Lab 2-1: Clone a BSP

Create, Build, and Run a New OS Design

Learning Objectives

- Create an OS Design using Visual Studio
- Identify the catalog features included in the design
- Extend the standard design by adding catalog items
- Build configuration for the run-time image and build a run-time image
- Run the OS image on the target device

Prerequisites

• Knowledge of the vocabulary used in OS design, Visual Studio, and the Platform Builder Plug-in for Visual Studio

Estimated time to complete this lab: 45 minutes

Lab Setup

To complete this lab, you must have:

- A development workstation running Windows XP
- Visual Studio 2005 (Version 8) with Platform Builder plug-in
- CE 6.0 with 2006 Roll up and Service Pack 1
- CE 2007 Updates upto month 9th month {QFEs}
- CE R2
- CE 2007 Updates 11th and 12th month {QFES}
- CE 2008 Updates
- EVM Reference platform
- TI_EVM_3530-Training BSP

Exercise 1 Clone BSP

In this exercise, you will use the Clone BSP tool in Visual Studio 2005 to create a copy of the existing EVM Board Support Package (BSP). We can modify this copy instead of modifying the original that was delivered as a part of the Windows Embedded CE 6.0 tools.

* Clone the EVM BSP

1. Launch Microsoft Visual Studio 2005.

Note If this is the first time Visual Studio is launched after installation the Choose Default Environment Setting dialog will be displayed. For the purposed of this course select Platform Builder Development Settings and select Start Visual Studio.

2. Select the **Tools** | **Platform Builder for CE6.0** | **Clone BSP** from the menu in Visual Studio to bring up the Clone BSP dialog box.

С	Clone Board Support Package					
	OS build tree (WINCEROOT):				
	C:\WINCE600	Browse				
	Source BSP:					
	TI_EVM_3530-Training: AF	RMV4I 💌				
	New BSP information					
	<u>N</u> ame:	EVMBSP				
	Description:	BSP Clone for EVM				
	Platform directory:	EVMBSP				
	⊻endor:	GeneriCo				
	V <u>e</u> rsion:	1.0				
Open new BSP catalog file in Catalog Editor						
		Clone Close				

- 3. In the Clone BSP dialog select **TI_EVM_3530-Training: ARMV4I** from the Source Board Support Package: drop down box.
- 4. Type **EVMBSP** in the Name field in the New Board Support Package Info area.

- 5. Type a description [**BSP Clone for EVM**] for your new BSP in the Description field.
- 6. Type **EVMBSP** in the Platform Directory field.
- 7. Type GeneriCo in the Vendor field.
- 8. Type **1.0** in the Version field.
- 9. Click the **Clone** Button. The Clone BSP tool will create a new Board Support Package based on the EVM Board Support Package.
- 10. Acknowledge the Clone BSP success message by selecting OK.

The EVM Board Support Package has now been cloned into a new Board Support Package called **EVMBSP**. This EVMBSP Board Support Package will be used in the remaining labs.

Exercise 2 Create, build and run the OS design

In this exercise you will create an OS design, and then customize that design by adding components from the catalog and build the result. You will run the OS Design on the EVM reference platform.

This OS Design will be used in other labs and will be a suitable platform for running a variety of Windows CE applications.

You will learn how to:

- Create an OS Design
- Set up the build configuration for your OS run-time image
- Build an OS run-time image
- Run the OS Design on the EVM reference platform.

* Create an OS design

1. Select File | New | Project... from the Visual Studio menu.

New Project		? 🛛
Project types:	<u>T</u> emplates:	
Visual C++	Visual Studio installed templates	
	oS Design	
General	M. Tamalahan	
Win32	Search Online Templates	
Other Languages Other Project Types		
Platform Builder for CE 6.0		
A project for creating a Windo	ws Embedded CE 6.0 operating system	
Name: EVMOSDe	sign	
Location: E:\WINCE	600\OSDesigns	Browse
Solution Name: EVMOSDe	sign Create directory	for solution
		OK Cancel

2. Select the **Platform Builder for CE 6.0** project type in the New Project dialog.

- 3. Select **OS Design** under Visual Studio installed templates.
- 4. Type **EVMOSDesign** in the Name field. The solution name will default to EVMOSDesign as well.
- 5. Click **OK**. Visual Studio will launch the Windows Embedded CE 6.0 OS Design Wizard.
- 6. Click Next.

Windows Embedded CE 6.0 OS Design Wizard					
Board Support Packages					
Available BSPs:					
Aruba Board: ARMV4I CEPC: x86	A BSP contains a set of device drivers that are added to your OS design.				
Device Emulator: ARMV4I EVMBSP: ARMV4I	Select one or more BSPs for your OS design.				
H4Sample OMAP2420: ARMV4I ICOP_eBox2300_60B: X86 MainstoneIII PXA27X: ARMV4I HP Compaq t5530 Thin Client:X86 TI_EVM_3530:ARMV4I Voice over IP PXA270: ARMV4I	BSP CLone for EVM				
	Note: Only BSPs supported by installed CPUs are displayed in the list.				
< <u>Previous</u> <u>N</u> ext > <u>Finish</u> Cancel					

7. In the list of available BSPs, select EVMBSP: ARMV4I and click Next.

Windows Embedded CE 6.0 OS Design Wizard	? 🗙
Design Templates	
Available design templates: Consumer Media Device Custom Device Industrial Device PDA Device Small Footprint Device Thin Client	A design template is a set of predefined catalog items. Choose the design template that is most closely aligned with the purpose of your target device. Provides the starting point for a range of personal digital assistants (PDAs) or mobile devices with a clamshell-and-keyboard design.
< <u>P</u> revious	<u>N</u> ext > <u>E</u> inish Cancel

8. From the list of available design templates, select **PDA Device** and click **Next.**

Windows Embedded CE 6.0 OS Design Wizard					
N	Design Template Variants				
<u>V</u> ariants: Mobile Ha Enterpris	ndheld e Web Pad	Mobile Handheld			
	< Previous	<u>N</u> ext > <u>E</u> inish Cancel			

9. From the list of available design variants, select **Mobile Handheld** and click **Next**. The **Applications & Media** configuration window will appear.

Windows Embedded CE 6.0 OS Design Wizard 🛛 🔹 🛛 😨 🔀						
Applications Media						
.NET Compact Framework 2.0 File Systems and Data Store Windows Embedded CE Error Reporting ActiveSync Internet Browser Quarter VGA Resources - Portrait Mode Windows Media Audio/MP3 Windows Messenger WordPad	Support for applications and services designed for the .NET V2.0 Compact Framework.					
< Previous	Next > Einish Cancel					

10. Deselect .NET Compact Framework 2.0 and Quarter VGA Resources – Portrait Mode and click Next. The Networking & Communications configuration window will appear.

Windows Embedded CE 6.0 OS Design Wizard	
Networking Communications	
TCP/IPv6 Support Wide Area Network (WAN) Local Area Network (LAN) Personal Area Network (PAN) Security	The Internet standard protocol, version 6.
< Previous	<u>N</u> ext > <u>F</u> inish Cancel

- 11. Deselect TCP/IPv6 Support.
- 12. Deselect **Personal Area Network (PAN)**. This will deselect Bluetooth and IrDA.
- 13. Click **Next**, and then **Finish** to complete the Windows Embedded CE 6.0 Design Wizard.

Windows	Embedded CE 6.0 OS Design Wizard	?×
N	OS Design Project Wizard Complete	
You have	e completed the wizard. Press Finish to create your OS Design project.	
	< <u>P</u> revious <u>N</u> ext > <u>Finish</u> Can	cel

Note The wizard creates the initial configuration for your OS Design. We will have the opportunity to make further changes to the OS Design after completing the wizard.



14. Click Acknowledge on the Catalog Item Notification dialog.

On completion, Visual Studio will display your OS design project. The Solution Explorer tab should be active and show your new EVMOSDesign project in your EVMOSDesign Solution.



* Inspect the OS Catalog

- 1. Click on the Catalog Items View tab to display the Catalog.
- 2. Click on the **Filter** drop down box in the upper left hand corner of the Catalog Items View. Observe the different filtering options. The filter controls the items that are displayed in the catalog. Ensure that **All Catalog Items in Catalog** is selected.
- 3. Observe the selection boxes and icons in the catalog by expanding the **nodes**. Selection boxes with a green check mark indicate an item that was specifically selected as a part of the OS design. Selection boxes with a green square indicate an item that was brought in to the OS design as a dependency. Selection boxes that are not marked indicate items that are not included in the OS design but are available to be added.
- 4. Locate a catalog item with a green square in its checkbox.
- 5. Right click on the catalog item and choose **Reasons for Inclusion of Item**. The **Remove Dependent Catalog Item** dialog box displays the catalog items you selected that caused this catalog item to automatically be included in the OS design.
- 6. Close the Remove Dependent Catalog Item dialog box.

- 7. Expand the Core OS | CEBASE | Applications End User | Active Sync node in the catalog.
- 8. Right click on either of the ActiveSync system cpl items and select Display in Solution View. The view will change to the Solution Explorer tab. The subproject containing the ActiveSync component is displayed. This is a great way to navigate the source code that is available as part of Windows Embedded CE 6.0.

* Add Additional Catalog Items to the OS Design

- > Add support for Internet Explorer 6.0
 - 1. Select the Catalog Items View tab to display the OS Design catalog.

Note	If the filtering option was not set to All Catalog Items in Catalog, you would not
	see catalog items that were not already included in the OS Design.

Enter the text Internet Explorer 6.0 Sample into the search text box to the right of the filter button. Press Enter or click the green arrow. The path Core OS | CEBASE | Internet Client Services | Browser Application | Internet Explorer 6.0 for Windows Embedded CE – Standard Components should be expanded.

Note Depending on where you are currently located in the catalog, you may have to restart the search from the top.

3. Select the Internet Explorer 6.0 Sample Browser catalog item.



- > Add support for managed code development to your OS design
 - 4. Enter the text **ipconfig** into the Search box and press Enter. The **Network Utilities (IpConfig, Ping, Route)** will be highlighted.
- **Note** Again, depending on node selected when starting a search in the catalog, you may have to restart the search from the top.
 - 5. Add the Network Utilities to your design by selecting the component.
 - 6. Enter the text wceload into the Search box and press Enter. The CAB File Installer/Uninstaller component will be highlighted. This is due to the fact that the SYSGEN name for the component is "wceload".
 - 7. Add the Cab File Installer/Uninstaller utility to your OS design.
 - 8. Enter the text sysgen_dotnetv2_support into the Search box and press Enter. The OS Dependencies for.NET Compact Framework 2.0 component will be highlighted.
 - 9. Add the OS Dependencies for .NET Compact Framework 2.0 to your OS design.

Note There are two separate components in this category. Be sure you select the one that does **NOT** have the – **Headless** modifier in its description.

* Build the OS run-time image

- 1. Select **Build | Configuration Manager...** from the Visual Studio menu to bring up the **Configuration Manager** dialog box.
- 2. Select **EVMBSP ARMV4I Release** from the Active solution configuration drop down box and then close the dialog box.
- 3. Select the Solution Explorer view by selecting the Solution Explorer tab.
- 4. In the **Solution Explorer** window, right click on the **EVMOSDesign** project (not the Solution node) and choose **Properties.** This will launch the Property Pages dialog for your OS design.
- 5. Expand the **Configuration Properties** tree and click on the **Build Options** node.
- 6. Ensure the following build options are set:
 - Enable eboot space in memory
 - Enable kernel debugger
 - Enable KITL
 - Enable profiling

EVM_3530 Property Pages	? 🛛
Configuration: Active(TI_EVM_3530, ■ Common Properties ■ Build Tree (WINCEROOT) ■ Configuration Properties ■ General ■ Locale ■ Build Options ■ Environment ■ Custom Build Actions ■ Subproject Image Settings	AR Platform: N/A Cgnfiguration Manager Build options: Buffer tracked events in RAM (IMGOSCAPTURE=1) Enable eboot space in memory (IMGEB00T=1) Enable event tracking during boot (IMGCELOGENABLE=1) Enable hardware-assisted debugging support (IMGHDSTUB=1) Enable kernel debugger (no IMGNODEBUGGER=1) Enable kernel debugger (no IMGNODEBUGGER=1) Enable rolling (IMGPROFILER=1) Enable ship build (WINCESHIP=1) Flush tracked events to release directory (IMGAUTOFLUSH=1) Bur-time image can be larger than 32 MB (IMGRAM64=1) Use xcopy instead of links to populate release directory (BUILDREL_USE_COPY=1) Write run-time image to flash memory (IMGFLASH=1)
	OK Cancel Apply

7. Select OK



8. Select **Build | Build EVMOSDesign** from the Visual Studio menu.

Note This will take several minutes to complete depending on the capabilities of your development system. The following steps for configuring connectivity may be accomplished while building.

* Configure connectivity options

- 1. Select **Target** | **Connectivity Options...** from the Visual Studio menu. The **Target Device Connectivity Options** dialog will appear showing the Kernel Service Map configuration for the **CE Device** named connection.
- 2. Select **Ethernet** from the **Download** drop down box.
- 3. Select **Ethernet** from the **Transport** drop down box.
- 4. Select KdStub from the Debugger drop down box.

Add Device	Target Device:		
Delete Device	CE Device		
	Download:		
Service Configuration	Ethernet	*	Settings
Kernel Service Map	(BSQR003200)		
Core Service Settings	Transport		
Service Status	Ethernet	-	Settings
	(BSQR003200)		
	Debugger:		
	KdStub	-	Settings
	(Prompt On Error)		

Change the device configuration

The device has a number of configurable options.

5. Connect serial cable to UART3. Connect Ethernet cable to Ethernet jack

Note 7	The device	must be on	the same	subnet as	host PC	running	PlatformBuilder
--------	------------	------------	----------	-----------	---------	---------	-----------------

6. Open Terminal program on host PC (115200, N, 8, 1)

Tera Term: Serial port setup 🛛 🔀				
<u>P</u> ort: <u>B</u> aud rate:	COM1 • OK			
<u>D</u> ata:	8 bit Cancel			
P <u>a</u> rity:	none			
<u>S</u> top:	1 bit 💌 Help			
<u>F</u> low control:	none 💌			
Transmit delay 0 msec/ <u>c</u> har 0 msec/ <u>l</u> ine				

7. In the **Target Device Connectivity Options** dialog box in Visual Studio, Click the **Settings** button next to the **Download** drop down menu.

Note Steps 8 through 13 must immediately follow step 7. Read all of these steps and be prepared for the complete set of steps before performing step 7.

- 8. Power on the EVM Board. The power switch is located on the right side of the board by the power cord.
- 9. Status messages will be displayed on serial port and a 4 color boot screen will appear on the EVM Board. On first boot only the flash will be reformatted. This can be a lengthy operation, be patient.
- 10. Monitor the boot loader progress using the serial terminal program.
- 11. Boot Menu appears on the EVM.
- 12. Wait for the EVM Board to get a DHCP address and broadcast BOOTME packets. . This will allow the Platform Builder to see your EVM on the network.



13. When the appropriate device name shows up in the Active Target Devices list in the Ethernet Download Settings dialog box, select it and click OK.

Ethernet Download Settings	×
Target <u>d</u> evice boot name:	
EVM3530-35265	
IP address: 192.168.1.113 Boot loader: 1.0	
Active target devices:	
EVM3530-35265	
<u>T</u> FTP block size in bytes:	
<u>QK</u> estore	

Note In the case of your EVM board, the device name is based on the MAC address. Each platform actually has its own method of determining a device name, which it includes in its BOOTME packet.

* Test your OS run time image on the Device

1. Select Target | Attach Device from the Visual Studio menu.

Once the download has begun, wait for the transfer. It can take up to two minutes, during which the Platform Builder dialog will include a transfer rate and an estimated time to completion.

