or unstable drivers that can be tested in user-mode and then moved to kernel-mode.

4. Changes required to port BSP

A board support package (BSP) is a software that implements and supports an operating system (OS) on a standard development board (SDB). It typically consists of a boot loader, OEM adaptation layer (OAL), device drivers, and run-time image configuration files.
This point will describe a process of porting BSP for „RMI Alchemy™ DBAu1550™ Development Board" dedicated for Windows CE 5.0 to new IDE with support for Windows Embedded CE 6.0. Information about required changes in OAL and devices drivers will be described in details in next point of the document.

4.1. Introduction

The BSP porting analysis will be cut down to "RMI Alchemy™ DBAu1550™ Development Board". This development board is an integrated hardware and software system that leverages the power of the Au1550™ security network processor - a MIPS-based™ System-on-a-Chip (SoC) with integrated security technology developed by SafeNet for wired and wireless applications where security is important. The DBAu1550 development board is a turnkey solution that brings processor, memory and peripherals together on a single board, along with debugging assist features and software.
The processor accelerates networking and remote access applications such as gateways and network addressable storage (NAS) units, wireless access points and Voice over Internet Protocol (VoIP) environments. Low power consumption enables the Au1550 processor to support battery and Power-over-Ethernet applications.
The DBAu1550 development board underscores RMI's commitment to providing customers with integrated systems that address their hardware and software needs, and positions RMI as a valued partner in the development of future access and network solutions.

4.2. Current state

There are two versions of BSP dedicated to the hardware described above: for Windows CE 4.2 and for Windows CE 5.0. For further analysis we assume that the conversion will be done for Windows CE 5.0.

4.3. Limitations

The porting process will be done with the following assumptions:

- The new BSP will be based on one of existing BSPs for MIPSII processor delivered by Microsoft together with Platform Builder for CE 6.0 plug-in to Microsoft Visual Studio 2005
- The analysis is done basing on available BSP sources for Windows CE 5.0 for „RMI Alchemy™ DBAu1550™ Development Board" with RMI Db1550 processor.
- Migration will be done according to requirements and recommendations from Microsoft.
- Migration will be done for new IDE created for Windows Embedded CE 6.0 – Platform Builder for CE 6.0 for Microsoft Visual Studio 2005

4.4. Changes

The easiest way to create a BSP is to clone the existing BSP that is designed for similar hardware. Platform Builder includes a BSP for each supported CPU and for associated core logic or chipsets.

The following list presents various aspects that should be taken into consideration during migration process from Windows CE 5.0 to Windows Embedded CE 6.0 and to a new IDE (Microsoft Visual Studio 2005 with Platform Builder for CE 6.0 plug-in).

4.4.1. Boot Loader and KITL changes

In previous releases, exception handling using __try / __except was not supported in boot loaders and KITL. These components are linked to the same limited functionality libraries. The libraries defined exception handling symbols that caused a link-time build break if exception handling was used.

In Windows Embedded CE 6.0, exception handling has been enabled for the KITL component, but not for the boot loader (which runs prior to kernel initialization).

To prevent random crashes in the boot loader phase, you must explicitly ensure you do not use exception handling in the boot loader code.

4.4.2. OAE changes

Minimizing changes to the OAL is one of the goals for the new BSP design in Windows Embedded CE 6.0. To meet this goal in Windows Embedded CE each module provides a table of function pointers for use by the other modules:

- Kernel exports NKGLOBAL with pointers to functions and global variables. To prevent other modules code reorganization was added stub library called NkStub, which exports all the functions defined in the NKGLOBAL structure. This for example allows any OAL call to an exported kernel function to be automatically routed through NKGLOBAL.
- OAL exports an OEMGLOBAL table with OAL functions and global variables. As with NKGLOBAL, there is a wrapper library oemstub.lib that wraps the OAL exports.

Besides nkstub.lib and oemstub.lib, there are a few additional libraries in CE6 for OEM use:

- Kitlcore.lib: This is a replacement for kitl.lib, implementing the KITL protocol and the initialization of the kitl.dll interface with the NKGLOBAL and OEMGLOBAL structures.
- Nkldr.lib: This library implements KernelInitialize / KernelStart, to link into oal.exe.
- Oemmain.lib: This library implements the OEMInitGlobals function which exchanges function pointers with the kernel.

4.4.3. Separate kernel, OAL, and KITL

In Windows Embedded CE 6.0, the kernel and OEM code are divided into the following three components:

- Oal.exe - contains the startup code and the OEM adaptation layer (OAL) implementation. This component was previously named Kern.exe.
- Kernel.dll - contains the OAL-independent kernel implementation.
- Kitl.dll - contains the platform-specific KITL support.

