Windows® Embedded Compact

# BSP Porting Guide for Windows Embedded Compact 7

Writers: Wes Barcalow, Scott Johnston, Glen Langer, Ralph Brand, Jina Chan

Technical Reviewers: Travis Hobrla, Vincent Tam

Published: March 2011

Applies To: Windows Embedded Compact 7

## Abstract

Documents changes in the Board Support Packages (BSPs) between Windows CE 5.0, Windows Embedded CE 6.0, and Windows Embedded Compact 7 that include:

- Removal of the folder %_WINCEROOT%\Public\Common\Oak\Csp
- Separation of the kernel from the OEM adaption layer (OAL) and kernel independent transport layer (KITL)
- Separation of drivers into kernel mode and user mode
- New locations for prebuilt binaries, RNE_MDD.lib, VBridge functionality, and some paths in the Sources files

# Contents

# Introduction

The goal of this porting guide is to aid board support package (BSP) developers who are porting a Windows CE 5.0 or Windows Embedded CE 6.0 BSP to Windows Embedded Compact 7. If you want to port a BSP Windows CE 5.0 to Windows Embedded Compact 7, you must first port your BSP from Windows CE 5.0 to Windows Embedded CE 6.0 and then port from Windows Embedded CE 6.0 to Windows Embedded Compact 7. Porting from Windows CE 5.0 to CE 6.0 represents about 90 percent of the overall effort needed, because it is during this process that you must break the kernel into separate libraries and evaluate the drivers as candidates for kernel mode or user mode. The specific steps to port BSPs are listed below together with information about other porting issues.

This document is divided into two sections: porting a BSP from Windows CE 5.0 to Windows Embedded CE 6.0 and porting a BSP from Windows Embedded CE 6.0 to Windows Embedded Compact 7. Windows Embedded CE 6.0 was released in 2006, so additional resources now exist on porting a BSP to Windows Embedded CE 6.0. MSDN, for example, is a good source for supplemental information.

The time required to port a BSP varies depending on which kernel the BSP is being ported from. You may be able to port from Windows Embedded CE 6.0 to Windows Embedded Compact 7 in just hours. However, a full port will most likely take weeks—possibly even months—depending on the complexity of your drivers and OEM adaptation layer (OAL). In general, though, you can port a BSP to a CEBASE configuration in a few days.

## Differences Between CE 5.0 and CE 6.0

Changes introduced in the Windows Embedded CE 6.0 kernel are the source of the greatest amount of effort in porting to Windows Embedded CE 6.0. For the Windows Embedded CE 6.0 kernel, the kernel-independent transport layer (KITL) was redesigned into a separate, optional DLL, and drivers are separated into user mode and kernel mode. The result is that the kernel mode provides better performance but at the cost of stability. A failure in a kernel-mode driver can bring down the kernel also. A user-mode driver offers isolation from the kernel at the cost of performance.

MSDN has additional information on OAL and kernel separation, user-mode drivers, and kernel-mode drivers.

- Microsoft Showcase: Porting a BSP to Windows Embedded CE 6.0 (http://go.microsoft.com/fwlink/?LinkId=153790&clcid=0x409)
- Microsoft Showcase: Porting Drivers to Windows Embedded CE 6.0 (http://go.microsoft.com/fwlink/?LinkId=157392&clcid=0x409)
- Windows CE Base Team Blog - CE6 OAL: What You Need to Know (http://go.microsoft.com/fwlink/?LinkId=153791&clcid=0x409)
- Windows CE Base Team Blog - CE6 Drivers: What You Need to Know (http://go.microsoft.com/fwlink/?LinkId=153792&clcid=0x409)

- Windows CE Base Team Blog - The CE6 OS Differences in a Nutshell (http://go.microsoft.com/fwlink/?LinkId=153793&clcid=0x409)
- User Mode Driver Framework (http://go.microsoft.com/fwlink/?LinkId=153794&clcid=0x409)

## Differences Between CE 6.0 and Compact 7

From a BSP point of view, there are few changes between Windows Embedded CE 6.0 and Windows Embedded Compact 7. The changes consist primarily of the relocation of binaries during build and some altered code.

# Migrating from CE 5.0 to CE 6.0

The steps to port your BSP from Windows CE 5.0 to Windows Embedded CE 6.0 are presented in three sections. The first section, Required Changes, contains updates that you must make to successfully port your BSP. The second section, Design-Dependent Changes, covers changes that you may need to address depending on the BSP that you are porting. The final section, Cleanup, covers the cleanup of deprecated flags, functions, and so on.

The time required to port from the Windows CE 5.0 kernel to the Windows Embedded CE 6.0 kernel varies depending on the complexity of the drivers and the code. Simple drivers, like GPIO and NLED, port very quickly. However, more complex display drivers can take much longer.

## Required Changes

You must make the following changes to your BSP before it will run on Windows Embedded CE 6.0. These changes include directory restructuring, separation of the kernel from OAL and KITL, deprecated header files and functions, dividing drivers into kernel mode and user mode, changes to the function **MapCallerPtr**, and replacement of the function **OEMEthGetSecs**.

**Important**

Always back up data before making changes to code.