🕹 Downloaded 27% of Runtime Image to CE Device 🛛 🔲 🔀			
	>		
Downloading: E:\WI	NCE600\OSDesigns\SampleOSDesign\EVM_353		
Estimated time left:	13 sec (4.5 MB of 16.6 MB copied)		
Download through:	Ethernet		
Transfer rate:	894 KB/sec		
Close this dialog box when download completes			
	Close Cancel		

- 2. If the Download Runtime Image dialog remains open after the download completes, click **Close this dialog box when download completes** and then click **Close**.
- 3. After the device boots to the touch calibration screen, follow the instructions to calibrate the touch screen and continue.
- **Note** During target device initialization and operation, diagnostic messages are displayed on the Debug tab of the Visual Studio output window. Some of these messages may sound serious, for example "OEMIoControl: Unsupported Code …" but do not indicate an error condition. Usually a serious error will be followed by additional failures or exceptions.

You will now be able to interact with the device and test the features of your new OS Design. Congratulations, you have successfully built and run your first Windows Embedded CE 6.0 OS Design!

If you are continuing with the next Hands-On Lab, keep your image running.

Lab 2-2: Develop and Test an Application Subproject

Objectives

- Create a simple Hello World application subproject
- Deploy the application to the device
- Debug the application running on the device

Prerequisites

• Completed Lab 2-1

Estimated time to complete this lab: 20 minutes

Lab Setup

To complete this lab, you must have:

- A development workstation running Windows XP
- Visual Studio 2005 (Version 8) with Platform Builder plug-in
- CE 6.0
- CE 6.0 2006 Roll up
- CE 6.0 Service Pack 1
- CE 2007 Updates upto month 9th month {QFEs}
- CE R2
- CE 2007 Updates 11th and 12th month {QFES}
- CE 2008 Updates
- A running image from Lab 2-1

Exercise 1 Create and configure an application subproject

In this exercise you will create and configure an application subproject.

> Create the subproject

- 1. Click on the Solution Explorer tab to display the Solution Explorer.
- 2. Locate the **Subprojects** node below the **EVM_3530** project in the Solution Explorer window.
- 3. Right click on the **Subprojects** node and select **Add New Subproject...** The Windows Embedded CE Subproject Wizard will appear.
- 4. Select the WCE Application template.
- 5. Type MyHelloWorldApp in the Subproject name text box.

Windows Embedded CE Subproject Wizard 🔹 💽 🔀				
Select name, location and template				
Available templates: WCE Application WCE Console Application WCE Dynamic-Link Library WCE Static Library WCE TUX Dynamic-Link Library	Subproject name: MyHelloWorldApp Logation: E:\WINCE600\OSDesigns\EVMOSDesign\EVMOSDe	15		
< <u>P</u> r	evious <u>N</u> ext > <u>F</u> inish Can	cel		

- 6. Click Next.
- 7. Select **A typical "Hello World" application** and click **Finish**. The wizard will create the files necessary for the typical Hello World application subproject.

Configure the subproject image settings

We will configure the subproject settings so that we can easily debug it without needing to rebuild our OS Design. This is a good debugging technique that can save development time. We will use this same technique in most future labs.

- 1. Right click on the EVM_3530 OSDesign project in the Solution Explorer and select Properties.
- 2. From the Configuration drop down select All Configurations.
- **Note** Remembering to select **All Configurations** can save a lot of time when switching between configurations. The following labs will reference this procedure and each time you should make sure to select **All Configurations** from the **Configuration** drop down.
 - 3. Expand the **Configuration Properties** node and select **Subproject Image Settings**.
 - 4. Double click the **MyHelloWorldApp** entry in the **Project settings in run-time image** box. The **Edit Run-Time Image Settings** dialog box will appear.
 - 5. Select the **Exclude from image** and **Always build and link as debug** check boxes, and click **OK**.
 - 6. Click OK on the EVM_3530 OSDesign Property Pages dialog.

4 Lab 2-2 Develop and Test an Application Subproject

🍫 EVM_3530 - Microsoft Visual Studio					
File Edit Wew Project Build Debug Target Data Tools Window Community Help					
Device: CE Device					
Solution Explorer - Solution 'EVM_3530' (1 project)	EVM 3530 Droporty Dagor	2 🛛			
G.	rem_5550 Froperty Fages				
Solution 'EVM_3530' (1 project)	Configuration: Active(TI_EVM_3530_AR 💙 Platform: N	Configuration Manager			
	Common Properties Project settings in ru	n-time image:			
Favorites	Configuration Properties Project	Build Image Always Debug			
Parameter Files	- General MultielloW/orld@po	Included Included No			
JUKS JUKS JUKS JUKS	-Build Options	ettings 🛛			
MyHelloWorldApp (E:/WINCE600/OSDesigns/SampleOSDesigner)	Environment Selected workspace project				
Include files Parameter files	Subproject Image Setti				
Resource files	Exclude from build				
Source files Source files	Exclude from image				
- prelink.bat	Always build and link as	lebug			
E ReadMe.txt		OK Cancel			
		Edit			
		OK Cancel Apply			
<					
Solution Explorer 👌 Catalog Items View 🐼 Class View					
🔀 Code Definition Window) 🔁 Cali Browser 🗊 Output 🖼 Find Results 1 👸 Error List					
Ready					
🛃 Start 🖉 Microsoft Outlook We 🔁 CE6.0 Labs On EV	🔅 EVM_3530 - Microsoft 📑 Lab 2-2_EVM.do	:- M 🤇 🏷 🖓 🖓 12:40 PM			

- 7. Select Build | Targeted Build Settings from the Visual Studio menu.
- 8. Ensure that **Make Run-Time Image After Building** does NOT have a check mark beside it. If it does, deselect it clicking on the menu item.
- **Note** This step will prevent the OS run-time image from being rebuilt after we build individual subprojects. This setting will persist for all targeted builds through out the life of this OS Design.

Set a breakpoint in the application

- 1. Locate and expand the **MyHelloWorldApp** subproject in the **Subprojects** node of the Solution Explorer.
- 2. Expand the Source files node of the MyHelloWorldApp subproject.
- 3. Double click the **MyHelloWorldApp.cpp** file. The file will load in the Visual Studio editor.
- 4. Locate the WndProc() function near the bottom of the file.
- 5. Click on the **DrawText(...)** function call and press **F9** to set a breakpoint there.



- Build and run your subproject
 - 1. Right click the **MyHelloWorldApp** subproject in the Solution Explorer and select **Build**. The application will build and should complete with 0 errors and 0 warnings in the build output window.
- **Note** Your Device instance should still be running from the previous lab. If it is not, you can restart it now by choosing **Target** | **Attach Device** from the Visual Studio menu.
 - 2. Select Target | Run Programs... from the Visual Studio menu.
 - 3. Select **MyHelloWorldApp.exe** from the Available Programs box, and click on **Run.**

6 Lab 2-2 Develop and Test an Application Subproject

Run Program	X
Available Programs:	Bup
iltiming.exe loaddbg.exe MyHelloWorldApp.exe	Cancel
nk.exe oal.exe osbench.exe	
oscapture.exe rapisrv.exe repllog.exe	
rnaapp.exe	
Execution command line:	
MyHelloWorldApp.exe 🗸	

The kernel debugger will halt execution at the breakpoint we just set. Notice the yellow arrow inside the red circle at the line where we set our breakpoint in the source code file. This indicates the next statement to be executed.

- 4. Look at the **EVM Board.** Notice that the MyHelloWorldApp application is running, but the Hello World! string has not yet been drawn on the display.
- 5. Press F10 or select Debug | Step Over from the Visual Studio menu.
- 6. Look at the **EVM Board** again. Notice that the Hello World! string has now been printed on the display.
- 7. Press F5 or select Debug | Start to allow the EVM to continue running.

Note The sample application does not include a mechanism to allow it exit. We must close the application using the capabilities of the development environment.

8. Select **Debug** | **Windows** | **Processes** to bring up the **Process** window. This window shows all processes running on the Device.

Process				×
			_ 4	> \$8 😣 📝 🖻
Process Name	Process	AccessKey	Process	CurZone
emulatorstub.exe	0x00010000	0x00000000	0x04E80002	0x00000000
explorer.exe	0x00010000	0x00000000	0x04E10002	0x00000000
myhelloworldapp.exe	0x00010000	0x00000000	0x04E40006	0x00000000
nk.exe	0x80070000	0x00000000	0x00400002	0x0000000B
servicesd.exe	0x00010000	0x00000000	0x050E0002	0x0000000B
shell.exe	0x00010000	0x00000000	0x01110002	0x00000000
udevice.exe	0x00010000	0x00000000	0x01CB0002	0x0000000B
udevice.exe	0x00010000	0x00000000	0x01B30006	0x0000000B
udevice.exe	0x00010000	0x00000000	0x016A000A	0x0000000B
udevice.exe	0×00010000	0x00000000	0x04170002	0x0000000B

- 9. Right click on the **myhelloworldapp.exe** process and select **Terminate** to kill the process.
- 10. Select **Yes** to verify. The Process window will refresh after a short delay and the application will be gone.
- 11. Close the **Process** window by clicking on the **X** in the upper right hand corner.

Congratulations! You have successfully created, built and tested a simple Windows Embedded CE 6.0 application on your OS Design. We will follow a similar methodology in future labs.

If you are continuing with the next Hands-On Lab, keep your image running.

Lab 2-3: Using the Remote Tools

Objectives:

- Use the Remote System Information tool to see information about your device
- Use the Remote File Viewer to explore and change files on your device
- Use the Remote Performance Monitor to examine system resource loading

Prerequisites

• Completed Lab 2-1

Estimated time to complete this lab: 30 minutes

Lab Setup

To complete this lab, you must have:

- A development workstation running Windows XP
- Visual Studio 2005 (Version 8) with Platform Builder plug-in
- CE 6.0
- CE 6.0 2006 Roll up
- CE 6.0 Service Pack 1
- CE 2007 Updates upto month 9th month {QFEs}
- CE R2
- CE 2007 Updates 11th and 12th month {QFES}
- CE 2008 Updates
- A running image from Lab 2-1

Exercise 1 Using the Remote File Viewer

In this exercise you will use the Remote File Viewer to transfer files between your development workstation and EVM CE 6.0 target device.

Note Your Device should still be running from the previous lab. If not, restart it by selecting **Target** | **Attach Device** from the Visual Studio menu.

- > Starting the Remote File Viewer
 - 1. Select **Target** | **Remote Tools** | **File Viewer** from the Visual Studio menu. The **Select a Windows CE Device** dialog will appear.
 - 2. Expand the **Windows CE Default Platform** node and select **Default Device.** Click **OK**. Visual Studio will begin transferring the required files for the Remote File Viewer to the EVM.

Select a Windows CE Device			
Windows CE Default Platform Comparison Default Device			
<u> </u>			

Note You will get a dialog box asking for the location of an executable. This is because the kernel debugger running on the device is attempting to monitor all processes that run on the device, including the remote tools. This can not be done because the debugging information is not available for these tools, and we can safely ignore the request. We will configure Visual Studio to suppress these dialogs in the future. Note that this issue only occurs because we have left the kernel debugger running inside our OS run-time.

- 3. Select **Don't display this dialog again** in the **Find Executable** dialog box, and select **Cancel**. The Remote File Viewer tool should connect.
- Note The Remote File Viewer may not connect if there has been too long of a delay while you were canceling the Find Executable dialog. If this occurs, you will need to restart the EVM by selecting **Target | Detach Device** followed by **Target | Attach Device**.

> Explore the device file system

- 4. Expand the **Default Device** node in the left hand pane and select the **Windows** directory. The right hand pane shows a list of the files in the Windows directory on the target device.
- 5. Select **View** | **Details** from the Remote File Viewer menu to see the details of each file in the folder.



Copy a file from the target device to the development workstation

- 6. Select **WindowsCE.jpg** in the right hand pane. We will copy this file from the device to the PC.
- 7. Select File | Import File from the Remote File Viewer menu.
- 8. Save the file to a convenient folder on your development workstation desktop.

> Copy a file to the target device

- Select the Desktop folder in the right hand pane of the Remote File Viewer. Expand the Windows folder, if necessary, to find the Desktop folder. Select File | Export File from the Remote File Viewer menu.
- 10. In the **Export File** dialog select the **WindowsCE.jpg** file from the development workstation and click Open. The file will be transferred from the development workstation to the device.
- 11. Close the **Remote File Viewer** application.

Exercise 2 Remote System Information

In this exercise you will use the Windows CE Remote System Information tool to examine system settings and properties of the EVM running your OS Design.

- > Launch the Remote System Information tool
 - 1. Select **Target** | **Remote Tools** | **System Information** from the Visual Studio menu.
 - 2. Select **OK** to accept the Default Device configuration. There will be a delay while the System Information tool is transferred to the device and gathers information.

> Explore System Information data

3. Expand the **System Information** node and select **System Summary**. Details of the OS version, current time and time zone settings, and locale are presented on the right-hand pane.



- 4. Select **Components** | **Memory** in the left hand pane. The total and available memory, the fraction of program memory in use (Memory load), the amount of memory allocated to the object store, and other system memory statistics are reported.
- 5. Expand **Components** | **Devices** in the left hand pane. Observe the list of devices detected. You can click on individual devices to see information available about each device.

6 Lab 2-3 Using the Remote Tools

- 6. Browse the other information available from the tool.
- 7. Close the Windows CE Remote System Information tool.

Exercise 3 Remote Performance Monitor

In this exercise, you will use the Windows CE Remote Performance Monitor to log consumption of system resources and other performance related metrics on a EVM CE 6.0 target device.

- Launch the Remote Performance Monitor
 - 1. Select Target | Remote Tools | Performance Monitor from the Visual Studio menu.
 - 2. Click **OK** to use the Default Device connection. The Windows CE Remote Performance Monitor tool will load and show the Chart view.

> Add Performance counters to the Chart view

- 3. Select Edit | Add to chart from the Remote Performance Monitor tool to bring up the Add to Chart dialog.
- 4. Select CE Process Statistics from the Object drop down box.
- 5. Select % Processor Time from the Counter drop down box.
- 6. Select _Total from the Instance drop down box.
- 7. Click on Add.

🕅 Windows CE Remote Performance Monitor			💶 🖻 🔀
File Edit View Options Connection Help			
100 96 32 88 64 60 56 52 44 40 36 32 28 24 20 16 12 8 4 0	Add to Chart Object CE Process Statistics (Legacy) Instance Counter: Kernel Time R User Time Hop Memory ProcessO ProcessO ProcessO Colog: Scale: Default Width: —	Image: Centrel of the second seco	
Last 0.000 Average 0.000 Min 0.000 Max 0.0	00 Graph Time 100.000		
Color Scale Counter	Instance	Parent	Object
Data: Current Activity			
Start Microsoft Outlo	Lab 2-3 EVM d 🖉 NK (Rupping) - 🔲 Tara Tara	C Readme byt - N St Windows CE P	

- 8. Select **CE Memory Statistics** from the **Object** drop down box.
- 9. Select Memory Load from the Counter drop down box.
- 10. Click Add and then click Done.
- 11. On the Windows CE desktop, drag an icon rapidly and note the increase in % **Processor Time**.



> Create an Alert view and add performance counters

- 12. Select View | Alert from the Remote Performance Monitor menu.
- 13. Select Edit | Add to Alert....
- 14. Select CE Process Statistics from the Object drop down box.
- 15. Select % Processor Time from the Counter list.
- 16. Select _Total from the Instance list.
- 17. Select the Over radio button in the Alert if group and enter the number 10.
- 18. Click on Add and then Done.