The goals for this design are to:

- Minimize OAL changes
- Clearly define the functions that the OAL can and cannot use
- Provide version information between components to provide forward compatibility

This model provides the following advantages:

- Allows Microsoft to update the kernel for retail devices without going through the OEM because the kernel is no longer dependent on the OAL.
- Simplifies the debugging process with an instrumented kernel. You only need to include a new instrumented Kernel.dll into the release directory and then re-make the run-time image.

- Provides OEMs with a binary-only version of the platform, which they can give to their customers.
- Makes loadable KITL possible.
- Enables all communication between the OEM code and the kernel to take place through well-defined interfaces.

4.4.4. Adopt the new OAL directory structure

In Windows Embedded CE 6.0, the OAL directory structure in Windows CE 5.0 was modified to reflect the separation of the OAL, KITL, and the kernel. Microsoft recommends that OEMs adopt the new directory structure when migrating their BSPs to Windows Embedded CE 6.0.
Microsoft recommends that OEMs adopt a new directory structure to reflect these changes. This is optional. However, OEMs must use the new names for the executables being built because this is not optional.

Recommended BSP Directory Structure

\Platform\<Hardware Platform Name> subdirectory	Description
Cesysgen	Contains a makefile for filtering any of the configuration files in the files directory.
Files	Contains project-specific files for building the run-time image, initial directory structure, initialized databases, and initialized registry.
Src	Contains the boot loader, OAL, and include files for the hardware platform.
Src\Bootloader	Contains all the boot loader-specific code.
Src\Bootloader\Eboot	Contains the boot loader source files.
Src\Common	Contains all the code common to the boot loader and OAL.
Src\Drivers	Contains the local BSP drivers.
Src\Inc	Contains hardware platform-specific include files.
Src\Oal\OalLib	Contains the hardware platform-specific OAL code.

Src\Oal\OalExe	Contains the build files, and possibly stub functions, for building the basic OAL image (without KITL support or with KITL in Kitl.dll). This builds Oal.exe.
Src\Kitl	Contains the build files and source code for building Kitl.dll.

Previous BSP Directory Structure

\Platform\<Hardware Platform Name> subdirectory	Description
Cesysgen	Contains a makefile for filtering any of the configuration files in the Files directory.
Files	Contains project-specific files for building the run-time image, initial directory structure, initialized databases, and initialized registry.
Src	Contains the boot loader, OAL, and include files for the hardware platform.
Src\Bootloader	Contains all the boot loader specific code.
Src\Bootloader\Eboot	Contains the boot loader source files.
Src\Common	Contains all the code common to the boot loader and OAL.
Src\Drivers	Contains the local BSP drivers.
Src\Inc	Contains hardware platform-specific include files.
Src\Kernel	Contains device-specific source code files for building and linking the kernel and OAL.
Src\Kernel\Kern	Contains build files, and possibly stub functions, for building the basic kernel image.
Src\Kernel\Kernkitl	Contains build files, and possibly stub functions, for

	building a kernel with Kernel Independent Transport Layer (KITL) support.
Src\Kernel\Kernkitlprof	KernKitlProf.exe does not have an equivalent in Windows Embedded CE 6.0. If your OAL implements profiling support, this should always be included in Kern.exe and/or KernKitl.exe.
Src\Kernel\OAL	Contains the hardware platform-specific OAL code.

5. Changes required to port OAL and drivers

5.1. Introduction

Additionally to changes required for BSP, it is very important to verify existing OAL layer and device drivers' implementation. In most cases small-scale changes are sufficient to fit new Windows Embedded CE 6.0 requirements. Only for non-standard programming solutions (or in the case of undocumented functions usage) it might be necessary to redesign internal built or interface.

5.2. Current state

The operations described below affect implementation of OEM and board drivers.

Driver elements list

AUDIO

- Au1550 PSC AC97 Audio
- PSC I2S Audio

DISPLAY

- Alchemy ATI Rage XL
- Silicon Motion Voyager

Local Area Networking (LAN) devices

- Alchemy Ethernet

PCMCIA (PC Card)

- Alchemy PCMCIA

Serial

- Alchemy Serial

USB Host Controllers

- Alchemy USB (OHCD)

USB Function

- Alchemy USB Function

Storage Devices

- Highpoint HPT371/372 IDE Controller
- Au1550 NAND FLASH Controller

SPI Controller

- Au1550 PSC SPI Controller

5.3. Limitations

Further divagations will be done for drivers that are a part of Windows CE 5.0 BSP for „RMI Alchemy™ DBAu1550™ Development Board".