### Working with Restructured Directories

Windows Embedded CE 6.0 removed the folder Public\Common\Oak\Csp and relocated its contents. The new location provides a central location for system-on-chip (SOC) components to help OEMs and silicon vendors (SV) develop and ship them more easily. The following table shows just two of the many folders that have been moved and renamed:

**Table 1: Examples of Relocated CSP and SOC Code**

| Windows CE 5.0 | Windows Embedded CE 6.0 |
|---|---|
| Public\Common\Oak\Csp\ | Platform\Common\Src\SOC\ |
| Platform\Common\Src\ARM\Intel\pxa27x\ | Platform\Common\Src\SOC\pxa27x_ms _v1\ |

If your BSP makes use of any files in Public\Common\Oak\Csp, you must modify your code to look in the new location. For more information about the new SOC directory, see BSP and SOC Directory Layout (http://go.microsoft.com/fwlink/?LinkId=166647).

## Separating the Kernel from OAL and KITL

In Windows CE 5.0, kern.exe incorporated both the Microsoft-supplied kernel and the OEM-supplied OAL. Windows Embedded CE 6.0 separated these two libraries into OAL.exe and kern.dll to make future ports easier, to encourage production-quality OAL (PQOAL) development, and to reduce dependencies among the OAL, the kernel, and the KITL.

In Windows CE 5.0, the kernel was a single binary, but three versions existed for different build types: kern.exe (OAL and kernel), kernkitl.exe (OAL, kernel, and KITL), and kernkitlprof.exe (OAL, kernel, KITL, and profiler). In Windows Embedded CE 6.0, there are three separate binaries: oal.exe, kernel.dll, and the optional kitl.dll. If you are doing production-quality OAL development, we highly recommend separating the OAL and the KITL.

In Windows Embedded CE 6.0, the kernel is built as a DLL, and the oal.exe file contains the startup process. For information about the startup process, see Windows Embedded Base Team Blog - How Does Windows Embedded CE 6.0 Start? (http://go.microsoft.com/fwlink/?LinkId=153796&clcid=0x409)

The following sections contain procedures for porting your BSP from Windows CE 5.0 to Windows Embedded CE 6.0. They must be followed in the order listed here.

### Production-Quality OAL (PQOAL)

Before you start porting your BSP, decide whether to transition the structure of your code to production–quality OAL (PQOAL). Transitioning your code takes a few additional hours to complete but is worth the investment for code that you intend to reuse in other designs. For more information, see Production-Quality OAL (http://go.microsoft.com/fwlink/?LinkId=203925&clcid=0x409).

**To prepare your BSP for migration**

1. Make a copy of the platform that you are porting by making a new folder, such as %_WINCEROOT%\Platform\<*Hardware Platform Name*>-Original. Copy the platform into the new folder and then delete the files in the \<*Hardware Platform Name*>\lib and \<*Hardware Platform Name*>\target folders.
2. Run the blddemo -q command. The build will fail because you have not finished porting the

BSP, but the command will create libraries that you need, like NKldr.lib and OemMain.lib.

3. Create the following new folders under %_WINCEROOT%\Platform\<*Hardware Platform Name*>:
   - Src\Kitl
   - Src\Oal
   - Src\Oal\OalLib
   - Src\Oal\OalExe
4. Edit the Dirs file in %_WINCEROOT%\Platform\<*Hardware Platform Name*>\Src to include the following lines:

```
DIRS= \
  Kitl \
  Oal
```

5. Edit the Src\dirs file to remove the kernel from the list.
6. Create a Dirs file in %_WINCEROOT%\Platform\<*Hardware Platform Name*>\Oal containing the following lines:

```
DIRS= \
  OalLib \
  OalExe \
```

7. Move all your BSP code and build files from Src\Kernel to the Src\Oal folder by using the following steps:

```
move src\kernel\oal -> src\oal\oallib
move src\kernel\kern -> src\oal\oalexe [*]
move src\kernel\kernkitl -> src\kitl [*]
```

**Note**

In Windows Embedded CE 6.0, you do not need to build a profiling versus non-profiling version of the OAL. If your OAL supports profiling, always include it in the final or shipping image. The kernel detects and uses OAL profiling if it is available. For more information, see “Implementing Profiling Support in the OAL” in the Windows Embedded CE 6.0 Help documentation.

8. Delete the following folders:
   - Src\Kernel
   - Src\Kernel\Oal\

- Src\Kernel\Kern
- Src\Kernel\KernKitl
- Src\Kernel\KernKitlProf

## Oal.exe

The oal.exe file is the entry point into the kernel and uses the OEMGLOBAL structure to define functions and variables that the kernel needs. The oemmain.lib library implements the function **OEMInitGlobals** to populate this structure and to register the similar kernel structure, NKGLOBAL.

The oal.exe file links to the following libraries:

- nkldr.lib (home of KernelStart)
- nkstub.lib
- oemmain.lib (if KITL is separately linked)
- oemmain_statickitl.lib (if KITL is statically linked)

**To remove KITL from the OAL**

1. Remove any KITL stubs from the Src\Oal\OalExe folder.
2. Move any KITL-related files from the Src\Oal\OalLib folder to the Src\Kitl folder.
3. Remove KITL-related file names and libraries from the Src\Oal\OalLib Sources file.
4. Remove the following IOCTLs from your OAL IOCTL table (Oal_IOCTL_Tab.h):
    - IOCTL_VBRIDGE_ADD_MAC
    - IOCTL_VBRIDGE_CURRENT_PACKET_FILTER
    - IOCTL_VBRIDGE_GET_ETHERNET_MAC
    - IOCTL_VBRIDGE_GET_RX_PACKET
    - IOCTL_VBRIDGE_GET_RX_PACKET_COMPLETE
    - IOCTL_VBRIDGE_GET_TX_PACKET
    - IOCTL_VBRIDGE_GET_TX_PACKET_COMPLETE
    - IOCTL_VBRIDGE_SHARED_ETHERNET
    - IOCTL_VBRIDGE_WILD_CARD
    - IOCTL_VBRIDGE_WILD_CARD_RESET_BUFFER
    - IOCTL_VBRIDGE_WILD_CARD_VB_INITIALIZED

    These IOCTLs are now handled by Kitl.dll (KitlIoCtl).