19. Open **My Device** on the device desktop to generate processor activity. The configured alert should fire.

20. Close the **Remote Performance Monitor** tool.

If you are continuing with the next Hands-On Lab, keep your image running.
Lab 3-1: Using the Remote Process Viewer

Objectives

• Use the Remote Process Viewer to explore the processes and threads running on a Windows Embedded CE 6.0 device

Prerequisites

• Completed Lab 2-1

Estimated time to complete this lab: 15 minutes

Lab Setup

To complete this lab, you must have:

- A development workstation running Windows XP
- Visual Studio 2005 (Version 8) with Platform Builder plug-in
- CE 6.0
- CE 6.0 2006 Roll up
- CE 6.0 Service Pack 1
- CE 2007 Updates upto month 9th month {QFEs}
- CE R2
- CE 2007 Updates 11th and 12th month {QFES}
- CE 2008 Updates
- A running image from Lab 2-1

Exercise 1

In this exercise, you will use the Windows CE Remote Process Viewer to examine the processes and threads running on a Windows CE target device.

Note Your Device should still be running from the previous lab. If not, restart it by selecting **Target** | **Attach Device** from the Visual Studio menu.

- Launch the Remote Process Viewer
 - 1. Select Target | Remote Tools | Process Viewer from the Visual Studio menu.
 - 2. Click **OK** to use the **Default Device** connection. The Windows CE Remote Process Viewer tool will load.



> Explore running processes and threads

3. Examine the list of active **processes** in the upper pane.

🔏 Windows CE Rem	ote Process Viewe	er					
File View Connection	Help						
E 😓 🔁 🐲 🗶							
Process	PID	Base Pri	ority # Threa	ds Ba	se Addr	Access Key	Window
NK.EXE	00400002	3	66	80	070000	00000000	CursorWindow
shell.exe	01110002	3	1	00	010000	00000000	
udevice.exe	01CB0002	3	5	00	010000	00000000	
udevice.exe	01B30006	3	1	00	010000	00000000	
udevice.exe	0176000A	3	1	00	010000	00000000	
udevice.exe	04160002	3	1	00	010000	00000000	
explorer.exe	04E20002	з	4	00	010000	00000000	Task Manager
EmulatorStu	. 04E90002	з	1	00	010000	00000000	
servicesd.exe	050F0002	3	6	00	010000	00000000	BluetoothSVC
CEMGRC.EXE	03450056	3	з	00	010000	00000000	
CEPWCLI.EXE	0496000A	3	2	00	010000	00000000	
Thread ID		Current PID		Thread Pr:	iority	Access Key	<u>^</u>
05E40002		00400002		251		00000000	
05CF0002		00400002		249		00000000	_
04DC0002		00400002		249		00000000	
04D70002		00400002		249		00000000	
04D30002		00400002		109		00000000	
04BF0002		00400002		251		00000000	
04B80002		00400002		240		00000000	
04B60002		00400002		249		00000000	
04B30002		00400002		251		00000000	
04B10002		00400002		249		00000000	~
<							
Module	Module ID	Proc Count	Global C	Base Addr	Base Size	hModule	Full Path 🔼
cetlkitl.dll	95670EF4	2	2	411B0000	24576	95670EF4	∖Release ──
cetlstub.dll	9569BE10	1	1	411C0000	20480	9569BE10	\Windows
toolhelp.dll	97F04478	2	2	40180000	24576	97F04478	\Windows
timesvc.dll	95670588	1	1	40CE0000	36864	95670588	\Windows
obexsrvr.dll	9564681C	1	1	40CB0000	126976	9564681C	\Windows
wspm.dll	956437A4	1	1	40370000	32768	956437A4	\Windows 🥃
Conselli dil	07555600	11	11	40010000	1126400	07555600	(Windows)
Ready				Connected	Default Device		NUM

4. Click on **NK.EXE** in the Process pane. Thread details are presented in the center pane.

on Windows CE Rer	note Process View	er										
<u>F</u> ile ⊻iew <u>C</u> onnection	n <u>H</u> elp											
	r											
Process	PID	Base H	riority #	Thread	is	Base A	ddr	Acc	ess Key	Win	dow	^
NK.EXE	00400002	3	8	8		881000	00	000	00000	Cur	sor₩indo	DW ≣
shell.exe	01470002	3	1			000100	000	000	00000			_
udevice.exe	01E40002	3	2			000100	000	000	00000			
udevice.exe	013C000A	3	2			000100	000	000	00000			
udevice.exe	02250002	3	1			000100	000	000	00000			
udevice.exe	02530002	3	1			000100	100	000	00000			_
udevice.exe	05660002	3	1			000100	100	000	00000			~
												>
Thread ID		Current P	ID		Thread	Priori	ty		Access	Key		^
06700002		00400002			249				0000000	0		_
066A0002		00400002			249				0000000	0		
06630002		00400002			251				0000000	0		
06500002		00400002			249				0000000	0		
063A0002		00400002			95				0000000	0		
06200002		00400002			251				0000000	0		
06180002		00400002			249				0000000	0		~
<												>
Module	Module ID	Proc Coun	t Global	C	Base Ad	dr	Base S	ize	hModule	Fi	ill Path	1 🔼
toolhelp.dll	8B8BB764	3	3		400E000	0 :	24576		8B8BB76	4 💊	Vindows.	–
cetlkitl.dll	897F2918	3	3		400F000	0 :	24576		897F291	8 \1	Release.	
cetlstub.dll	89997BF8	2	2		4010000	0 :	20480		89997BF	8 🗸	Vindows.	• • •
<						-				· ·		>
Ready					Connect	ed Def	fault Device				NUM	

5. The DLLs loaded by a process are listed in the lower 'Module' pane. Note the base address of **coredll.dll**.

Module	Module ID	Proc Count	Global Count	Base Addr	Base Size
toolhelp.dll	97F04478	2	2	40180000	24576
timesvc.dll	95670588	1	1	40CE0000	36864
obexsrvr.dll	9564681C	1	1	40CB0000	126976
wspm.dll	956437A4	1	1	40370000	32768
coredll.dll	97FFE6CC	11	11	40010000	1126400
ceshell.dll	955F1AD4	1	1	40D10000	491520
ssllsp.dll	956345A4	1	1	403B0000	94208
msim.dll	956218C8	1	1	402E0000	102400
fpcrt.dll	97F109C0	2	2	40130000	98304
btdrt.dll	95621600	1	1	404D0000	110592
htsvc.dll	956214C8	1	1	404F0000	77824
<					
Ready					Connected [

6. Click on **shell.exe** in the Process pane. This process has one thread and loads two modules. Note that coredll.dll is loaded at the same base address in **shell.exe** as it was in **NK.EXE**.

old Windows CE Rer	note Process View	er					
<u>File View Connection</u>	n <u>H</u> elp						
	r						
Process	PID	Base P	riority # Th	reads	Base Addr	Access Key	Window 🔼
NK.EXE	00400002	3	88		88100000	00000000	CursorWindow
shell.exe	01470002	3	1		00010000	00000000	
udevice.exe	01E40002	3	2		00010000	00000000	
udevice.exe	013C000A	3	2		00010000	00000000	
udevice.exe	02250002	3	1		00010000	00000000	
udevice.exe	02530002	3	1		00010000	00000000	
udevice.exe	05660002	3	1		00010000	00000000	~
<							>
Thread ID		Current PI	D	Thread	Priority	Access	Key
01480002		01470002		130		000000	10
Module	Module ID	Proc Count	Global C.	. Base Ad	ir Base :	Size hModule	Full Path
toolhelp.dll	8B8BB764	1	3	400E000	24576	8B8BB76	4 ∖Windows
cored11.d11	8BADE6B4	1	23	4003000	0 536571	6 8BADE6B	4 \Windows
Ready				Connect	ed Default Devi	ice	NUM

- 7. Click on **explorer.exe** in the Process pane. This process has many threads and many loaded modules. Scroll to the bottom of the module list. Note that **coredll.dll** is loaded at the same base address as in the other processes.
- 8. Close the Remote Process Viewer.

If you are continuing with the next Hands-On Lab, keep your image running.

Lab 3-2: Exploring the Heap

Objectives

• Become familiar with the Windows Embedded CE 6.0 heap

Prerequisites

• Completed Lab 2-1

Estimated time to complete this lab: 30 minutes

Lab Setup

To complete this lab, you must have:

- A development workstation running Windows XP
- Visual Studio 2005 (Version 8) with Platform Builder plug-in
- CE 6.0
- CE 6.0 2006 Roll up
- CE 6.0 Service Pack 1
- CE 2007 Updates upto month 9th month {QFEs}
- CE R2
- CE 2007 Updates 11th and 12th month {QFES}
- CE 2008 Updates
- A running image from Lab 2-1

Exercise 1 Create and build the HeapTest1 subproject

The purpose of this exercise is to create, configure and build the subproject that we will use to examine the Windows Embedded CE 6.0 heap.

- Create HeapTest1 Subproject
 - 1. Select Project | Add New Subproject... from the Visual Studio menu.
 - 2. Enter the WCE Application subproject name: HeapTest1 as shown below.

Windows Embedded CE Subproject Wizard								
Select name, location and template								
Available templates: WCE Application WCE Console Application WCE Dynamic-Link Library WCE Static Library WCE TUX Dynamic-Link Library	Subproject name: HeapTeat1 Logation: E:\WINCE600\OSDesigns\EVMOSDesign\EVMOSDes							
< <u>P</u> r	evious Next > Einish Cancel							

Note By default, new subprojects are located in the current OS Design folder.

- 3. Click Next to continue the New Subproject Wizard.
- 4. Select A simple Windows Embedded CE application.

5. Click **Finish** to finish the wizard. The **HeapTest1** subproject can be accessed in the Subprojects folder in Solution Explorer.

Configure Subproject For Debug

- 6. From the Solution Explorer, right-click on the **EVMOSDesign** project and select **Properties.**
- 7. Expand the Configuration Properties tree select Subproject Image Settings.
- 8. Select All Configurations from the Configuration drop down.
- 9. Double click the **HeapTest1** subproject to bring up the **Edit Run–Time Image Settings** dialog.
- 10. Check the boxes **Exclude from image** and **Always build and link as debug** and click **OK**.
- **Note** Exclude from image will prevent the subproject from being included in the OS run-time image if it is built in the future. This is one way to allow the application to be run directly from the operating system build output folder (typically referred to as the flat release directory.)

Always build and link as debug will make the kernel debugger experience better by disabling compiler optimizations in a release build for this application.

11. Click OK to close the EVMOSDesign Property Pages dialog.

Build the application subproject

- 12. Right click on the HeapTest1 subproject in the Solution Explorer and select **Build**. The application should build with zero errors and warnings.
- **Note** This is referred to as a **Targeted build**. We built a specific subproject in the Solution Explorer. Builds that are initiated from the Visual Studio Build menu will cause the entire solution or project to be built (sometimes referred to as a **Global build**).

Global builds and Targeted builds can be separately configured with regard to whether they also cause the OS run-time image to be rebuilt. These settings are located in **Build | Global Build Settings** and **Build | Targeted Build Settings** from the Visual Studio menu. We previously configured the Targeted Build Settings not to rebuild the OS run-time image.

> Set a breakpoint in the application

- 13. Expand the HeapTest1 subproject in the Solution Explorer.
- 14. Open the **Source files** branch and double-click on the **HeapTest1.cpp** file to open it in the Visual Studio editor.
- 15. Set a breakpoint in this source file by clicking on the **return** statement and pressing **F9**. We will know that we have built and run the application successfully when we hit this breakpoint.

Æ	HeapTest1.cpp Catalog Item Dependencies Start Page
(Glo	ibal Scope) 💌
[<pre> // HeapTest1.cpp : Defines the entry point for the application. // #include "stdafx.h"</pre>
[int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPTSTR lpCmdLine, int nCmdShow)
	// TODO: Place code here.
G	<pre>return 0; }</pre>

- > Run the application on the target device
 - 16. Select **Target** | **Run Programs** from the Visual Studio menu to bring up **Run Program** dialog.
 - 17. Select HeapTest1.exe, and click Run.

Run Program	
Available Programs:	
eventrst.exe	Bun
Itiming.exe	Cancel
MyHelloWorldApp.exe	
osbench.exe oscapture.exe	
Execution command line:	
HeapTest1.exe 💌	

18. Observe that the kernel debugger halts execution at the previously set breakpoint. We have successfully created, built and run our application.



19. Leave the application at the breakpoint.

Exercise 2 Explore Process Model of Simple Application

This exercise will use some of the debug capabilities of Platform Builder to explore some features of the HeapTest1 application.

- Examine HeapTest1 process details
 - 1. Select **Debug** | **Windows** | **Threads** from the Visual Studio menu to bring up the **Threads** window. This window will allow you to see information for all the threads in a particular process,
 - 2. Select **heaptest1.exe** from the **Process** drop down box. Note that heaptest1.exe has a single thread running at the default thread priority.
 - 3. Expand the thread node to show the **call stack**.

Threads					2005. 		ia divisi		×
Process: heaptest1.exe								😼 🍫 📢 🖼	3 🧭 🗗
hThread	pThread	RunState	InfoStatus	WaitState	hVMProc	hOwnerProc	CurPrio	BasePrio	Ker 🐴
HEAPTEST1 WinMain() HEAPTEST1 WinMainC HEAPTEST1 WinMainC COREDLL ! MainThread f101fffc()	0x9490B240 HINSTANCE_ * 0x ATStartupHelper(ATStartup() lin BaseFunc(HINSTAN(Awak, Rung 55550002, HIN 1 line 71 + 105 CE * 0x0000	UMode ISTANCE * 0x 20 bytes 00000, HINSTAN	Blocked :00000000, un ICE * 0x000	0x05C50002 signed short 00000) line	0x05C50002 * 0x0002fe8c 1068 + 92 by	251 , int 0x000 tes	251 00005) line :	0 13 >

- **Note** You can double click on any of the entries in the call stack to attempt to see the source code at that point. If you have installed the Shared Source you would be able to view the source code for these functions. Otherwise you will see the disassembly view.
 - 4. Close the Threads and Call Stack windows.
 - 5. Select **Debug** | **Windows** | **Modules** from the Visual Studio menu. The **Modules** window will appear. This view shows all modules running on the target device.

Modules						×
Process: ALL				😵 🍄 🔽	🖾 🖾	🀮 🎽 🔐
Module	Image Address Range	Module Handle	Usage Mask	Relocated Data Addre	Status	Tim 📥
gwes.dll	0xC02A0000 - 0xC0463FFF	0x95463DF8	0x00000001	0xC0443000 - 0xC0449E90	Loaded	0x0(
hd.dll	0xC0050000 - 0xC0057FFF	0x97FDB378	0x00000001	0xC0055000 - 0xC0055994	Loaded	0x0(
heaptest1.exe	0x00010000 - 0x00013FFF	0x043B000A	0x00000001		Loaded	0x00
ieceext.dll	0x40830000 - 0x40838FFF	0x955F6948	0x00000002	0x40835000 - 0x40835118	Loaded	0x0(
imaging.dll	0x40710000 - 0x407BEFFF	0x956A439C	0x00000002	0x407B6000 - 0x407B6414	Loaded	0x0(
iphlpapi.dll	0x40330000 - 0x4034CFFF	0x954631F8	0x00000003	0x4034A000 - 0x4034AAF0	Loaded	0x0(
ipv6hlp.dll	0xCOAF0000 - 0xCOB08FFF	0x9543AE94	0x00000001	0xC0B05000 - 0xC0B0689C	Loaded	0x0(
ircomm.dll	0xC06F0000 - 0xC06FBFFF	0x9542F300	0x00000001	0xC06F9000 - 0xC06F9624	Loaded	0x0(—
irdastk.dll	0xC0BF0000 - 0xC0C2BFFF	0x953CA480	0x00000001	0xC0C26000 - 0xC0C28364	Loaded	0x0(
irsir.dll	0xD2820000 - 0xD2831FFF	0x95369C30	0x00000001	0xD282F000 - 0xD282F694	Loaded	0x0(
k.ceddk.dll	0xC0600000 - 0xC060DFFF	0x96D0A904	0x00000001	0xC060B000 - 0xC060B140	Loaded	0x0(
k.coredll.dll	0xC00C0000 - 0xC01CDFFF	0x97FFE09C	0x00000006	0xC01B9000 - 0xC01B9FC8	Loaded	0x0(
k.dhcpsrv.dll	0xC0D70000 - 0xC0D75FFF	0x9536981C	0x00000001	0xC0D73000 - 0xC0D734DC	Loaded	0x0(
k.fpcrt.dll	0xC01D0000 - 0xC01E7FFF	0x97FB2600	0x00000001	0xC01E5000 - 0xC01E5084	Loaded	0x0(
k.iphlpapi.dll	0xC0700000 - 0xC071CFFF	0x9539678C	0x00000001	0xC071A000 - 0xC071AAF0	Loaded	0x0(
k.nspm.dll	0xC0760000 - 0xC0768FFF	0x954AFC54	0x00000001	0xC0766000 - 0xC07661BC	Loaded	0x0(
k.ssllsp.dll	0xC07A0000 - 0xC07B6FFF	0x954285DC	0x00000001	0xC07B3000 - 0xC07B4058	Loaded	0x0(
k.touch.dll	0xC10F0000 - 0xC10FBFFF	0x955A7AA4	0x00000001	0xC10F9000 - 0xC10F96B8	Loaded	0x0(🔽
<						>

- 6. Observe that the heaptest1.exe module has an Image Address starting at 0x00010000. This same starting address is used by all processes on the device with the exception of nk.exe.
- 7. Select **heaptest1.exe** from the **Process** drop down box. This shows just the module information related to the heaptest1.exe process.
- 8. Observe that the only module listed other than heaptest1.exe is **coredll.dll**. Coredll.dll is generally loaded by every process as it provides access to most system APIs.
- 9. Close the **Modules** window.
- 10. Press F5 to allow this application to continue running, and it will exit.