5.4. Changes

For Windows CE 5.0 and earlier, drivers ran in a Device.exe process. For Windows Embedded CE 6.0, kernel drivers run in the NK.exe process. Due to the updated driver model, Windows CE 5.0 and earlier compatible drivers should be modified in order to work properly with Windows Embedded CE 6.0. Stability and security are also extremely important for drivers, as an instable driver in Windows Embedded CE 6.0 can potentially cause the OS to fail. CE 6.0 introduces new support for user mode servers, drivers (managed by user mode version of driver manager: udevice.exe) and services.

5.4.1. Access Checking

In Windows CE 5.0 and earlier, MapCallerPtr was used to validate a region of memory pointed to by a pointer parameter. MapCallerPtr was used in the I/O controls (IOCTLs) to validate a pointer parameter passed by a calling process. Device drivers in Windows CE 5.0 ran with a relatively high privilege, and had adequate access to memory. MapCallerPtr was used to verify both pointer parameters as well as embedded pointers.

In Windows Embedded CE 6.0, the kernel performs a full access check of buffer pointer parameters. This takes the responsibility for pointer parameter validation away from a device driver. However, a driver still must verify that the caller has access to memory addressed by embedded pointers. A driver must use the CEOpenCallerBuffer and CeCloseCallerBuffer functions to verify if the caller has access to the memory that is pointed to by embedded pointers.

CeOpenCallerBuffer can be called with the ForceDuplicate parameter set to TRUE. This allocates a temporary heap buffer in the current process. If you choose to copy an input buffer for security purposes, and use CeOpenCallerBuffer for access checking, you can set ForceDuplicate to TRUE. This allows to perform both the input buffer copy and the access check with one function call.

5.4.2. Marshalling

For Windows CE 5.0 and earlier, the MapCallerPtr function performed marshalling for pointers too. A driver called MapCallerPtr with parameters as well as with embedded pointers to validate and marshal.

For Windows Embedded CE 6.0, marshalling depends on the way a pointer is used: synchronously or asynchronously. If a pointer parameter or embedded pointer is used synchronously, the address space of the calling process is accessible for the duration of a call into the driver. This eliminates any requirements for marshalling.

However, if a pointer is used asynchronously, it is critical that the caller buffer is accessible when the caller's address space is unavailable. This means that direct access is not possible for any kind of asynchronous work after the call has returned. Windows Embedded CE 6.0 includes the CeAllocAsynchronousBuffer and CeFreeAsynchronousBuffer functions for drivers to marshal pointer parameters and embedded pointers when asynchronous access is required.

5.4.3. Secure copy

Performing a secure copy of input parameters is one of the best practices for developing a device driver for Windows Embedded CE. It is not safe for a driver to access the buffer of a caller. It is possible that the caller may be malicious or even poorly written. The solution for these data integrity problems is to perform a secure copy. If a driver must access the buffer of a caller asynchronously, it must call the CeAllocAsynchronousBuffer and CeFreeAsynchronousBuffer functions. This eliminates the need to perform an additional parameter's copy. If a driver accesses parameters synchronously, CeAllocDuplicateBuffer and then CeFreeDuplicateBuffer should be used to secure copy helper functions to copy the buffer of the caller.

If a developer handles embedded pointers by calling the CeOpenCallerBuffer function for access checking, he/she should set the ForceDuplicate parameter to TRUE to obtain a local copy of the buffer of the caller. This allows him/her to avoid an additional function call to CeAllocDuplicateBuffer. The local buffer is then freed upon calling CeCloseCallerBuffer.

5.4.4. Thread permissions

For Windows CE 5.0 and earlier, process server libraries had access to the buffer of a caller since execution took place in the context of the caller's thread. Other threads, such as ISTs did not have access to the buffer of the caller and had to call the SetProcPermissionsfunction to set the internal permissions bitmask for the thread. This enabled access to the address space of the caller's process. The SetProcPermissions function is no longer supported in Windows Embedded CE 6.0, and should be removed from drivers. The CeAllocAsynchronousBuffer function marshals the buffer of the caller into the virtual memory of the kernel. This eliminates the need to change the permission of the thread.

For Windows CE 5.0, a driver should save the permissions of a calling process and then set the driver's thread permissions to include the calling process, when needed. After the calling process finishes, the driver's thread permissions should be reset.