### To create and build Oal.exe

1. Start by building oal.lib by running **build –c** in Oal\OalLib.
2. Modify the Sources file in the Oal\OalExe folder.

   To do this, do the following tasks:

   a. Change TARGETNAME from kern to oal.

   b. Remove any KITL stub source file references.

   c. Add the following TARGETLIBS:

   ```
   $(_SYSGENOAKROOT)\lib\$(_CPUDEPPATH)\NkLdr.lib
   $(_SYSGENOAKROOT)\lib\$(_CPUDEPPATH)\OEMMain.lib
   ```

   d. Replace Nk.lib with NkStub.lib.

   e. Remove TARGETLIB entries for anything besides NKStub with NK in the name.

   f. If you are dynamically linking the KITL with the library, add `$(_SYSGENOAKROOT)\lib\$(_CPUDEPPATH)\oemmain.lib \`.

   g. If you are statically linking the KITL with the library, add `$(_SYSGENOAKROOT)\lib\$(_CPUDEPPATH)\oemmain_statickitl.lib`.
3. Replace the call to **OalKitlStart** in **OEMInit** with the following function call:

   ```
   KITLIoctl(IOCTL_KITL_STARTUP, NULL, 0, NULL, 0, NULL);
   ```
4. Replace all calls to **EdbgOutputDebugString** with **KITLOutputDebugString**.
5. Rename **SC_GetTickCount** to **OEMGetTickCount**.
6. In Src\Oal\Oalexe, run **build –c** and note the errors. Fix the errors using the information in the section “To resolve errors”.
7. Run the build command again and verify that Oal.exe builds without any errors.

When you first build your BSP, you will inevitably encounter linking errors not covered by this guide.

### To resolve linking errors

- When you encounter a library linking error, search for the library in Public\Common\Oak or Platform\Common\Src.
- If you run into unresolved functions, search the same locations for their implementations and include the missing libraries. If there is a function that you cannot find, look for an equivalent function in the NKGLOBAL and OEMGLOBAL structures.
- If you have unresolved external symbols, you may need to copy the declarations from the existing location. For example, you may need to declare some global variables in both your OAL and KITL code.

## Kernel.dll

Similar to oal.exe, the kernel implements its own global structure, NKGLOBAL, which is populated with functions and variables that the OAL needs.

Kernel.dll, which is provided by the OS, links to oemstub.lib.

In Windows Embedded CE 6.0, the profiler is no longer in the kernel. You can use the standard profiler if the OAL exports a high-resolution timer. However, if you plan to port your BSP to Windows Embedded Compact 7, you may want to skip this step: in Windows Embedded Compact 7, the kernel defaults to using a 1-millisecond timer if no other is provided.

If you are only porting to Windows Embedded CE 6.0, you can find an example of a 1-millisecond version on MSDN at Windows CE Base Team Blog - Poor Man's Monte Carlo (http://go.microsoft.com/fwlink/?LinkId=153797&clcid=0x409). To export the timer, you must implement functions of type **PFN_QueryPerfCounter** and **PFN_QueryPerfFreq**. For an example implementation, see %_WINCEROOT%\Platform\BSPTemplate\Src\Oal\Oallib\timer.c.

## Kitl.dll

Windows Embedded CE 6.0 uses the Kernel Independent Transport Layer (KITL) transport for communication between the development computer and the target device over any hardware for which the OEM supplies an appropriate transport.

The KITL Ethernet transport on the device reuses much of the Ethernet debug (EDBG) code. To map to the new KITL names, the header file %_WINCEROOT%\Public\Common\Oak\Inc\Halether.h has been included. It maps the KITL functions to the EDBG functions.

Kitl.dll links with the following libraries:

- kitlcore.lib (replacement for kitl.lib and location of KitlDllMain)
- nkstub.lib (if the KITL uses NKGLOBAL functions)
- oemstub.lib (if the KITL uses OEMGLOBAL functions)

Moving the KITL source code from the OAL into Kitl.dll is typically a fairly straightforward process. Kitl.dll must export the following new functions:

- **OEMKitlStartup**

  This function is called from the kernel when KITLIoctl(IOCTL_KITL_STARTUP, NULL, 0, NULL, 0, NULL) is invoked by the OAL.

  This function is equivalent to the OALKitlStart function, but it has been renamed to follow the convention for public OEM functions. The Windows Embedded CE 6.0 kernel requires that you use the new function name OEMKitlStartup.

- **OEMKitlIoctl**

  This function handles all KITL-related IOCTLs when KITL is removed from the OAL. In most cases, you can use the common version of this function, which is implemented in Oal_Kitl.lib.

- **OEMKitlInit**

  This function is called by the KitlInit function to initialize the KITL device and the KITLTRANSPORT structure.

**To create and build Kitl.dll**

1. Copy the makefile from the Src\Oal\OalLib folder to the Src\Kitl folder.
2. Create the Sources file based on the KernKitl Sources file that you copied earlier.

   To do this, use the following commands:

   ```
   TARGETNAME=KITL

   TARGETTYPE=DYNLINK

   DLLENTRY=KitlDllMain

   DEFFILE=
   ```

   Delete entries for EXEENTRY and EXEBASE.

   Delete nk.lib and oal_log.lib, if present.

   Except for oal_kitl.lib (or oal_kitl_x86.lib on x86-based platforms), delete all OAL libraries.