Remove breakpoint

- 11. Click on the line containing the breakpoint in heaptest1.cpp
- 12. Press F9 to remove the breakpoint.

Exercise 3 Local Heap

The purpose of this exercise is to explore the implementation of Local Heap memory in a simple Windows CE application. This application will allocate a number of blocks, free the blocks and allocate the blocks again. You will use Platform Builder tools to view the heap structures and to demonstrate the behavior of the heap management algorithm.

- > Copy HeapTest1 code to HeapTest1.cpp
 - 1. Open HeapTest1.cpp file from HeapTest1 subproject if not already open.
 - 2. Open Lab 3-2 HeapTest code.txt file from the Student files, and then copy the code snippet to HeapTest1.cpp as below.



3. Save HeapTest1.cpp.

Build and run HeapTest1.exe

- 4. Right click on the HeapTest1 subproject in the Solution Explorer and select **Build**.
- 5. Run HeapTest1.exe using the Target | Run Programs menu in Visual Studio.
- 6. Examine the debug output window in Visual Studio, which should be similar to the following:

🏶 EVM_3530 (Running) - M	icrosoft Visual Studio 📃 🖻 🔀
File Edit View Project Bu	uild Debug Target Data Tools Window Community Help
i 🛅 + 🛅 + 💕 📕 🗿 🕹	🗈 🕄 ウ・C・ - 見・艮 ト 🛛
🕨 n 🖬 🖬 🗘 🕫 🖓	🎽 Hex 🗔 • 🖕 🧾 💦 🔹 👘 👘 👘 👘 👘 👘 👘
Device: CE Device 🔹 🖓	
Solution Explorer - So 👻 📮 🗙	HeapTest1.cpp MutexDemo.cpp
	(Unknown Scope)
SDKs Subprojects Subprojects HeapTest1 (E:/W HeapTest1 (E:/W Include files	<pre>// TODO: Place code here. #define HEAP_BLOCK_SIZE 32 // allocate heap block 1 char *pbuf1 = (char *) LocalAlloc(LPTR, HEAP_BLOCK_SIZE); RETAILMSG(1, (TEXT("HeanTest1 block 1 (%d bytes) at %08Xb")).</pre>
Output	× + +
Show output from: Windows (E D 12041314 PID:4980032 TI s HeapTest1 18:41:45 0 End s HeapTest1 18:41:45	Vebug Image: Constraint of the second s
12041314 PID:4980032 TI 12041314 PID:4980032 TI 12041314 PID:4980032 TI 12041314 PID:4980032 TI 12041314 PID:4980032 TI 12041314 PID:4980032 TI 12041314 PID:4980032 TI	D:4990032 HeapTestl block 3 (32 bytes) at 00030580h D:4990032 HeapTestl block 1 (32 bytes) at 00030540h freed D:4990032 HeapTestl block 2 (32 bytes) at 00030560h freed D:4990032 HeapTestl block 3 (32 bytes) at 00030580h freed D:4990032 HeapTestl block 4 (32 bytes) at 00030580h D:4990032 HeapTestl block 5 (32 bytes) at 00030580h
12041314 PID:4980032 TI	D:4990032 HeapTestl block 6 (32 bytes) at 000305C0h
Cutout C Error List	
start 🔅 EV	🛄 Ter 💿 Mic 🖡 Wi 🗿 Mic 🌔 2 N. + 🗋 2 W 🖳 Lab 🌾 🏪 6:43 PM

Three identically sized blocks were allocated from the local heap and then freed in the same sequence they were allocated. Three more identically sized blocks were then allocated again.

Observe the addresses of each heap allocation in the debug output. Notice that the second group of allocations did not start at the same point as the first group even though those blocks had been freed and were available for use. Instead, the second set of allocations was given addresses starting with the address of the last item in the original group. The virtual address range that originally backed the first two memory allocations is now unused.

This demonstrates one of the characteristics of Windows Embedded CE 6.0 heaps. New heap allocations are taken from the last allocated or freed item first; the heap manager does not start allocation attempts at the beginning of the heap. In this case, only block 3 was reused because it was the last one previously freed. The virtual memory would have been used more efficiently if the application had freed the memory in the reverse order that it had been allocated.

Run Remote Heap Walker Viewer

- 7. From the Visual Studio Platform Builder menu, select **Target** | **Remote Tools** | **Heap Walker.**
- 8. Click **OK** to select the **Default Device** configuration. The Remote Heap Walker window will appear.
- 9. Double click on the line containing **HeapTest1.exe** in the process list. A second window containing the process heap information will appear.

Windows CE Remo	te Heap Walker - H	leap_List for He	apTest1.exe	ProcessId=05CA	
File View Window Con	nection Help				
Process_List					
Heap Id	Process Id	Process	1	Flag	~
0xD048EA00 0xD048DA00 0xD048D700 0xD048B4A0 0xD00487E0 0xD00451C0 0xD0042D20 0xD00409C0 0xD00409C0 0xD0040700 0xD0040480 0xD0040000 0x00040000 0x00030F80 0x00030000	0x00400002 0x00400002 0x00400002 0x00400002 0x00400002 0x00400002 0x00400002 0x00400002 0x00400002 0x00400002 0x00400002 0x00400002 0x00400002 0x00400002 0x0019A0002	NK.EXE NK.EXE NK.EXE NK.EXE NK.EXE NK.EXE NK.EXE NK.EXE NK.EXE Shell.e udevice	exe e.exe e.exe	HF32_DEFAULT HF32_DEFAULT HF32_DEFAULT	
Heap_List for Hea	apTest1.exe Proce	ssld=05CA0002	HeapId=000	30000	
Address	BlockSize	Flags			
0x00030000 0x00030460 0x00030480 0x000304A0 0x000304C0 0x000304E0 0x00030520 0x00030560 0x00030560 0x00030580 0x000305A0 0x000305C0	1120 32 32 32 32 64 64 64 32 32 32 32 2624	Fixed Fixed Fixed Fixed Fixed Free Fixed Fixed Fixed Free			
Ready			Connected	Default Device	

10. Double click on the heap entry at the **address** corresponding to **block 4** in the debug window. This will be the first Fixed block after the first Free block.



- 11. Observe the first byte is a '4' (0x34). This entry was written by the HeapTest1 application to identify its memory block.
- 12. Take some time and explore the source code to the application and what the tools can tell you about it. Step through the code and examine the various Platform Builder tools as you go through. This code could be used as a starting point to examine other aspects of heap management.
- 13. Close the Remote Heap Walker window.

Terminate the HeapTest1 application

- 14. Select **Debug** | **Windows** | **Processes** from the Visual Studio menu.
- 15. Right click on the heaptest1.exe process and select Terminate.
- 16. Click Yes to confirm.
- 17. Close the Processes window.
- If you are continuing with the next Hands-On Lab, keep your image running.

Lab 3-3: Scenario - Fixing a Memory Leak

Objectives

• Use the Target Control utility to identify a memory leak

Prerequisites

• Completed Lab 2-1

Estimated time to complete this lab: 30 minutes

Lab Setup

To complete this lab, you must have:

- A development workstation running Windows XP
- Visual Studio 2005 (Version 8) with Platform Builder plug-in
- A development workstation running Windows XP
- Visual Studio 2005 (Version 8) with Platform Builder plug-in
- CE 6.0
- CE 6.0 2006 Roll up
- CE 6.0 Service Pack 1
- CE 2007 Updates upto month 9th month {QFEs}
- CE R2
- CE 2007 Updates 11th and 12th month {QFES}
- CE 2008 Updates
- A running image from Lab 2-1

Exercise 1 Virtual memory leak

QA is testing the functionality of some utilities that were written for your new CE 6.0 device. The utilities initially appeared to work correctly, however during the QA phase one application continues to lock up the system. A memory leak is suspected. You have been asked to take a look at the utility to identify the problem and solve it.

- > Add the LeakingMemory subproject to your development environment
 - Copy the LeakingMemory folder from the Lab 3-3 Project Files from the Student files into your OS Design folder at C:\WINCE600\OSDesigns\EVMOSDesign\EVMOSDesign.
 - 2. In the Solution Explorer window right click on the **Subprojects** folder and select **Add Existing Subproject...**
 - 3. Navigate to the LeakingMemory folder that you just copied.
 - 4. Select LeakingMemory.pbpxml and click Open. Visual Studio will add the project to your current solution.

🛷 EVM_3530 (Running) - Microsoft Visual Studio								- - X
File Edit View	Project Build	Debug ⁻	arget Data	Tools Window	Community	Help		
1 - 🖼 - 🚰 🖢	Open						? 🗙	
	Lo	ok in: 🛅 S	ampleOSDesig	n	~	G 🕸 🖻 🛙		
Device: CE Device		☐ EV	4 3530					
Solution Explorer - So	MuBece		kingMemory					
EVM_3530	Documer	nts						
E:/WINCE600 E:/WINCE600 E://WINCE600 E:///WINCE600 E:///WINCE600 E://WINCE600 E://WINCE600 E://WINCE600 E://WINCE600 E://WINCE600 E://	Desktop	0						
Parameter Files Files SDKs Subprojects LeakingMemory	My Docum	ents						
Include files Include files Include files Parameter file Besource file Source files Source files	My Compu	uter						
postiink.bat		File na	me:			~	Open	
StdAfx.cpp	My Netwo	ork Files o	type:	Windows Embedded	CE Subproject	Files (.pbp: 💙	Cancel	
<	>							
< Solution Expl	Class View							
Output								→ ₽ ×
Show output from: W Ready	indows CE Deb	ua	• R		7			
🦺 start 🔰	🐢 EVM	🛄 Tera	🛞 Micro.	💽 Wind	🔁 4 W	🕞 🔮 Lab	Micro	C 🔒 🕲 3:49 PM

- 5. Configure the LeakingMemory subproject to be excluded from the image and always build and link as debug, just as you did in Lab 2-2 Exercise 1.
- 6. Right click the LeakingMemory subproject and select Build.
- Investigate the memory leak
 - 7. Run the application by selecting **Run Programs...** in **Target** menu.
 - 8. Observe the output on your device. The following message box appears:

LeakingMemory OK	×
Utility START	

Note This dialog box is used to stop the application from continuing without actually breaking into the debugger, thereby allowing us to use the memory tools.

This particular dialog indicates a point in the initialization sequence of the utility prior to any memory reservations. It allows us to get a picture of the initial memory usage that we can use as a basis for future comparisons.

- 9. Select Target | Target Control from the Visual Studio menu. The Windows CE Command Prompt window will open.
- 10. At the **Windows CE>** prompt, type **mi full**. This will show memory information for all processes running on the device.
- 11. Locate the section for the LeakingMemory.exe process.
- 12. Study the virtual memory contents of the process using the following legend:

Character	Definition
<blank></blank>	A blank space indicates a virtual page that is not currently allocated. Does not require a physical page.
-	Reserved but not in use. Indicates a virtual page that is currently allocated but not mapped to any physical memory. Does not require a physical page.
С	Code pages in ROM. Does not require a physical page.
С	Code pages in RAM. Requires a physical page.
S	Indicates a virtual page that holds a stack. Requires a physical page.
Р	Peripheral memory (pages used to map target device memory by using VirtualAlloc). Indicates a virtual page that is used to map a range of hardware addresses. Does not require a physical page. Peripheral memory may include frame buffer memory.
W	Indicates a virtual page that holds read-write data. Requires a physical page. Read-write pages include global variables as well as dynamically allocated memory.
0	Indicates a virtual page that is used by the object store. Requires a physical page. Should only appear in the Filesys process.
?	Contents unknown.

Character	Definition
r	Read-only data pages in RAM. Requires a physical page. Read-only data primarily comes from data items that are declared as a const type in the source code.
R	Read-only data pages in ROM. Does not require a physical page. Read-only data primarily comes from data items that are declared as a const type in the source code.
	executable code pages in hardware, use R(r) to represent both data and code.

- 13. Copy the memory information for the LeakingMemory process to a temporary text file so that we can easily compare it with another run later.
- 14. In the device window, click **OK** in the message box. The utility will run, doing its useful work.
- 15. The following message box will display on the device:



- **Note** This dialog box indicates the end of the utility processing. Presumably any resources that have been allocated have now been freed, and there should be no memory leaks.
 - 16. At the **Windows CE>** prompt, type **mi full**. This will show the current memory information for all processes running on the device.
 - 17. Compare the memory usage for the LeakingMemory.exe program against the one that was previously saved.
 - 18. Click OK on the dialog box on the device, allowing it to exit.

> Analysis

The application should have similar memory usage after it has performed its useful work and cleaned up as it did before it started. The two memory dumps should be the same.

Before	After				
Memory usage for Process	Memory usage for Process				
'LeakingMemory.exe' pid 5e0000e	'LeakingMemory.exe' pid 5e0000e				
0000000:	0000000:				
00010000: -cWc	00010000: -cWc				
00020000:S	00020000:S				
00030000: W	00030000: W				
00040000: RRRRRRRRRRRRRRRR	00040000: RRRRRRRRRRRRRRRR				
00050000: RRRRRRRRRRRRRRRR	00050000: RRRRRRRRRRRRRRRR				
00060000: RRRRRRRRRRRRRRRR	00060000: RRRRRRRRRRRRRRRR				
00070000: RRRR	00070000: RRRR				
	00080000:				
	00090000: ЖЖЖЖЖЖЖЖЖЖЖЖЖЖ				
40000000:	40000000:				
40010000: -CCCCCCCCCCCCC	40010000: -CCCCCCCCCCCCC				
40020000: CCCCCCCCCCCCCC	40020000: CCCCCCCCCCCCCC				
40030000: CCCCCCCCCCCCCC	40030000: CCCCCCCCCCCCCC				
40040000: CCCCCCCCCCCCCC	40040000: CCCCCCCCCCCCCC				
40050000: CCCCCCCCCCCCCC	40050000: CCCCCCCCCCCCCC				
40060000: CCCCCCCCCCCCCC	40060000: CCCCCCCCCCCCCC				
40070000: CCCCCCCCCCCCCC	40070000: CCCCCCCCCCCCCC				
40080000: CCCCCCCCCCCCCC	40080000: CCCCCCCCCCCCCC				
40090000: WCCCCCCC	40090000: WCCCCCCC				
400a0000:	400a0000:				
Page summary: code=134(2) data r/o=52 r/w=3 stack=1 reserved=63	Page summary: code=134(2) data r/o=52 r/w=35 stack=1 reserved=63				

However, the output of the Target Control utility shows that there is 128KB of read/write memory committed after the utility has finished that wasn't there before it started. That memory was allocated by the utility, but never released. That memory has been leaked.