5.4.5. User Interface

For Windows CE 5.0 and earlier, drivers ran in user mode and could display a user interface (UI) with no restrictions. For Windows Embedded CE 6.0, the kernel cannot display a UI and a kernel driver must forward a display request to display the required UI to a user mode component. The CeCallUserProc function should be used in this case. Developer need to create a user mode component DLL file to implement the required UI portion of the code, and export this functionality using a function. The DLL file and function name are passed to CeCallUserProc with in and out buffers to achieve required driver UI functionality.

CeCallUserProc does NOT allow embedded pointers. All arguments must be stored inside the single "in" buffer passed to CeCallUserProc, and return data must be stored in the single "out" buffer. The problem is, if kernel code calls user code, user code cannot use CeOpenCallerBuffer or any other method to get the contents of kernel memory.

5.4.6. Embedded pointers

User-mode drivers cannot receive embedded pointers from the kernel (kernel drivers). The best way is reorganize the communication between drivers to "flatten" the communication structure - all the data have to be stored directly in the IN and OUT buffers instead of referenced via embedded pointers.

5.4.7. Handles

For previous versions of CE, a handle value was global for the whole system. In a lot of cases, handles could be passed between processes. For CE6, handles are unique for each process.

5.4.8. Callbacks from user drivers

Call-backs from a user-mode driver to any kernel process are prohibited. If a developer moves a bus driver to user mode, he/she has to move the client drivers to user mode too. Developer can't have the client driver in the kernel since he/she cannot call back to the bus driver. Bus driver and all of its client drivers should be placed in the same udevice.exe instance, so that the callbacks are all within a single process.

5.4.9. APIs limitations

- SetKMode function - not supported
- SetProcPermissions / GetCurrentPermissions function – not supported
- MapCallerPtr function - not supported
- MapPtrToProcess function - not supported

- VirtualCopy function – in case of calling VirtualCopy in user mode drivers, the reflector will check addresses to make sure your driver is allowed to access requested physical address
- OAL layer can not use kernel functions that are not exported

6. Time estimation

The time estimation below is done for BSP for „RMI Alchemy™ DBAu1550™ Development Board" delivered for BSQUARE Company. Please, note that it is impossible to list general time values for any BSP conversion, because it hardly depends on relevant BSP implementation (especially on the programming techniques used to implement each driver).

Below are listed all actions that need to be done to prepare full BSP conversion from Windows CE 5.0 to Windows Embedded CE 6.0. Timing includes also project conversion required as a consequence of new IDE for Windows Embedded CE 6.0.

6.1. BSP conversion

BSP conversions should be done with compliance of following elements:

- Folder structure changes
- Windows Embedded CE 6.0 changes
- Required IDE changes

These elements are described in point 4 of the document.

6.2. OAL conversion

This issue includes conversion of the layer responsible for assure communication between kernel and hardware. For more details about the issue, please, see point 5.

6.3. Drivers conversion

Drivers' conversion process contains conversion of all drivers that are a part of BSP except OAL driver mentioned above. This step is also described in point 5 of the document.
Here is a list of drivers to be converted:

- **AUDIO -** Au1550 PSC AC97 Audio
- **AUDIO -** PSC I2S Audio
- **DISPLAY -** Alchemy ATI Rage XL
- **DISPLAY -** Silicon Motion Voyager
- **Local Area Networking (LAN) devices -** Alchemy Ethernet
- **PCMCIA (PC Card) -** Alchemy PCMCIA
- **Serial -** Alchemy Serial
- **USB Host Controllers -** Alchemy USB (OHCD)
- **USB Function -** Alchemy USB Function
- **Storage Devices -** Highpoint HPT371/372 IDE Controller
- **Storage Devices -** Au1550 NAND FLASH Controller
- **SPI Controller -** Au1550 PSC SPI Controller

6.4. SDK generating

The issue contains the time necessary to prepare complete SDK for Windows Embedded CE 6.0.

6.5. Testing

Testing process contain debugging and testing of the prepared BSP and SDK with the target board. It should contain also execution of acceptance tests procedure required by customer.

6.6. Estimations

Conversion step	Time [in days]
BSP	10
OAL	14
Driver - Au1550 PSC AC97 Audio	4
Driver - PSC I2S Audio	4
Driver - Alchemy ATI Rage XL	4
Driver - Silicon Motion Voyager	4
Driver - Alchemy Ethernet	4
Driver - Alchemy PCMCIA	4
Driver - Alchemy Serial	4
Driver - Alchemy USB (OHCD	4
Driver - Alchemy USB Function	4
Driver - Highpoint HPT371/372 IDE Controller	4
Driver - Au1550 NAND FLASH Controller	4
Driver - Au1550 PSC SPI Controller	4
SDK	4
Testing and verification	20
Total	**96**