   Rename kitl.lib to kitlcore.lib.

   Then, add the following TARGETLIBS: KitlCore.lib, OEMStub.lib, and NkStub.lib.

   ```
   $(_SYSGENOAKROOT)\lib\$(_CPUDEPPATH)\nkstub.lib \.

   $(_SYSGENOAKROOT)\lib\$(_CPUDEPPATH)\oemstub.lib \.

   $(_PLATCOMMONLIB)\$(_CPUDEPPATH)\kitl_log.lib \.
   ```

   These libraries are in addition to any other libraries required by your KITL implementation. If you are porting to Compact 7, delete vbridge.lib.

   If you are porting from Windows CE 5.0 to Windows Embedded CE 6.0 and not continuing to Compact 7, add:

   ```
   $(_SYSGENOAKLIB)\$(_CPUDEPPATH)\vbridge.lib \.
   ```

   Except for ddk_io.lib and vbridge.lib, delete all.lib files that don’t have KITL in their name.
3. Rename OALKitlStart to OEMKitlStartup.
4. Initialize two KITL function pointers in OEMKitlInit.

   To do this, use the following code example:

   ```
   pKitl->pfnPowerOn = OALKitlPowerOn;

   pKitl->pfnPowerOff = OALKitlPowerOff;
   ```

This step is required only if your OAL is implementing these functions.

5. In the Kitl directory, run **build –c** and troubleshoot errors following the same procedure as you did when building Oal.exe.

## Additional Steps for CEPC

A few additional steps are required to separate the kernel, OAL, and KITL for a Windows Embedded Compact PC-based platform (CEPC). The files oal_cache_x86.lib and oal_kitl_x86.lib are specific to x86-based hardware platforms. The file globals.c is specific to CEPC.

**To divide the CEPC kernel into three pieces**

1. Follow the preceding steps to prepare your BSP for migration and to remove KITL from the OAL.
2. In addition to the preceding steps to create and build Oal.exe, edit the \Oal\Oalexe\Sources file as follows:
   a. Add the following line:

      ```
      $(_PLATFORMCOMMON)\lib\$(_CPUDEPPATH)\oal_cache_x86.lib \.
      ```

   b. Remove oal_cache.lib from the Sources file.
3. In addition to the preceding steps to create and build Kitl.dll:
   a. Edit \Kitl\Sources to add globals.c to Sourceslist.
   b. Copy `\Oal\Oallib\globals.c to \Kitl\`.
   c. Edit globals.c as follows:

      Remove all `#include` statements except the following:

      ```
      #include <windows.h>.
      ```

      ```
      #include <x86kitl.h>.
      ```

      If these `#include` statements do not exist in globals.c, add them.

      Delete all global variable declarations except `g_oalIoCtlPlatformType` and `g_ucDlftKitlAdaptorType`.
   d. Do not rename the OALKitlStart function to OEMKitlStartup.

## Replacing Deprecated Header Files

The functionality that was in oal_nkxp.h and oal_profiler.h is now in two other header files, nkexport.h and bcoemglobal.h. If your build fails due to the inclusion of either of these files, do the following:

▶ **To replace the deprecated header files**

1. Delete the include statements for oal_nkxp.h and oal_profiler.h.
2. Include %_WINCEROOT%\Platform\Common\Src\Common\oal.h, which includes nkexport.h. Nkexport.h contains the functionality that we are concerned with here.
3. Try to build your project.
4. If your build still fails, look for the missing functions in %_WINCEROOT%\Public\Common\Oak\Inc\bcoemglobal.h and include this header file if needed.

## Replacing Deprecated String Functions

In Windows Embedded CE 6.0, a number of string function prototypes were deprecated. If your build breaks on one of these missing functions, search strmisc.h or other public headers for a suitable replacement function.

## Choosing Kernel Mode or User Mode

Kernel-mode drivers are loaded by device.dll inside the kernel and have full access to kernel APIs without having to go through the user-mode driver reflector. Kernel-mode drivers are faster than user-mode drivers because they do not have to switch user processes, so they are well suited to read or write to hardware registers. However, instability in the driver, such as incorrect or invalid memory use, can cause the kernel to stop functioning.

User-mode drivers are loaded via udevice.exe. They improve system stability by isolating driver failures to udevice.exe. The trade-off is performance. For user-mode drivers, the **VirtualCopy** function only works for physical addresses that you declare in the registry. You must use the user-mode driver reflector to forward I/O requests from the device manager. For more information on the reflector and how it works, see User Mode Driver Reflector. (http://go.microsoft.com/fwlink/?LinkId=153880&clcid=0x409)For information on porting a driver to Windows Embedded CE 6.0, see the video Microsoft Showcase: Porting Drivers to Windows Embedded CE 6.0 (http://go.microsoft.com/fwlink/?LinkId=157392&clcid=0x409).

### Kernel Mode

To make your driver run in kernel mode, in the **FLAGS** registry key for the driver, clear the flag `DEVFLAGS_LOAD_AS_USERPROC (0x10)` and then add the K flag to the binary image builder (.bib) file entry, as shown in the following example:

```
gpio.dll        $(_FLATRELEASEDIR)\gpio.dll                NK SHK
```

### User Mode

To make your driver run in user mode, in the **FLAGS** registry key for the driver, set the flag `DEVFLAGS_LOAD_AS_USERPROC (0x10)` and delete the K flag from the .bib file entry, as shown in the following example:

```
decodeCombo.dll $(_FLATRELEASEDIR)\decodeCombo.dll        NK    SH
```

## Replacing MapCallerPtr and OEMEthGetSecs

### MapCallerPtr

Prior to Windows Embedded CE 6.0, you used the function **MapCallerPtr** for pointers embedded in data structures that were passed into drivers. For pointers passed as function arguments, it was not necessary to use **MapCallerPtr**, because Device Manager took care of mapping pointers passed in as parameters.