> Fix the application

- 19. Uncomment the call to **VirtualFree** in the file **LeakingMemory.cpp**. This looks suspiciously like it might be the cause of the problem.
- 20. Right click on the LeakingMemory subproject and select Build.
- 21. Run the program again, and redo the analysis.
- 22. Observe that the two memory usage maps are now the same; the memory leak is gone!

Note This particular leak was not all that serious and created just to illustrate the point. The memory would have been reclaimed automatically when the process exited. However, in other scenarios the leak could have been more serious. For example, a kernel mode dll could have a leak that would never be recovered since the kernel process never exits.

If you are continuing with the next Hands-On Lab, keep your image running.

Lab 3-4: Exploring Threads Using Kernel Tracker

Objectives

- Learn which build options are required to work with the Remote Kernel Tracker
- Be familiar with the Kernel Tracker menu options and what they control, such as the time scale
- Recognize different execution patterns in Kernel Tracker such as when threads start and stop running.

Prerequisites

- Completed Lab 2-1
- Completed Lab 3-2

Estimated time to complete this lab: 30 minutes

Lab Setup

To complete this lab, you must have:

- A development workstation running Windows XP
- Visual Studio 2005 (Version 8) with Platform Builder plug-in
- CE 6.0
- CE 6.0 2006 Roll up
- CE 6.0 Service Pack 1
- CE 2007 Updates upto month 9th month {QFEs}
- CE R2
- CE 2007 Updates 11th and 12th month {QFES}
- CE 2008 Updates
- A running image from Lab 2-1

Exercise 1 Using the Remote Kernel Tracker

In this exercise you will become familiar with the Remote Kernel Tracker tool. The Remote Kernel Tracker allows you to see the kernel events that occur on your Windows Embedded CE6.0 device.

- Launch the Remote Kernel Tracker
 - 1. Select Target | Remote Tools | Kernel Tracker from the Visual Studio menu.
 - 2. Click **OK** to accept the **Default Device** connection.

Select a Windows CE Device							
Windows CE Default Platform							
OK Cancel							
<u>Sit</u>							

3. Click **Lie** to toggle Show/Hide **Legend** mode as desired.

🖬 Windows CE Remote Kern	nel Tra	icke r											
<u>File E</u> dit <u>V</u> iew <u>C</u> onnection <u>H</u> e	elp												
to 🐓 🏶 🖻 🖬 🔯	🔍 d	1 51	🕫 🖶	Q Q	🕈 Za	oom Range (ms):	10,0	00 🗾]					
Time (seconds) from 0:09:21.870000	<u> </u>		0:09:23. . I .	0:09:24 I	н. 	0:09:25. I .		0:09:26. I .	Show/Hide Leger	nd .	0:09:28. I .	D:09:29.	
■ Edd << 50x000000000000000000000000000000000													
CEMGRC.EXE <<0x054E006E> CEPWCLI.EXE <<0x07980012> MemoryLogger.exe <<0x07A80 RTH.EXE <<0x06C2001A>>	Ē				-								_
Shows/Hides Legend	<												

Events occurring on the CE Device are continuously being logged. First events are logged into local memory. CeLogFlush.exe periodically transmits the logged data to the host via the Platform Manager connection. The Remote Kernel Tracker tool displays the logged information in graphical form.

- 4. Click to **refresh** the logging data being displayed and note the time frame increasing as new data is being added to the right.
- 5. Click to search for an event by type or by a particular process or thread, etc.
- 6. Expand the Interrupts node.
- 7. Set the Zoom range to 10ms using the Zoom Range drop down box.

Note We enabled the profiling option back in Lab 2-1. This option is what allows interrupts to be observable in the Remote Kernel Profiler. The profiler option also provides support for a Monte Carlo profiler that can be run using the Target Control utility.

Exercise 2 Logging a Simple Application

Before looking at the complex relationships between multiple threads, we will look at the logging data being generated by a simple application. There is a great deal of data for even a simple application that only has one thread.

Ensure no breakpoints are set

- 1. Select **Debug** | **Windows** | **Breakpoint** from the Visual Studio menu. This will bring up the Breakpoints window.
- 2. Delete any breakpoints that might have been left in source code files.

Breakpoints	X
)ō 🗙 🏀 🍭 🖣 🎫 🕷 🎽	
🔽 at '{,HeapTest1.cpp,,,,} .15'	

Examine the HeapTest1 application

3. Launch the **HeapTest1** application using the Visual Studio **Target** menu.

Note Moving the cursor over events in the Remote Kernel Tracker displays info tips showing more detail regarding the event.

- 4. Position the mouse over an icon, in the HeapTest1.exe process, for a few seconds to see a pop-up for more details on the Load module event. Use the Find Event tool if necessary to find a Load Module icon.
- 5. Position the mouse over a dark green line for a few seconds to see a pop-up for more details about **Process Info**.
- 6. Click on an event to set the cursor to assist in zooming and stepping through events and threads using menu buttons.
- 7. Change the Zoom Range to view thread transitions in more detail.
- 8. Observe the calls to **Sleep** that occur every 1000 milliseconds in the HeapTest1 application. Note that the application still has a loop at the end that keeps it from exiting automatically.

- 9. Terminate the **HeapTest1** application using the Processes window available from the Visual Studio menu.
- 10. Observe the **Free Module** event that occurs on the HeapTest1 application in the Remote Kernel Tracker.

Take some time to explore the information provided by the kernel tracker to see how it shows the thread transitions and the reasons for them.

11. Close the **Remote Kernel Tracker.** Do not save the collected data.

6 Lab 3-4 Exploring Threads Using Kernel Tracker

Lab 3-5: Thread Synchronization

Objectives

- Understand the read / modify / write vulnerability
- Be able to implement an atomic read / modify / write sequence using a critical section that removes the vulnerability

Prerequisites

• Completed Lab 2-1

Estimated time to complete this lab: 15 minutes

Lab Setup

To complete this lab, you must have:

- A development workstation running Windows XP
- Visual Studio 2005 (Version 8) with Platform Builder plug-in
- CE 6.0
- CE 6.0 2006 Roll up
- CE 6.0 Service Pack 1
- CE 2007 Updates upto month 9th month {QFEs}
- CE R2
- CE 2007 Updates 11th and 12th month {QFES}
- CE 2008 Updates
- A running image from Lab 2-1

Exercise 1 Observe the vulnerability

This sample demonstrates a read / modify / write vulnerability. Multiple threads increment a shared global variable by

- reading the current global variable value into a temporary variable (READ)
- adding 1 to the value in the temporary variable (MODIFY)
- writing the temporary variable back to the global variable (WRITE)

Each thread iterates over this sequence a set number of times. The final value of the variable should be the number of threads multiplied by the number of iterations each thread makes.

- > Add the existing ThreadSynchronization subproject to the OS Design
 - 1. Copy the **ThreadSynchronization** subproject from the Student files to your OS Design at C:\WINCE600\OSDesigns\EVMOSDesign\EVMOSDesign.
 - 2. Right click on the **Subprojects** node in the Solution Explorer and select **Add Existing Subproject.**
 - 3. Select the **ThreadSynchronization.pbpxml** file from the **ThreadSynchronization** folder.
 - 4. Configure the **ThreadSynchronization** subproject to be **excluded from the image** and **always build and link as debug**, as documented in Lab 2-2.

Build and run the application

5. Right click on the **ThreadSynchronization** subproject in the Solution Explorer and select **Build**.



6. Launch **ThreadSynchronization.exe** using **Target** | **Run Programs**... from the Visual Studio menu.

7. Observe **output** similar to the following:

🐲 EVM_3530 (Running) - Microsoft Visual Studio	. 7 🗙
File Edit View Project Build Debug Target Data Tools Window Community Help	
1 - 図 - 図 - 図 - 図 - 図 - 図 - 図 - 図 - 図 -	
j 🕨 🖬 🖬 🖕 🚝 🗍 🛬 📕 Hex 🗔 🗸 🌉 i 🔛 🔹 🚽 💷 🔹 🚽	
Device: CE Device 🗸 🖓 😓 🖭 😭 😓	
Solution Explorer - So 🗸 🖡 🗙	
Solution Expl 🐼 Class View	
Output	→ ₽ ×
Show output from: Windows CE Debug 🔹 🔖 🎝 🖓 🗐	_
adSynchronization	~
tion 16:00:40 09/22/2008 Pacific Daylight Time	
nization 16:00:40 09/22/2008 Pacific Daylight Time	
TID:5f30016 THREADSYNCHRONIZATION: ThreadCount = 2	
TID:5f30016 THREADSYNCHRONIZATION: Expected Total = 20000000 TID:5f30016 THREADSYNCHRONIZATION: Actual Total = 14612635	
	~
	>
Build succeeded Ln 9 Col 1 Ch 1	INS
🛃 start 🛛 🐢 EVM 🛄 Tera 💿 Micro 🖡 Wind 🗀 5 W 🗸 🗿 Micro 📳 Lab 🔇 👫 💿	4:00 PM

8. Notice that the final total is **less** than the expected total.

Analysis

The final value is less than expected because the read / modify / write sequence is not atomic. The scheduler can interrupt a thread after it has performed the read, and run another thread that is performing the same algorithm. The second thread continues to increment the variable from the same value where the original thread stopped. When the original thread is eventually scheduled again it continues where it left off, and writes the value stored in the temporary register out to the global variable. In so doing, it destroys the work that was done on the variable by other threads while it was blocked. Each thread ran the prescribed number of times, but some of the work was inadvertently reset.

This vulnerability exists any time a variable shared between multiple threads is accessed with a non-atomic read/modify/write sequence.

Exercise 2 Fix the vulnerability

The read / modify / write sequence can be made atomic using synchronization objects. In this exercise we will use a critical section, although mutexes could be used as well. Note that the best way to protect an increment of a single variable is to use an interlocked function. However, we have used functions to do our work so the entire sequence must be protected.

Instantiate the critical section

- 9. Open the ThreadSynchronization.cpp file from the Solution Explorer.
- 10. Uncomment the global variable MyCritSec.

Initialize the critical section

11. Uncomment the call to InitializeCriticalSection in the function WinMain.

> Protect the vulnerable code sequence

- 12. Uncomment the call to EnterCriticalSection in the function DoWork.
- 13. Uncomment the call to LeaveCriticalSection in the function DoWork.

> Clean up resources

14. Uncomment the call to **DeleteCriticalSection** in the function **WinMain**.

Build and run the application

- 15. Right click on the **ThreadSynchronization** subproject in the Solution Explorer and select **Build**.
- 16. Launch **ThreadSynchronization.exe** using **Target** | **Run Programs** from the Visual Studio menu.



17. Observe correct **output** similar to the following:

If you are continuing with the next Hands-On Lab, keep your image running.

Lab 3-6: Exploring Synchronization Objects

Objectives

- Explain the different types of synchronization available in Windows CE
- Understand differences among synchronization objects.

Prerequisites

• Completed Lab 2-1

Estimated time to complete this lab: 40 minutes

Lab Setup

To complete this lab, you must have:

- A development workstation running Windows XP
- Visual Studio 2005 (Version 8) with Platform Builder plug-in
- CE 6.0
- CE 6.0 2006 Roll up
- CE 6.0 Service Pack 1
- CE 2007 Updates upto month 9th month {QFEs}
- CE R2
- CE 2007 Updates 11th and 12th month {QFES}
- CE 2008 Updates
- A running image from Lab 2-1

Exercise 1 Mutex synchronization

In this exercise you will see a simple implementation of synchronization using mutexes.

- > Add the existing MutexDemo subproject to the OS Design
 - 1. Copy the **MutexDemo** subproject from the Student files to your OS Design at C:\WINCE600\OSDesigns\EVMOSDesign\EVMOSDesign.
 - 2. Right click on the **Subprojects** node in the Solution Explorer and select **Add Existing Subproject.**
 - 3. Select the MutexDemo.pbpxml file from the MutexDemo folder.
 - 4. Configure the **MutexDemo** subproject to be **excluded from the image** and **always build and link as debug**, as documented in Lab 2-2.
- Build MutexDemo subproject

6.

5. Right click the MutexDemo subproject in the Solution Explorer and select Build.

🥗 EVM_3530 (Build		dio 📒	
File Edit Viev		Rebuild		Data Tools Window Community Help	
1 🖬 - 🛅 - 🞽	đ	Open		· 📮 - 🖳 🕨 🔹 🔹 💌	
i 🕨 🖬 🖬		Open With		- 6,月4.3 6,月3.4.3 - 6,日3.4. 年	÷
Device: CE Devic		Add	•		
Solution Explorer -		Set Subproje	ect Build Order		• ×
	2	Find in Files.		▼	~
	<u> (</u>	Explore		mo.cpp : Kernel Mutex demo	_
Favorites		Show in Fav	orites		
Parameter Files	<u>C:X</u>	Open Build V	Vindow	'stdafx.h"	_
Subprojects	\times	Remove			
LeakingMemor	L.	Properties		n/MutayDama/saussac)	
E Include file	:,, :S		DWORD GL	ign/hutes/bend/sources/ Jata;	
표 📄 Parameter	files	=	HANDLE hM	lutex;	
Resource l	iles s		L		
Mutex	- Demo	.cpp	🖃 DWORD Thr	ead1Proc(LPVOID lpv)	
💿 💽 postlink.ba	t		{		
prelink.bat EadMe to	/F		DWORD) dwThreadData;	
StdAfx.cp	5	✓	while	(TRUE)	
<	_		{		~
Solution Expl	1	Class View 📘	()		>
Output					↓ ₽ ×
Show output from:	Wir	ndows CE Deb	uq	- Ŗ 🚑 🛼 🛒 🖃	
🔄 Output [🔂 Erro	or List				
	_				
🦺 start	4	🌮 EVM	🛄 Tera	🛞 Micro 🖡 Wind 🗀 4 W 🔹 🐴 Micro 🕎 2 Mi 👻 🥵 🚼	4:23 PM
> Run application

7. Launch MutexDemo.exe using Target | Run Programs... from the Visual Studio menu.



8. Select **Windows CE Debug** from the drop down box in the Output window. This will allow us to see the debug output from the device when we run our test application.

4 Lab 3-6 Exploring Synchronization Objects

🏶 EVM_3530 (Running) - Microsoft Visual Studio	J 🗙
File Edit View Project Build Debug Target Data Tools Window Community Help	
i 🗊 = 🗃 = 💕 🛃 🗿 🐰 ங 🛍 🔊 = (* = 🚚 = 🖳) 🔹 🔹 👘	
i 🕨 🖬 🖬 🗢 역 🖟 앱 Hexl 🗔 - 🖕 i 🔜 🐨 👘 副 🕰 🕰 🖕 i 📑 筆	
Device: CE Device 🗸 🖓 😓 📴 🔛	
Solution Explorer - So • 4 X MutexDemo.cpp	• ×
(Global Scope)	~
PRIVATE	
Proventes	
#include "stdafx.h"	~
Solution Expl Constraints View	>
Output	Ψ×
Show output from: Windows CE Debug 🔹 🔖 🎝 🎝 🗐	
4070388 PID: 469001a TID: 4720012 1 1F0	~
4070439 PID:469001a TID:473001a 2 1F1	_
4070741 PID:469001a TID:473001a 2 1F2	
4071043 PID:469001a TID:473001a 2 1F3	
40/1313 PD:469001a TD:466001a IF3	
40/1345 FL-465001a TLD-4750012 2 TF4	
4073441 PID: 469001a TID: 473001a 2 1F6	
4073743 PID:469001a TID:473001a 2 1F7	
4074045 PID:469001a TID:473001a 2 1F8	
4074347 PID:469001a TID:473001a 2 1F9	_
	~
<u><</u>	>
🔄 Output 🖧 Error List	
Build succeeded Ln 541 Col 1 Ch 1	INS
🛃 start 🔰 🐢 EVM 🖳 Tera 💿 Micro 🕞 Wind 🗀 4 W 🖉 Micro 🕎 2 Mi 🔇 👫 4:2	8 PM

- **Note** The Output window is currently displaying the build output because we just performed a build. In many circumstances, Visual Studio anticipates what we would like to see and switches the Output window appropriately. However, this does not always work, and the Output window ends up displaying something other than the operation we are interested in. The following step is one of those that Visual Studio does not anticipate.
 - 9. Select **Target | Target Control** from the Visual Studio menu to bring up the **Windows CE Command Prompt** window (the Target Control utility). You may dock this window in a convenient location if you wish.
- **Note** We are running the application using the Target Control utility this time, instead of using the **Target** | **Run Programs** menu in Visual Studio. The Target Control utility is a lower level interface into the debug shell supported by Windows Embedded CE 6.0, and exposes a great deal of functionality. The Target | Run Programs menu item leverages the same debug shell functionality to launch programs as does the Target Control utility.
 - 10. Type **s MutexDemo** command into **Windows CE Command Prompt** window as follows:

Windows CE>s MutexDemo



11. Press **<Enter>** and verify debug output is similar to the following sequence:

3771174	PID:469001a	TID:46e001a	Priority	= 251
3771174	PID:469001a	TID:46e001a		0
3773184	PID:469001a	TID:4720012	1	1
3773235	PID:469001a	TID:473001a	2	2
3773537	PID:469001a	TID:473001a	2	3
3773839	PID:469001a	TID:473001a	2	4
3774141	PID:469001a	TID:473001a	2	5
3776186	PID:469001a	TID:4720012	1	6
3776237	PID:469001a	TID:473001a	2	7
3776539	PID:469001a	TID:473001a	2	8
3776841	PID:469001a	TID:473001a	2	9
3777143	PID:469001a	TID:473001a	2	A
3779188	PID:469001a	TID:4720012	1	В
3779239	PID:469001a	TID:473001a	2	С
3779541	PID:469001a	TID:473001a	2	D
3779843	PID:469001a	TID:473001a	2	E
3780145	PID:469001a	TID:473001a	2	F
3781185	PID:469001a	TID:46e001a		F
3782190	PID:469001a	TID:4720012	1	10
3782241	PID:469001a	TID:473001a	2	11
3782543	PID:469001a	TID:473001a	2	12
3782845	PID:469001a	TID:473001a	2	13
3783147	PID:469001a	TID:473001a	2	14
3785192	PID:469001a	TID:4720012	1	15
3785243	PID:469001a	TID:473001a	2	16
3785545	PID:469001a	TID:473001a	2	17
3785847	PID:469001a	TID:473001a	2	18
3786149	PID:469001a	TID:473001a	2	19
3788194	PID:469001a	TID:4720012	1	1A
3788245	PID:469001a	TID:473001a	2	1B
3788547	PID:469001a	TID:473001a	2	1C
3788849	PID:469001a	TID:473001a	2	1D
3789151	PID:469001a	TID:473001a	2	1E
3791186	PID:469001a	TID:46e001a		1E
3791196	PID:469001a	TID:4720012	1	1F
3791247	PID:469001a	TID:473001a	2	20
3791549	PID:469001a	TID:473001a	2	21
3791851	PID:469001a	TID:473001a	2	22
3792153	PID:469001a	TID:473001a	2	23

- 12. Launch the **Remote Kernel Tracker** to observe interaction between threads that are using the mutex. Try to correlate the events you see with the debug output and the source code.
- **Tip** Expand the MutexDemo node in the left hand pane to see the activity on the individual threads running in the process. Set the cursor on a particular area of thread activity, then change the zoom range to something small (10 milliseconds). This way, you can see everything that is going on in the thread.

> Terminate the application using the Target Control utility

- 13. Type **gi proc** in the Windows CE Command Prompt window. This will display a list of processes, including the name and an identification number for each one.
- 14. Determine the process identifier number for the mutexdemo application. The identifier for the mutexdemo in the dialog shown below is **09**, your's may differ.

```
Windows CE Command Prompt (Alt-1)
Windows CE Command Prompt
    <command>: Shell commands ('?' for she]
     .<command>': Debugger commands ('.?' 1
    '!<command>': Debugger extension commar
    Ctrl-Q: Abort pending command
    Ctrl-L: Clear all
    Ctrl-A: Select all
    Ctrl-F: Find (F4: Search forward, Shift
Windows CE>s mutexdemo
Windows CE>qi proc
                      hProcess: CurAKY :dwl
PROC: Name
P00: NK.EXE
                      00400002 00000000 801
P01: shell.exe
                      00ef0002 00000000 001
P02: udevice.exe
                      019a0002 00000000 001
P03: udevice.exe
                      01f30002 00000000 001
P04: udevice.exe
                      00ee0006 00000000 001
                      03d50002 00000000 001
P05: udevice.exe
P06: explorer.exe
                      044b0002 00000000 001
P07: EmulatorStub.exe 04530002 00000000 01
P08: servicesd.exe
                      045e0002 00000000 001
P09: mutexdemo.exe
                      05a70006 00000000 001
Windows CE>_
                                          >
```

15. Terminate the process using the command **kp<space><id>**, where <id> is the process identifier returned from the gi proc command.

Windows CE Command Prompt	(Alt-1)				×
P14: MutexDemo.exe	0487001a	00000000	00010000	00000000	^
Windows CE>kp 13					_
Attempting to kill pro	cess of :	id 046900 ⁻	laSuco	eeded:	
Windows CE>gi proc					
PROC: Name	hProcess	: CurAKY :	:dwVMBase:	CurZone	
POO: NK.EXE	00400002	00000000	84001000	00000000	
P01: shell.exe	01ba0002	00000000	00010000	00000000	
P02: udevice.exe	01e70002	00000000	00010000	00000000	
P03: udevice.exe	00580006	00000000	00010000	00000000	
P04: udevice.exe	03e90002	00000000	00010000	00000000	
P05: explorer.exe	048c0002	00000000	00010000	00000000	
P06: servicesd.exe	04f20002	00000000	00010000	00000000	
P07: repllog.exe	02b4000a	00000000	00010000	00000000	
P08: rapisrv.exe	058a0002	00000000	00010000	00000000	
P09: rnaapp.exe	05b70002	00000000	00010000	00000000	-
P10: udp2tcp.exe	05 03 0 0 06	00000000	00010000	00000000	
P11: CEMGRC.EXE	04e5000a	00000000	00010000	00000000	
P12: Clientside.exe	055b000a	00000000	00010000	00000000	
P13: MutexDemo.exe	0487001a	00000000	00010000	00000000	=
Windows CE>kp 13					
Attempting to kill pro	cess of :	id 048700 ⁻	laSuco	ceded	
· - ·					
Windows CE>					~
<				>	

> Measure synchronization performance

16. Type **s osbench** –**t 3** and press <Enter> at the **Windows CE Command Prompt**. This will run the OSBench utility that tests the performance of various kernel API calls. This particular command line will limit the testing to mutexes only.

Note The command line parameters for OSbench can be obtained by using the **-h** command line parameter to the OSBench utility.

17. Examine the **OSBench** output. You may wish to compare the performance of mutexes to other synchronization methods.

8 Lab 3-6 Exploring Synchronization Objects

🏶 EVM_3530 (Running) - Microsoft Visual Studio				_ 7 🗙
File Edit View Project Build Debug Target Data Tools W	indow Community He	р		
👔 - 🗐 - 😰 🛃 🙏 🖬 🏨 🔊 - 🝽 - 🗐 -	*		· 🖄	
i 🕨 💷 🖬 💠 🖓 🗐 📮 🧤 🗛 🗸 🙀	- D.	A 4 5	🔝 🗟 🖕 🔺	律律 🖞
Device: CE Device 🔹 🗣 😓 🖭 😭 🚽				
Solution Explorer - So 4 × MutexDemo.cpp				• X
Solution Expl., Class View (Global Scope)	*			*
Output				— П X
Show output from: Windows CE Debug	I			V T A
4910200 RTD: 400002 TTD: 4000010 RVIETD: Opening file of	hangh ang firan dag	let an		
4910312 PID:490002 11D:4900010 #ALFSD: opening 1110 05	t 3)	kcop		<u>^</u>
4911262 PID:400002 TID:49e001e RELFSD: Opening file os	bench dll.dll from	desktop		
4911387 PID:49d001e TID:49e001e OsBench -t 3	-	-		
4911537 PID:400002 TID:4d80016 RELFSD: Opening file os	bench.exe from des	ktop		
4912352 PID:4d70016 TID:4d80016 ++0SBench (Loader): (-	2)			
4912352 PID:4d70016 TID:4d80016 OsBench -2				
4912352 PID:49d001e TID:49e001e				
4912352 PID:49d001e TID:49e001e				
4912352 PID: 49d001e TID: 49e001e vvvvvvvvvvvvvvvvvvvvv	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~	~~~~~	
4912252 PID:49d001e TID:49e001e vvvvvvvvvvvvvvvvvvvvvv				
4912352 PID: 494001e TID: 49e001e	DETING THE OSBENCH	DFSIII.TS		
4912352 PTD: 49d001e TTD: 49e001e				
4912352 PID: 49d00le TID: 49e00le OSBench measures key	operating system	timings as cl	oselv as	
4912352 PID:49d001e TID:49e001e possible. All tests	use the QueryPerfo	rmanceCounter	(QPC) to get	
4912352 PID:49d001e TID:49e001e time stamps. The fre	quency of the coun	ter and the o	verhead of	
4912352 PID:49d001e TID:49e001e calling this function	n must be taken in	to account wh	en	-
4012052 DTD: 404001- TTD: 40-0011	-			
N Cutout 🖹 Error Liet				
			<i></i>	
Ready	Ln 1	Col 1	Ch 1	INS
🛃 start 🔰 🛷 EVM 🛄 Tera 🛞 Micro 💽	Wind 🛅 4 W	- 🙆 Micro	👿 2 Mi 👻 🔇	🔒 4:44 PM

🥙 EVM_3530 (Running) - Microsoft Visual Studio	Ξ×
File Edit View Project Build Debug Target Data Tools Window Community Help	
- -	
▶ 11 🖬 🖬 ◆ 9월 💭 📬 Hex 🗔 • 🛒	i
Device: CE Device 🗸 🖓 😓 🖾 😭	
Solution Explorer - So 4 × MutexDemo.cpp	• ×
(Global Scope)	
Solution Expl 22 Class View	
	- 4 ×
4912510 PID: $49d001e$ TID: $49e001e$ Max Time = 91.552 us	<u>^</u>
4312510 PID: 4340010 = TID: 4390010 = 1 min filme = 51.035 US	
4912510 PTD.494001a TTD.494001a - 64.11/ 45	
4912707 PTD-494001a TTD-494001a	
$4912747 \text{ PTD} \cdot 494001e \text{ TD} \cdot 49e001e \text{ I } 3.00 \text{ I } \text{TP} = YES \text{ I } CS = YES \text{ I } 1.75$	
4912747 PID: 494001e TID: 49e001e	
4912747 PID: 494001e TID: 49e001e Mutex inter-process :	
Time from a lower priority thread calling ReleaseMutex to a higher priority	
thread in a different process unblocking from a WaitForSingleObject call	
4912747 PID:49d001e TID:49e001e	
4912747 PID:49d00le TID:49e00le Max Time = 91.552 us	
4912747 PID:49d00le TID:49e00le Min Time = 30.517 us	
4912747 PID:49d00le TID:49e00le Avg Time = 65.967 us	
4912747 PID:49d00le TID:49e00le ==================================	
4912747 PID:4d70016 TID:4d80016OSBench (Loader): Done!	=
4913330 PID:49d00le TID:49e00leOSBench (Loader): Done!	
	×
	>
🔄 Output 📸 Error List	
Ready Ln 108 Col 1 Ch 1	INS
🛃 start 🛛 🛷 EVM 🖳 Tera 🛞 Micro 📴 Wind 🗀 4 W 🖉 Micro 👿 2 Mi 🔇 🔒 4	44 PM

Exercise 2 Event synchronization

In this exercise you will see a simple implementation of synchronization using events.

- > Add the existing EventDemo subproject to the OS Design
 - 1. Copy the **EventDemo** subproject from the Student files to your OS Design at C:\WINCE600\OSDesigns\EVMOSDesign\EVMOSDesign.
 - 2. Right click on the **Subprojects** node in the Solution Explorer and select **Add Existing Subproject.**
 - 3. Select the EvenDemo.pbpxml file from the EventDemo folder.
 - 4. Configure the **EventDemo** subproject to be **excluded from the image** and **always build and link as debug**, as documented in Lab 2-2.
- Build EventDemo subproject
 - 5. Right click the EventDemo subproject in the Solution Explorer and select Build.

> Run application

- 6. Select **Windows CE Debug** from the drop down box in the output window. This will allow us to see the debug output from the device when we run our test application.
- 7. Type **s EventDemo** into the **Windows CE Command Prompt** window as follows:

Windows CE>s EventDemo

8. Press **<Enter>** and verify debug output is similar to the following:

s EventDemo 17:14:10 09/22/2008 Pacific Daylight Time End s EventDemo 17:14:10 09/22/2008 Pacific Daylight Time 6786826 PID:49b002a TID:49d002a Primary = 251 6786826 PID:49b002a TID:49d002a Thread1 = 250 6786826 PID:49b002a TID:49d002a Thread2 = 249 6786826 PID:49b002a TID:49d002a -- 1 6788647 PID:49b002a TID:49d002a -- 2 6791559 PID:49b002a TID:49d002a -- 3 6791559 PID:49b002a TID:49d002a E1 auto 6791559 PID:49b002a TID:4d50022 T2 1 6793236 PID:49b002a TID:49d002a -- 4 6794827 PID:49b002a TID:49d002a -- 5 6794827 PID:49b002a TID:49d002a E2 manual 6794827 PID:49b002a TID:4d50022 T2 6796562 PID:49b002a TID:49d002a -- 6 6796562 PID:49b002a TID:49d002a E1 auto 6796562 PID:49b002a TID:49f002a T1