In Windows Embedded CE 6.0 and Windows Embedded Compact 7, you must marshal an embedded pointer with the **CeOpenCallerBuffer** and **CeCloseCallerBuffer** functions. As in prior releases, Device Manager marshals pointers that are passed in as parameters, so you need to marshal only embedded pointers. The **CeAllocAsynchronousBuffer** function is for drivers that need asynchronous access to the data, for example, in situations where the driver needs to retry sending data while not blocking the caller indefinitely. For more information about marshaling embedded pointers, see Marshalling for Windows Embedded CE 6.0 Driver Migration (http://go.microsoft.com/fwlink/?LinkId=153882&clcid=0x409).

### OEMEthGetSecs

In Windows Embedded CE 6.0 and Windows Embedded Compact 7, the platform-specific function **OEMEthGetSecs** has been replaced with the KITL implementation of **OEMKITLGetSecs**. If your BSP relies on the header file e_to_k.h to map the two functions, then you must include halether.h in your header file. If your BSP implements its own **OEMEthGetSecs**, then you do not need to make any changes.

## Design-Dependent Changes

If your BSP contains support for the **bProfileTimerRunning** variable, then you must make the following changes to your BSP before it can run on Windows Embedded CE 6.0.

### Updating the bProfileTimerRunning Variable

Windows Embedded CE 6.0 and Windows Embedded Compact 7 do not contain the **bProfileTimerRunning** variable. If your BSP uses the **bProfileTimerRunning** variable, you must declare **bProfileTimerRunning** as a global variable in the OAL. Set this variable in the **OEMProfileTimerEnable** and **OEMProfileTimerDisable** functions to an appropriate value, as shown in the following example:

```
BOOL bProfileTimerRunning = FALSE


OEMProfileTimerEnable ()

{

  // The rest of the code for this function.


  bProfileTimerRunning = TRUE;

}


OEMProfileTimerDisable ()

{

  bProfileTimerRunning = FALSE;


  // The rest of the code for this function.

}
```

## Cleanup

You may need to make the following changes to your BSP before it can run on Windows Embedded CE 6.0. These cleanup changes include the removal of page pool flags and deprecation of several functions and some IOCTLs.

### Removing Page Pool Flags

In Windows Embedded CE 6.0 and Windows Embedded Compact 7, you specify the page pool by adding and customizing the following ***fix-up*** variables within the config.bib file for the platform:

```
kernel.dll:LoaderPoolTarget            00000000 00300000 FIXUPVAR

kernel.dll:LoaderPoolMaximum           00000000 00800000 FIXUPVAR

kernel.dll:FilePoolTarget              00000000 00100000 FIXUPVAR
kernel.dll:FilePoolMaximum              00000000 00A00000 FIXUPVAR
```

Next, remove all references to IMGPAGINGPOOL and PAGINGPOOLSIZE from your BSP.

### Removing Deprecated Functions

The following functions are deprecated because of the new memory model in Windows Embedded CE 6.0. You must remove these functions from your code base. Stub functions are present, so existing code can continue to call these functions without generating compiler or linker errors.

- **CeZeroPointer**
- **GetCurrentPermissions**
- **MapPtrToProcess**   Replaced with **CeOpenCallerBuffer** and **CeCloseCallerBuffer**.
- **MapCallerPtr**   Replaced with **CeOpenCallerBuffer**, **CeCloseCallerBuffer**, **CeAllocAsynchronousBuffer**, and **CeFreeAsynchronousBuffer**.
- **SetKMode**
- **SetProcPermissions**
- **UnMapPtr**

For more information, see Unsupported Kernel APIs (http://go.microsoft.com/fwlink/?LinkId=153885&clcid=0x409).

### Removing Deprecated IOCTLs

The IOCTL_HAL_GETREGSECUREKEYS and DEVFLAGS_TRUSTEDCALLERONLY have been deprecated due to changes in Windows Embedded CE 6.0.

#### IOCTL_HAL_GETREGSECUREKEYS

The Windows Embedded CE 6.0 security model eliminates the need for this IOCTL. You can remove it from your code.

#### DEVFLAGS_TRUSTEDCALLERONLY

The Windows Embedded CE 6.0 security model eliminates the need for this IOCTL. The Device Manager no longer checks the DEVFLAGS_TRUSTEDCALLERONLY flag to restrict access to the drivers that use this flag.

# Migrating from CE 6.0 to Compact 7

Now that you have enhanced the BSP to support the Windows Embedded CE 6.0 kernel, 90 percent of the porting is complete. The final portion does not require source code changes.

## Required Changes

You must make the following changes to your BSP for it to run on Windows Embedded Compact 7. These changes include updates to some paths in the Sources files, moving prebuilt binaries, updating the path of RNE_MDD, removing Vbridge.lib, a change in GUID initialization, removal of the PFN_EDBG_XXXX typedefs, and updating function prototypes.

**Important**

Always back up data before making changes to code.

### Updating Sources Paths

You must update several paths in the Sources files to ensure that Windows Embedded CE 6.0 build scripts continue to work in Windows Embedded Compact 7. For change instructions, refer to “Updating Paths in Sources Files” in the article titled Build Porting Guide for Windows Embedded Compact 7 (http://go.microsoft.com/fwlink/?LinkId=205658).