6798442	PID:49b002a	TID:49d002a		7
6800180	PID:49b002a	TID:49d002a		8
6801880	PTD:49b002a	TTD:49d002a		9
0001000	DID.400002a	TID.40002a	- 1	5
6801880	PID:49b002a	T1D:49d002a	ΕT	auto
6801882	PID:49b002a	TID:4d50022	Т2	3
6803839	PID:49b002a	TID:49d002a		10
6803839	PTD:49b002a	TTD:49d002a	E2	manual
6905630	DTD:/0b002a	TTD: 194002a		11
0005055	DID.400002a	TID.40002a		10
6806674	PID:49b002a	T1D:49d002a		12
6806674	PID:49b002a	TID:49d002a	Ε1	auto
6806674	PID:49b002a	TID:4d50022	т2	4
6808499	PTD:49b002a	TTD:49d002a		13
6910354	DTD:/0b002a	TTD: 194002a		1 /
0010334	FID.490002a	TID.490002a		14
6812098	PID:49b002a	T1D:49d002a		15
6812098	PID:49b002a	TID:49d002a	Ε1	auto
6812098	PID:49b002a	TID:4d50022	т2	5
6812098	PTD:49b002a	TTD:49d002a	E2	manual
6913029	DTD:/0b002a	TTD: 194002a		16
0013920	PID:490002a	11D:490002a		10
68T26A8	PID:49b002a	TID:49d002a		Τ/
6817232	PID:49b002a	TID:49d002a		18
6817232	PID:49b002a	TID:49d002a	Ε1	auto
6817232	PTD • 49b002a	TTD·4d50022	т2	6
6010000	DTD: 10b002a	TID: 1000022	12	10
0010900	PID:490002a	11D:490002a		19
6820885	PID:49b002a	TID:49d002a		20
6820885	PID:49b002a	TID:49d002a	E2	manual
6820885	PID:49b002a	TID:4d50022	т2	7
6822497	PTD • 496002a	TTD·49d002a		21
0022407	DTD: 40b002a	mtp. 40-1002a	m 1	2 I .
0822497	PID:490002a	T1D:490002a	БI	auto
6822497	PID:49b002a	TID:49±002a	ΤΊ	2
6824389	PID:49b002a	TID:49d002a		22
6826793	PID:49b002a	TID:49d002a		23
6828271	PTD:49b002a	TTD:49d002a		24
6020271	DTD: 40b002a	TID: 1940024	r: 1	
0020271	PID:490002a	11D:490002a	<u>с</u> т	auto
6828271	PID:49b002a	T1D:4d50022	ΤZ	8
6830076	PID:49b002a	TID:49d002a		25
6830076	PID:49b002a	TID:49d002a	E2	manual
6831919	PTD:49b002a	TTD:49d002a		2.6
6833771	PTD:49b002a	TTD:49d002a		27
0000771	DTD: 10b002a	mtD: 10d002d	m 1	
0033771	PID:490002a	11D:490002a	<u>с</u> т	auto
6833771	PID:49b002a	T1D:4d50022	Τ2	9
6835713	PID:49b002a	TID:49d002a		28
6837640	PID:49b002a	TID:49d002a		29
6839533	PTD:49b002a	TTD:49d002a		30
6839533	PTD:/Qb002a	TTD://0/002a	F 1	211+0
00000000	DID.400002a	TID.4J0002a	ш <u>т</u>	10
6839533	PID:49b002a	T1D:4050022	ΤΖ	10
6839533	PID:49b002a	TID:49d002a	E2	manual
6841163	PID:49b002a	TID:49d002a		31
6842912	PTD:49b002a	TTD:49d002a		32
6911501	DTD:/0b002a	TTD: 194002a		33
001104	11D.10002d	TTD. TJUUUZA		J J J
0044504	FID:490002a	11D:490002a	타 I - ^	aulo
6844505	PID:49b002a	'I'ID:4d50022	Т2	11
6846250	PID:49b002a	TID:49d002a		34
6848938	PID:49b002a	TID:49d002a		35
6848938	PTD • 49h002a	TTD.4940029	E2	manual
0040000	DTD: 40b002a	TID.4JE0022	ш2 m 0	10
6848938	PID:49b002a	T1D:4050022	ΤΖ	12
6851540	PID:49b002a	TID:49d002a		36
6851540	PID:49b002a	TID:49d002a	Ε1	auto
6851540	PTD:49b002a	TTD:4d50022	т2	13
6853337	PTD · 49h002-	TTD.4940022		37 -2
COE 4000	LTD. 102002d	TTD. 104002d	-	20
0054992	riu:490002a	11D:490002a		30
6856724	PID:49b002a	'I'ID:49d002a		39
6856724	PID:49b002a	TID:49d002a	Ε1	auto
6856724	PID:49b002a	TID:4d50022	т2	14
6858360	PTD . 40h002-	TTD.404002-		40
6050303	LTD. 102002d	TTD. JOJ0024	- -	10 mon7
0000309	riu:490002a	11D:490002a	다.८	manua1
6859459	PID:49b002a	TID:49d002a		4⊥
6862226	PID:49b002a	TID:49d002a		42
6862226	PID:49b002a	TID:49d002a	Е1	auto
6862226	PTD:49b002a	TTD:4d50022	т2	1.5
6863017	DID. / Qb0020	TD. 1000022		13 10
0000394/	11D.43DUUZA	11D.490002d		L L
6865796	FID:49b002a	TID:49d002a		44

```
6867423 PID:49b002a TID:49d002a -- 45
6867423 PID:49b002a TID:49d002a E1 auto
6867423 PID:49b002a TID:4d50022 T2
6867423 PID:49b002a TID:49d002a E2 manual
6869214 PID:49b002a TID:49d002a -- 46
6870231 PID:49b002a TID:49d002a -- 47
6871994 PID:49b002a TID:49d002a -- 48
6871994 PID:49b002a TID:49d002a E1 auto
6871994 PID:49b002a TID:4d50022 T2
                                       17
6873860 PID:49b002a TID:49d002a -- 49
6875578 PID:49b002a TID:49d002a -- 50
6875578 PID:49b002a TID:49d002a E2 manual
6875578 PID:49b002a TID:4d50022 T2
                                       18
6877474 PID:49b002a TID:49d002a -- 50 shutdown
```

- 9. Use the **Remote Kernel Tracker** to observe interaction between threads that are using event objects. Try to correlate the events you see with the debug output and the source code.
- **Note** This application terminates automatically. If you see the shutdown message in the debug output, the application has already terminated. If Remote Kernel Tracker was still running from the last Exercise, you should still be able see the event information.

Measure synchronization performance

- 10. Type **s osbench –t 1** and press <Enter> at the **Windows CE Command Prompt**. This will run the OSBench utility that tests the performance of various kernel API calls. This particular command line will limit the testing to events only.
- 11. Examine the **OSBench** output. You may wish to compare the performance of events to other synchronization methods.

Exercise 3 Semaphore synchronization

In this exercise you will see a simple implementation of synchronization using semaphores.

- > Add the existing SemaphoreDemo subproject to the OS Design
 - 1. Copy the **SemaphoreDemo** subproject from the Student files to your OS Design at C:\WINCE600\OSDesigns\EVMOSDesign\EVMOSDesign.
 - 2. Right click on the **Subprojects** node in the Solution Explorer and select **Add Existing Subproject.**
 - 3. Select the SemaphoreDemo.pbpxml file from the SemaphoreDemo folder.
 - 4. Configure the **SemaphoreDemo** subproject to be **excluded from the image** and **always build and link as debug**, as documented in Lab 2-2.
- > Build SemaphoreDemo subproject
 - 5. Right click the **SemaphoreDemo** subproject in the Solution Explorer and select **Build.**

> Run application

- 6. Select **Windows CE Debug** from the drop down box in the output window. This will allow us to see the debug output from the device when we run our test application.
- 7. Type **s SemaphoreDemo** into the **Windows CE Command Prompt** window as follows:

Windows CE>s SemaphoreDemo

8. Press <Enter> and verify debug output is similar to the following:

 6887811
 PID:4f2000a
 TID:4f9000a
 1
 1

 6887838
 PID:4f2000a
 TID:4fa000a
 2
 2

 6887839
 PID:4f2000a
 TID:4fc000a
 3
 3

 6887943
 PID:4f2000a
 TID:4fa000a
 2
 4

 6887994
 PID:4f2000a
 TID:4fa000a
 2
 4

 6888015
 PID:4f2000a
 TID:4fa000a
 2
 6

 6888045
 PID:4f2000a
 TID:4fa000a
 2
 6

 6888046
 PID:4f2000a
 TID:4fa000a
 1
 7

 6888017
 PID:4f2000a
 TID:4fa000a
 2
 9

 6888168
 PID:4f2000a
 TID:4fa000a
 1
 10

 688219
 PID:4f2000a
 TID:4fa000a
 1
 11

 688246
 PID:4f2000a
 TID:4fa000a
 1
 13

 6888270
 PID:4f2000a
 TID:4fa000a
 1
 13

 6888297
 PID:4f2000a
 TID:4fa000a
 1
 13

 6888348
 PID:4f2000a
 TID:4fa000a
 1
 13

</tabr/>

6888399	PID:4f2000a	TID:4fa000a	2	\setminus	16	
6888451	PID:4f2000a	TID:4fc000a	3	\setminus	17	
6888471	PID:4f2000a	TID:4fa000a	2	\setminus	18	
6888502	PID:4f2000a	TID:4f9000a	1	\setminus	19	
6888522	PID:4f2000a	TID:4fc000a	3	\setminus	20	
6888573	PID:4f2000a	TID:4fa000a	2	\setminus	21	
6888626	PID:4f2000a	TID:4fc000a	3	\	22	
6888677	PID:4f2000a	TID:4fa000a	2	\	23	
6888706	PID:4f2000a	TID:4fc000a	3	\	24	
6888729	PID:4f2000a	TID:4f9000a	1	\setminus	25	
6888757	PID:4f2000a	TID:4fa000a	2	\setminus	26	
6888808	PID:4f2000a	TID:4fc000a	3	\	27	
6888885	PID:4f2000a	TID:4fa000a	2	\	28	
6888931	PID:4f2000a	TID:4fc000a	3	\	29	
6888939	PID:4f2000a	TID:4f9000a	1	\	30	
6888982	PID:4f2000a	TID:4fa000a	2	\setminus	31	
6889035	PID:4f2000a	TID:4fc000a	3	\setminus	32	
6889086	PID:4f2000a	TID:4fa000a	2	\setminus	33	
6889137	PID:4f2000a	TID:4fc000a	3	\	34	
6889140	PID:4f2000a	TID:4fa000a	2	\	35	
6889188	PID:4f2000a	TID:4f9000a	1	\	36	
6889191	PID:4f2000a	TID:4fc000a	3	\setminus	37	
6889242	PID:4f2000a	TID:4fa000a	2	\setminus	38	
6889293	PID:4f2000a	TID:4fc000a	3	\setminus	39	
6889344	PID:4f2000a	TID:4fa000a	2	\setminus	40	
6889389	PID:4f2000a	TID:4fc000a	3	\setminus	41	
6889395	PID:4f2000a	TID:4f9000a	1	\setminus	42	
6889440	PID:4f2000a	TID:4fa000a	2	\setminus	43	
6889491	PID:4f2000a	TID:4fc000a	3	\setminus	44	
6889542	PID:4f2000a	TID:4fa000a	2	\setminus	45	
6889593	PID:4f2000a	TID:4fc000a	3	\setminus	46	
6889596	PID:4f2000a	TID:4fa000a	2	\setminus	47	
6889644	PID:4f2000a	TID:4f9000a	1	\setminus	48	
6889649	PID:4f2000a	TID:4fc000a	3	\setminus	49	
6889701	PID:4f2000a	TID:4fa000a	2	\setminus	50	
6889752	PID:4f2000a	TID:4fc000a	3	\setminus	51	
6889758	PID:4f2000a	TID:4f3000a	Sł	nut	ting	down
6889818	PID:4f2000a	TID:4fa000a	2	\backslash	52	

9. Use the **Remote Kernel Tracker** to observe interaction between threads that are using semaphore objects. Try to correlate the events you see with the debug output and the source code.

. . .

Note This application terminates automatically. If you see the shutdown message in the debug output, the application has already terminated. If Remote Kernel Tracker was still running from the last Exercise, you should still be able see the event information.

Measure synchronization performance

- 10. Type **s osbench t 2** and press <Enter> at the **Windows CE Command Prompt**. This will run the OSBench utility that tests the performance of various kernel API calls. This particular command line will limit the testing to semaphores only.
- 11. Examine the **OSBench** output. You may wish to compare the performance of semaphores to other synchronization methods.

If you are continuing with the next Hands-On Lab, keep your image running.

Lab 4-1: Using the Remote Registry Editor

Objectives

• Use the Remote Registry Editor to explore and change the device registry.

Prerequisites

• Completed Lab 2-1

Estimated time to complete this lab: 20 minutes

Lab Setup

To complete this lab, you must have:

- A development workstation running Windows XP
- Visual Studio 2005 (Version 8) with Platform Builder plug-in
- A development workstation running Windows XP
- Visual Studio 2005 (Version 8) with Platform Builder plug-in
- CE 6.0
- CE 6.0 2006 Roll up
- CE 6.0 Service Pack 1
- CE 2007 Updates upto month 9th month {QFEs}
- CE R2
- CE 2007 Updates 11th and 12th month {QFES}
- CE 2008 Updates
- A running image from Lab 2-1

Exercise 1 Using the Remote Registry Editor

In this exercise, you will use the Windows CE Remote Registry Editor to examine and modify the registry on the target device.

- > Starting the Remote Registry Editor
 - 1. Select **Target** | **Remote Tools** | **Registry Editor** from the Visual Studio menu. Click **OK** to accept the **Default Device** connection.

Select a Windows CE Device						
Vindows CE Default Platform						
<u>D</u> K <u>C</u> ancel						

- 2. Wait a few seconds for the connection to be established and for required files to be transferred to the device.
- **Note** The Remote Registry Editor also displays the registry of your development workstation under the My Computer tree in the left hand pane. This tree will be available even if you are not connected to the target device. Make sure you don't get confused about which registry you are viewing and/or editing.

> Explore the target device registry

- 3. In the left-hand pane of the Windows CE Remote Registry Editor, right-click on **Default Device** and then click **Find**. This will bring up the Find dialog.
- 4. Type **NE20001** in the Find dialog and click **OK**.

🎎 Windows CE Remote Registr	y Editor							- - X
Registry Edit View Connection H	Help							
My Computer Key_CLASSES_ROOT HKEY_CLASSES_ROOT HKEY_CURRENT_USER HKEY_LOCAL_MACHINE KEY_LOCAL_MACHINE KEY_USERS	Name	Data						
	Finding registry	r items [ROOT\.aif]						
		[Cancel					
EVM_3530								NUM
🦺 start 🛛 🛷 E 🖉	🖁 т 🔕 м	💽 w	🛅 4 V 🕞	🖉 M	🚺 2 N 👻	🙀 W	🔮 L	C 🚼 5:56 PM

- 5. The [HKEY_LOCAL_MACHINE\Comm\NE20001] is displayed.
- 6. Expand the **NE20001** key and click on the **Parms** key.
- > Modify the device registry
 - 7. Right click the [HKEY_LOCAL_MACHINE\SOFTWARE] key and select New | Key.
 - 8. Create a key called GeneriCo and click OK.

4 Lab 4-1 Using the Remote Registry Editor



9. Right click on the new GeneriCo key and select New | String Value.

🎪 Windows CE Remote Registry	Editor		_ 7 🗙
Registry Edit View Connection He	lp		
	3		
HKEY_CLASSES_ROOT	Name	Data	
	(Default)	(value not set)	
Drivers32			
🗄 🧰 ExtModems			
I → → HARDWARE			
notify			
Services			
⊕ 📄 Apps			
Generico	L		
Hicros Expand			
E System New	Key	ey l	
Time Zope	Del	ring Value	
	Multi	ulti String Value	
Find	Ctrl+F Binar	nary Value	
EVM 3530\HKEY LOCAL Copy Key I	DWC	WORD Value	NUM
🐴 start 🛛 🛷 EV	Ter 🛞 Mic	💽 Wi 🗿 Mic 🚺 2 N. 👻 🎪 Wi 🖤 Lab 🌾	6:09 PM

- 10. Create the string value with the name **MyString**, and the value **MyStringValue**.
- 11. Click **OK** to create the value.

🎢 Windows CE Remote Regi	stry Edi	tor						_ 7 🗙
Registry Edit View Connection	Help							
		1 m m m D						
 HKEY_CLASSES_ROOT HKEY_CURRENT_USER HKEY_LOCAL_MACHINE AudioCompressionM Drivers Drivers32 Explorer ExtModems HARDWARE Ident init Loader MUI nls notify Services ShimEngine Snd SofTWARE GeneriCo Microsoft System TAPI Time Zones uiproxy 		ame](Default) ew String Value MyString /alue: MyStringValue.	Data (value not se	t)	Car	ncel		
[EVM_3530\HKEY_LOCAL_MACHINE	SOFTWAR	RE\GeneriCo]	_ 1	- 1				NUM
🦺 start 🛛 🥙 EV	🛄 Ter	r 🔞 Mic	💽 Wi	🍘 Mic	🗾 2 N. 🔫	🕵 Wi	💾 Lab	6:10 PM

12. Right click on the GeneriCo key and select New | Binary Value. This will bring up the New Data Value dialog.

🎪 Windows CE Remote Registry Ec	ditor				- 7 🛛
Registry Edit View Connection Help					
	1 💩 🛄 🧱 💟				
	Name	Data			
	벨(Default)	(value not set)			
	MyString	MyStringValue.			
E Comm					
🕀 🧰 Drivers					
Drivers32					
Explorer ExtModems					
Ident					
🗈 💼 init					
notify					
🕀 🧰 ОМАРРМХ					
E Services					
ShimEngine					
🕀 🧰 Apps					
Gen Expand					
🕀 🧰 System 🛛 New	Key				
🗈 🧰 TAPI 🛛 Delete	Del Stripg Valu				
Time Zol Rename	Multi String	g Value			
Find	Ctrl+F Binary Valu	ue			
	DWORD V	/alue			
	- In				NUM
🛛 Start 🖉 EV	er 🔞 Mic	💽 Wi 🛛 🕙 Mic	🕒 2 N. 👻 🏦 Wi	맫 Lab	6:10 PM

13. Create a binary value with the name MyBinaryValue. In the Value box, type 0123456789abcdef and click OK. Notice that the editor groups the entry into bytes, and adds an ASCII translation of the values in the right column of the value box.