#### Copying Prebuild Binaries

If your build includes prebuilt binary files from BSP vendors, you must copy these files to the appropriate directories to ensure that builds from Windows Embedded CE 6.0 continue to work in Windows Embedded Compact 7. For instructions, refer to “Copying Prebuilt Libraries” in the article titled Build Porting Guide for Windows Embedded Compact 7 (http://go.microsoft.com/fwlink/?LinkId=205658).

#### Working with the Relocation of RNE_MDD.lib

In Windows Embedded CE 6.0, RNE_MDD.lib was moved from Public\Common\Oak\Drivers\Ethdbg\Rne_mdd to Platform\Common\Src\Common\Ethdrv\Rne_mdd. The result of this change is that the output binary is written to the environment path of **_PLATCOMMONLIB** instead of **_SYSGENOAKLIB**. To update the path, edit all files that include rne_mdd.lib and change the system variable **_SYSGENOAKLIB** to **_PLATCOMMONLIB** as follows:

```
$(_PLATCOMMONLIB)\$(_CPUDEPPATH)\rne_mdd.lib
```

#### Working with the Relocation of Vbridge

For Windows Embedded Compact 7, Vbridge functionality is in oal_kitl.lib instead of in a stand-alone library. Include oal_kitl.lib or oal_kitl_pci.lib in the TARGETLIBS list when building KITL. Remove Vbridge.lib from the TARGETLIBS list.

The new location for this functionality is %_WINCEROOT%\Platform\Common\Src\Common\Kitl\Vbridge.

### Changing GUID Initialization

In Windows Embedded CE 6.0, the **DEFINE_GUID** macro creates either a GUID declaration or a GUID definition depending on the placement of the macro relative to the inclusion of initguid.h. In Windows Embedded Compact 7, the **DEFINE_GUID** macro always defines a GUID. Therefore, code carried forward from previous versions and compiled in Windows Embedded Compact 7 may contain multiple conflicting GUID definitions. To fix this problem, delete redundant definitions or move them inside include guards so GUIDs are defined only once.

### Removing PFN_EDBG_XXXX

The following function typedefs have been removed from the code base in Windows Embedded Compact 7. If you encounter build breaks, remove these from your BSP.

```
PFN_EDBG_GET_PENDING_INTS pfnEDbgGetPendingInts;

PFN_EDBG_READ_EEPROM pfnEDbgReadEEPROM;

PFN_EDBG_WRITE_EEPROM pfnEDbgWriteEEPROM;
```

## Design-Dependent Changes

If your BSP includes the features in this section, then you must make the following changes to your BSP for it to run on Windows Embedded Compact 7. These changes include modifying the kernel independent transport layer (KITL) driver or the **OEMAddressTable**, removing deprecated drivers and libraries and the **ENABLE_OAL_ILTIMING** variable, including header files for installable interrupt service routines (IISR), changing the name of the battery driver library model device driver (MDD), removing language DLLs, moving the **SystemHive** registry value and the associated system hive file to a specified path, a change to linker overrides, and changes to the profiler function prototypes.

### Removing PCMCIA/PC Card

PCMCIA, also known as PC Card, is no longer supported in Windows Embedded Compact 7. Although previous versions of Windows Embedded Compact supported this bus type, Windows Embedded Compact 7 supports alternative and more current buses like USB, which support the same set of peripherals. The source code for these features has been removed from the OS. As a result, you may experience some build difficulty if you are porting a platform that had previously supported these buses. Removing references to the removed components should remedy this build difficulty.

### Updating the ATAPI Driver Legacy Registry Setting

The ATAPI driver in Windows Embedded Compact 7 replaces the registry subkey `HKLM\Drivers\BuiltIn\PCI\Template\<Controller Key Name>\Legacy` with the subkey `LegacyIRQ`. You must replace any occurrence of `Legacy` in the registry with `LegacyIRQ`.

If `LegacyIRQ` is equal to `IRQ` (`IRQ 14` for CEPC), the value of the secondary channel IRQ will be the Primary IRQ plus 1 (`IRQ 15` for PC). If `LegacyIRQ` is absent or equal to -1 (0xffffffff), the driver will assume it is not using the legacy IRQ and both channels will use the same IRQ (IRQ sharing).

## Adding ATAPI Driver Registry Settings for IRQ Sharing

The ATAPI driver in Windows Embedded Compact 7 uses the exported functions **NativeConfig**, **CreateNativePCIHD**, and **CreateNativePCIHDCD** to support ATAPI controllers with IRQ sharing (such as SATA). To support IRQ sharing, the Platform.reg file in %_WINCEROOT%\Platform\<*your platform*>\files must contain the following settings:

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\PCI\Template\GenericIDE]

    "ConfigEntry"="NativeConfig"    ; PCI configuration entry point

    "IsrDll"="giisr.dll"            ; bus-agnostic; installable ISR

    "IsrHandler"="ISRHandler"       ; bus-agnostic; installable ISR

    "SpawnFunction"="CreateNativePCIHDCD" or "CreateNativePCIHD" ; depends on HD
only or HD + CD/DVD

    "LegacyIRQ"=dword:fffffff      ; explicitly sets native mode; not necessary for
most cases
```

In most situations, you do not need to set the `LegacyIRQ` value to support IRQ sharing. The registry contains the default setting of legacy mode, in either the %_WINCEROOT%\Public\Common\Oak\Drivers\Block\ATAPI\*.reg files or the Platform.reg file. The ATAPI driver automatically detects the controller mode from the `ProgIf` value set by PCIbus.dll, and overwrites the default registry value to native mode if needed. The default setting must be legacy mode because the ATAPI driver can overwrite the `LegacyIRQ` value to support native mode, but it cannot change the value from native to legacy mode.

If the `ProgIf` value does not report the controller mode (using the bit definitions described in the **NativeConfig** function definition in %_WINCEROOT%\Public\Common\Oak\Drivers\Block\atapi\pcicfg.cpp), you must set the `LegacyIRQ` value explicitly, as follows:

- To support native mode, set the LegacyIRQ value as in the preceding example.
- To support legacy mode, set the LegacyIRQ value as follows:

```
        "LegacyIRQ"=dword:e      ; explicitly sets legacy mode
```

## Modifying OEMAddressTable and KITL Drivers

The Windows Embedded Compact 7 kernel supports a new option for the **OEMAddressTable** that disables the uncached static mapping (from 0xA0000000 to 0xBFFFFFFF) and makes available more kernel virtual memory space. You can enable this address table option for any ARM-based or x86-based platform; the Windows Embedded Compact PC-based platform (CEPC) uses the new **OEMAddressTable** model by default. For any platforms that use a KITL driver that predates Windows Embedded Compact 7, there are two possible solutions:

1. (For any ARM-based or x86-based platform) Modify your KITL driver so it uses the direct memory access (DMA) buffer passed from KITL, assuming it is already uncached; in other words, without doing a cached-address to uncached-address conversion. The KITL driver must use the **OALVAtoPA** function to obtain the physical address of the DMA buffer. It must not assume static mapping of the kernel virtual memory, for either cached or uncached memory. We recommend this solution because any KITL driver modified in this way will work in Compact 7, whether or not your platform uses the new **OEMAddressTable** option.
2. (For CEPC only) Disable the new **OEMAddressTable** option by modifying the %_WINCEROOT%\Platform\Cepc\Src\Oal\Oalexe\Sources file so that it links to oal_startup_x86.lib instead of oal_startup_x86_newtable.lib. This solution is a workaround; if your platform is modified later to use the new **OEMAddressTable** option, you must implement the first solution.

If your KITL driver does not assume the existence of uncached static mapping, and it always uses OALxxxToyyy functions for address translation, your driver will work with either **OEMAddressTable** option. It works because the **OALCAtoUA** and **OALPAtoUA** functions in Windows Embedded Compact 7 return the same address passed from KITL to ensure backward compatibility.

## Updating HalAllocateCommonBuffer

The **HalAllocateCommonBuffer** function now supports the *CacheEnabled* parameter, which was ignored in Windows Embedded CE 6.0. To ensure that your BSP and drivers are compatible with Windows Embedded Compact 7, edit any calls to **HalAllocateCommonBuffer** so that *CacheEnabled* is set to false.

## Replacing Deprecated Drivers and Libraries

The following libraries were removed from the source tree. If you use any of these libraries in your Sources files, you must update the Sources files to use new drivers.

**Ethernet Drivers**

- 3C90X
- AM79C970 (replaced with AM79C973)
- AM79C970 (replaced with AM79C973)
- DP83815
- NET2890

**Bootloader Libraries**

- Edbgfrmt
- Rndsmini

The following libraries are now located only in Platform/Common/Src/Common/Ethdbg. If your Sources files include any of these libraries, you must update the files so that the linker can find the libraries, as shown in the following example:

```
$(_PLATCOMMONLIB)\$(_CPUDEPPATH)\<name>.lib
```

**Note**

In Windows Embedded Compact 7, the library name for NE2000 is oal_ethdrv_ne2000.lib; in previous versions it was ne2kdbg.lib.

**Ethernet Drivers**

- AM79C973
- CS8900A
- DEC21140
- RT8139
- NE2000

**Bootloader Libraries**

- blcommon
- Eboot
- Fallite
- Kitleth

## Removing ENABLE_OAL_ILTIMING

Instead of measuring interrupt latency by using the environment variable ENABLE_OAL_ILTIMING and rebuilding the kernel, Windows Embedded CE 6.0 implements an OEM adaption layer (OAL) interrupt latency IOCTL. Link the OAL to either OAL_ILT.lib and instrument your timer interrupt service routine (ISR) or link to OAL_ilt_stub.lib.

## Adding ISR Header File

Prior to Windows Embedded Compact 7, common Windows Embedded Compact device driver development kit (CEDDK) header files have included giisr.h, which provided installable interrupt service routines (IISR) functionality. In Windows Embedded Compact 7, if you need both CEDDK and IISR functionality, you must include two header files: one for CEDDK functionality and one for IISR.

## Linking to Battery Driver LIB and DEF Files

The battery driver library MDD has been updated and the name has been changed. If you link to battdrvr_lib.lib in your Sources file, change the name to batterdrvr_mdd.lib, as shown in the following example:

Windows Embedded CE 6.0

```
$(_SYSGENOAKROOT)\lib\$(_CPUINDPATH)\battdrvr_lib.lib \
```

Windows Embedded Compact 7

```
$(_SYSGENOAKROOT)\lib\$(_CPUINDPATH)\battdrvr_lib.lib \
```

The battdrvr.def file has been relocated from Platform\Common\Oak\Drivers\Battdrvr to Platform\Common\Oak\Inc. You no longer need to use a relative path to link to it. Here is what the change looks like:

Windows Embedded CE 6.0

```
DEFFILE=$(_PUBLICROOT)\common\oak\lib\$(_CPUINDPATH)\battdrvr.def
```

Windows Embedded Compact 7

```
DEFFILE=$(_SYSGENOAKROOT)\inc\battdrvr.def
```

## Removing Language DLLs

The following language files have been removed from the Windows Embedded Compact 7 image.

```
KbdNopUs.dll

KbdNopJpn1.dll

KbdNopJpn2.dll

KbdNopKor.dll
```

They have been replaced by kbdNop.dll and kbdUs.dll. Both are common drivers, meaning that they are located in %_WINCEROOT%\Public\Common\Oak\Drivers and are included by the Common.reg and Common.bib files. Define SYSGEN_KBD_US=1 to build and include these language libraries.

## Enforcing the Location of System Hive File

In prior versions of Windows Embedded Compact, the location of the system hive file has been left to BSP architects. Starting with Windows Embedded Compact 7, the system hive file must be in the path specified by Common.bib as shown in the following code example.

```
[HKEY_LOCAL_MACHINE\init\BootVars]
```

```
"SystemHive"="Windows\\Registry\\system.hv"
```

If you override the default location, you may encounter the following error when booting:

```
FSDMGR!ProcessRebootFlags: failed cleaning volume!
```

### Changing Linker Overrides

In Windows Embedded CE 6.0 and previous versions, it was possible to override an object (.obj) file by creating a second .obj file with the same filename. This technique was used in some places in the OS source code for these versions. This technique does not work in Windows Embedded Compact 7, because the build uses full file name paths. For information about overriding .obj files in the build, refer to “Changing Linker Overrides” in the article titled Build Porting Guide for Windows Embedded Compact 7 (http://go.microsoft.com/fwlink/?LinkId=205658).

### Changing Profiler Function Prototypes

In previous versions of Windows Embedded Compact that used x86, SH, or MIPS processors, **OALProfileIntrHandler** was defined as `UINT32 OALProfileIntrHandler(VOID)`. Windows Embedded Compact 7 defines this as `UINT32 OALProfileIntrHandler(UINT32 ra)`.

This function does not have a prototype in any Microsoft code because only assembly code calls it. Search your code base and ensure that it accepts ra (return address).

In your **OALProfileIntrHandler** function, call **ProfilerHit(ra)** instead of **ProfilerHit(GetEPC)**.

## Cleanup

You may need to make the following changes to your BSP for it to run on Windows Embedded Compact 7. These maintenance and cleanup changes include removing the image flag NOMIPS16CODE and the dimension macros.

### Replacing Deprecated Flags and Macros

#### NOMIPS16CODE

The image flag NOMIPS16CODE from Windows Embedded CE 6.0 is replaced by the NOIMPLICITIMPORT flag in Windows Embedded Compact 7. For information about this new flag, refer to “NOMIPS16CODE” in the article titled Build Porting Guide for Windows Embedded Compact 7 (http://go.microsoft.com/fwlink/?LinkId=205658).

### Dim (Dimension) Macros

The dim macro, **#define dim(x) (sizeof(x)/sizeof(x[0]))**, has been deprecated. This code is still present to prevent build breaks, but it now generates warnings. Replace the macro with **_countof**.

The following similar macros are also deprecated and should be replaced with **_countof**:

- ARRAYSIZE
- ARRAY_SIZE
- ARRAYSIZEOF
- ARRSIZE
- SIZEOF_ARRAY
- ARRAY_LENGTH
- NUM_ELEMENTS
- NELEMS
- NUM
- NUMBER_OF_ARRAY
- TABLE_COUNT
- COUNTOF
  ItemCount
- Dim
- DIMOF
- CCHSIZEOF

## Updating Multimedia

If your BSP includes multimedia support, you must make the following changes. These changes include access to the frame buffer and new locations for camera header files.

### Display: Video Memory

In Windows Embedded CE 6.0 and previous versions, you could get a pointer to the frame buffer and then update the display by writing directly to the frame buffer. However, hardware-accelerated graphics systems do not guarantee the format or location of data stored in the frame buffer. Therefore, direct pointers to the frame buffer are not supported in Windows Embedded Compact 7. Other than the display driver, functions must not directly access the frame buffer. You must rewrite applications or other drivers that directly access the frame buffer so that your applications use supported APIs.

### Camera

The following camera header files have been renamed and moved in Windows Embedded Compact 7.

**Table 3: Camera header file names**

| Windows Embedded CE 6.0 Names | Compact 7 Names |
|---|---|
| Public\Common\Oak\Driver\Capture\Camera\Layered\Inc\dgbSettings.h | Public\Common\Ddk\Inc\CameraDebug.h |
| Public\Common\Oak\Driver\Capture\Camera\Layered\Inc\PinDriver.h | Public\Common\Ddk\Inc\CameraPinDriver.h |

Scan through camera source files looking for these Windows Embedded CE 6.0 camera files and replace the include names with the new Windows Embedded Compact 7 names.

# Conclusion

You can port a BSP to Windows Embedded Compact in two stages: porting from Windows CE 5.0 to Windows Embedded CE 6.0, and then from Windows Embedded CE 6.0 to Windows Embedded Compact 7. The biggest changes occur when porting from Windows CE 5.0 to Windows Embedded CE 6.0: separating the kernel from OAL and KITL, and dividing the drivers into kernel mode and user mode. You make fewer changes when porting from Windows Embedded CE 6.0 to Windows Embedded Compact 7, and porting can be accomplished in hours. However, a full port of your BSP from Windows CE 5.0 to Windows Embedded Compact 7 may take weeks or months, depending on the complexity of your BSP. The focus of this document is to help you migrate a BSP as easily and painlessly as possible.

# Additional Resources

Windows Embedded website (http://go.microsoft.com/fwlink/?LinkId=183524)