🎢 Windows CE Remote Regi	y Editor	🔳 🗗 🗾
Registry Edit View Connection	lelp	
HKEY_CLASSES_ROOT HKEY_CURRENT_USER HKEY_LOCAL_MACHINE Drivers Drivers32 Explorer ExtModems HARDWARE Ident nit notify notify Services ShimEngine Snd SofTWARE Apps Mirrosoft	New Data Value Name: MyBinaryValue Value: 0000 01 0000 01 0008	g.«Iï
System TAPI Time Zones uiproxy	ОК	Cancel
[EVM_3530\HKEY_LOCAL_MACHINE]	FTWARE\GeneriCo]	NUM
🛃 start 🛛 🐼 EV	🛄 Ter 🔞 Mic 📳 Wi 🗿 Mic 🚺 2	N. 🔻 🎪 Wi 👜 Lab 🔇 🚼 6:12 PM

- **Note** You can also edit the ASCII translation area of this dialog box instead of typing in binary values. There is no visible differentiator between the two areas in the dialog, just click your mouse in the far right hand area if you wish to enter ASCII text.
 - 14. Right-click on MyBinaryValue and select Delete.
 - 15. Confirm your intention to delete by clicking Yes.



- 16. In the left-hand pane, right click on GeneriCo and select Rename.
- 17. Type GeneriCo2 and press Enter.
- Save a portion of the device registry to a file on the development workstation
 - 18. In the left-hand pane of the Remote Registry Editor, highlight the [HKLM\SOFTWARE\GeneriCo2] key.
 - 19. Select Registry | Export Registry File from the Remote Registry Editor menu.
 - 20. Save the file to your desktop as MyDeviceRegKey.txt.

WARNING It is possible to merge an exported Windows CE registry file into the registry of your desktop system. This can result in catastrophic corruption of the registry on your workstation. So be careful!

21. Open the MyDeviceRegKey.txt file using Visual Studio.

- 22. Observe that the format of the Windows CE registry file is identical to that used in other version of Microsoft Windows.
- 23. Close and delete MyDeviceRegKey.txt
- 24. Close the Remote Registry Editor.

Lab 4-2: Power Management

Objectives

- Introduce the Windows Embedded CE 6.0 power management architecture
- Utilize portions of the CE 6.0 Power Management architecture
- Become familiar with several Power Management APIs
- Allow a test application to receive notification about system power events and to put power requirements into place

Prerequisites

• Completed Lab 2-1

Estimated time to complete this lab: 30 minutes

Lab Setup

To complete this lab, you must have:

- A development workstation running Windows XP
- Visual Studio 2005 (Version 8) with Platform Builder plug-in
- CE 6.0
- CE 6.0 2006 Roll up
- CE 6.0 Service Pack 1
- CE 2007 Updates upto month 9th month {QFEs}
- CE R2
- CE 2007 Updates 11th and 12th month {QFES}
- CE 2008 Updates
- A running image from Lab 2-1

Exercise 1

This exercise demonstrates portions of the Windows Embedded CE 6.0 power management architecture. Several of the power management APIs are used, allowing the test application to receive notification about system power events and to put power requirements into place.

This lab will make use of the backlight driver to demonstrate the interaction between applications and drivers in the realm of power management. We will modify a portion of the existing backlight driver to better illustrate these concepts. We will rebuild the OS run-time image to include the modified backlight driver and the test application.

> Add the existing Power_Management subproject to the OS Design

- 1. Copy the **Power_Management** subproject from the Student files to your OS Design at C:\WINCE600\OSDesigns\EVMOSDesign\EVMOSDesign.
- 2. Right click on the **Subprojects** node in the Solution Explorer and select **Add Existing Subproject.**
- 3. Select the **Power_Management.pbpxml** file from the **Power_Management** folder.
- 4. This time, **do not** configure the subproject to be excluded from the image or built as debug. We will rebuild the OS run-time image including the modules from this subproject.

> Modify existing backlight driver

- 5. Expand the node C:/WINCE600 | PLATFORM | EVMBSP | src | drivers | backlight | MDD | Include Files in the Solution Explorer.
- 6. Double click the file **bkli.h** to open the file in the Visual Studio editor.
- 7. Locate the **#define ZONE_BACKLIGHT** near the top of the file, and set it to **1**. This will cause the backlight driver to print debug messages when implementing power management requests.

Examine registry changes

- 8. Locate the **Power_Management** subproject in the Solution Explorer.
- 9. Open the file Power_Management.reg in the Parameter files node.
- 10. Examine the registry entries under [HKLM\System\CurrentControlSet\Control\Power\State]. These registry

entries override the default behavior for the backlight driver in each of the named system power states. They cause the backlight driver to default to its low power state instead of operating in its normal full power state.

🏶 EVM_3530 - Microsoft Visual Studio				_ 7 🗙
File Edit View Project Build Debug Ta	arget Tools Window Comm	nunity Help		
🛅 • 🛅 • 💕 🛃 🗿 🗼 🗈 🛍 🔊 •	🝽 - 🚚 - 🖳 🕨 TI_EVM	_3530_ 🝷 Platform Builder (_TGTCF	PU) 🝷 🏄	
·	142 SA 📮 🖪 🗟 📐 AS	律 律 国 일 🗆 🖓 🛛	a 🗊 🖬 🚑 🔒	
Device: CE Device 🗸 🖓 😓 🖭 😭	-			
Solution Explorer - Solution 'EVM_3530' 👻 📮 🗙	Power_Management.reg	StdAfx.h gpio_backlight.cpp	tled_backlight.cpp	• ×
	.EY_CLASSES_ROOT	Name	Туре	Data
LeakingMemory (E:/WINCE600/OSI MutexDemo (E:/WINCE600/OSDes MutexDemo (E:/WINCE600/OSDes Dewer_Management (E:/WINCE600 Dewer_Management.es Power_Management.dat Power_Management.dat Power_Management.dat ProjSysgen.bat Solution Ex Catalog Ite Cass View	EY_CURRENT_USER EY_LOCAL_MACHINE SYSTEM CurrentControlSet Control State On SystemIdle UserIdle Timeouts RegEdit © Source	Cefault)	REG_SZ	(value not set)
Output				↓ ₽ ×
Show output from: Windows CE Debug	• Ŗ 📣 🛼 🛼	-		
				2
<				>
Ready				
📕 Start 📄 🕞 4 Windows Explorer 🔷	😵 EVM_3530 - Microsoft	👜 Lab 4-2_EVM.doc - Mi	() ()	😧 😵 12:58 PM

11. Examine the registry entry under

[HKLM\System\CurrentControlSet\Control\Power\Timeouts]. This changes the timeout period used by the Power Manager to determine when to move to the User Idle state.

🏽 EVM_3530 - Microsoft Visual Studio			
File Edit View Project Build Debug Target Tools Window Community Help			
👔 🗸 🛅 🗸 🚰 🛃 🖉 🔏 🤚 🛍 🦃 🤟 🖓 🗸 📮 🔍 🕨 TI_EVM_3530_ 🗸 Pla	atform Builder (_TGTCPU)	• 🏄	
● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ● ●	🗏 월 🗖 🖓 🗣	a a 4 B S S	
Device: CE Device - 🖓 😓 💷			
Solution Explorer - Solution 'EV 4 × Power_Management.reg StdAfx.h gpio_b	acklight.cpp tled_bad	klight.cpp	→ ×
HKEY_CLASSES_ROOT	Name	Туре	Data
LeakingMemory (E:/WINCE MutexDemo (E:/WINCE60(MutexDemo (E:/WINCE60(Power_Management (E:/W Include files Parameter files Power_Management ProjSysgen.bat HKEY_USERS	CDefault)	REG_SZ REG_DWORD	(value n 0x0000(
Solutio O Catalo C Catas View RegEdit 🗠 Source			
			₩ų×
Show output from: Windows CE Debug 🔹 🖓 🖓 😒 🕵 😨			
			~
			>
Ready			
🛃 Start 📄 🖆 4 Windows Explorer 🔹 🐢 EVM_3530 - Microsoft 🖳 Lab 4-	2_EVM.doc - Mi	🔇 😲 🐯	12:59 PM

Note The registry entries contained in subprojects are processed last when building the OS run-time image. These registry entries override any entries that are defined by operating system components or the BSP.

> Build and run the updated OS run-time image

- 12. Select **Target** | **Detach Device** from the Visual Studio menu to detach the existing device image.
- 13. Close any **remote tools** that you may have open.
- 14. Select Build | Advanced Build Commands | Build Current BSP and Subprojects from the Visual Studio menu.
- 15. Select **Target** | **Attach Device** from the Visual Studio menu to re-attach the device.
- Examine Power_Management application
 - 16. Open the **Power_Management.cpp** file from the Power_Management subproject using Solution Explorer.

- 17. Locate the PowerNotificationThread function. This secondary thread requests notification about power management events using the RequestPowerNotifications API. This thread allows the application to print out a message each time the system power state changes.
- 18. Locate the WinMain function. This function calls the SetPowerRequirement API against the backlight driver, forcing the backlight driver into a higher power state (D0). The ReleasePowerRequirement allows the backlight to return to its normal state (D4).

Run Power_Management application

- 19. Launch the **Power_Management.exe** application using **Target** | **Run Programs** from the Visual Studio menu.
- 20. Observe debug messages in the **Output** window similar to the following:

```
76711 PID:423000e TID:43f000e Power Notification Message: PBT_POWERINFOCHANGE
76711 PID:423000e TID:43f000e Length: 28
76711 PID:423000e TID:43f000e BatteryLifeTime = -1
76711 PID:423000e TID:43f000e BackupBatteryLifeTime = -1
76711 PID:423000e TID:43f000e BackupBatteryFullLifeTime = -1
76711 PID:423000e TID:43f000e BatteryFlag = 255
76711 PID:423000e TID:43f000e BatteryLifePercent = 255
76711 PID:423000e TID:43f000e BackupBatteryLigg = 255
76711 PID:423000e TID:43f000e BackupBatteryLige = 255
```

Analysis These messages come from the PowerNotificationThread, which received a PBT_POWERINFOCHANGE message. This message provides information from the battery driver about the state of the power sources on the device.

21. Click the **OK** button on the **Power dialog box** on the device. Observe the following debug messages in the **Visual Studio Output** window:

150459 PID:423000e TID:43f000e Power Notification Message: PBT_TRANSITION 150459 PID:423000e TID:43f000e Flags: 11000000 150459 PID:423000e TID:43f000e Length: 18 150459 PID:423000e TID:43f000e SystemPowerState: on 150459 PID:423000e TID:43f000e BKL_IOControl IOCTL code = 3280904 150459 PID:423000e TID:43f000e BKL: Received IOCTL POWER_SET 150459 PID:423000e TID:43f000e IOCTL POWER SET to D0

Followed shortly by:

450461 PID:423000e TID:43f000e Power Notification Message: PBT_TRANSITION 450461 PID:423000e TID:43f000e Flags: 0 450461 PID:423000e TID:43f000e Length: 22 450461 PID:423000e TID:43f000e SystemPowerState: systemidle

- Analysis These messages show a system power state change to on, because we clicked a button on the screen. In addition, the backlight driver was forced to move to the D0 state because the application called SetPowerRequirement against it. The application subsequently received notification that the device changed to a system power state of useridle after 5 seconds of inactivity.
 - 22. Click the **OK** button on the **Power dialog box** on the device. This will cause the application to release the power requirement on the backlight, resulting in debug message output similar to the following:

```
150459 PID:423000e TID:43f000e Power Notification Message: PBT_TRANSITION 150459 PID:423000e TID:43f000e Flags: 11000000 150459 PID:423000e TID:43f000e Length: 18 150459 PID:423000e TID:43f000e SystemPowerState: on 150459 PID:423000e TID:43f000e BKL_IOControl IOCTL code = 3280904 150459 PID:423000e TID:43f000e BKL: Received IOCTL_POWER_SET 150459 PID:423000e TID:43f000e IOCTL_POWER_SET to D0 450461 PID:423000e TID:43f000e Flags: 0 450461 PID:423000e TID:43f000e Length: 22 450461 PID:423000e TID:43f000e SystemPowerState: systemidle
```

23. Click the **OK** button on the **Power dialog box** again. This will cause the application to exit.

Lab 5-1: Static and Dynamic Libraries

Objectives

- Create simple static library
- Link the static library with a dynamic library
- Link the dynamic library with an executable

Prerequisites

• Completed Lab 2-1

Estimated time to complete this lab: 45 minutes

Lab Setup

To complete this lab, you must have:

- A development workstation running Windows XP
- Visual Studio 2005 (Version 8) with Platform Builder plug-in
- A development workstation running Windows XP
- Visual Studio 2005 (Version 8) with Platform Builder plug-in
- CE 6.0
- CE 6.0 2006 Roll up
- CE 6.0 Service Pack 1
- CE 2007 Updates upto month 9th month {QFEs}
- CE R2
- CE 2007 Updates 11th and 12th month {QFES}
- CE 2008 Updates
- A running image from Lab 2-1

Exercise 1 Create a static library (LIB)

In this exercise you will create a static library with routines that you will later link to when you build a dynamic library and an executable.

Note This exercise involves a bit of typing; if you perfer you may copy the text from the Student files.

> Create a static library subproject

- 1. Select **Project** | **Add New Subproject...** from the Visual Studio menu. This will bring up the **Windows CE Subproject Wizard**.
- 2. Select the WCE Static Library template.
- 3. Set the Subproject name to Power_Status and click Next.
- 4. Check the **Precompiled header** box.
- 5. Click Finish.
- 6. Configure the **Power_Status** subproject to be **excluded from the image** and **always build and link as debug**, as documented in Lab 2-2.

> Create files

- 7. Right click on the **Power_Status** subproject in the Solution Explorer and select **Add** | **New Item...**
- 8. Select the **Code** category, the **C++ File** (.cpp) template, and then type **Power_ON_OFF** in the name field.

Categories:		Templates:		<u> </u>
Platform B	uilder for CE 6.0	Visual Studio installed template	es	
Code Resource Utility		C++ File (.cpp) Header File (.h) Module-Definition File (.def) My Templates Search Online Templates	C File (.c) -ƳƳMidl File (.idl)	
Creates a file (containing C++ source	e code		
ame:	Power_ON_OFF	-		

- 9. Click on Add to add the new file.
- 10. Right click on the **Power_Status** subproject in the Solution Explorer and select **Add** | **New Item...**
- 11. Select the **Code** category, the **Header File** (.h) template, and then type **Power_Status** in the name field.
- 12. Click on Add to add the new file.
- 13. Using the Solution Explorer, locate the **Power_ON_OFF.cpp** file in the **Power_Status** subproject and open it.
- 14. Add the following code to the **Power_ON_OFF.cpp** file:

```
#include "stdafx.h"
LPCTSTR g_StrOn = L"Power is on";
LPCTSTR g_StrOff = L"Power is off";
LPCTSTR PowerOn()
{
    return g_StrOn;
}
LPCTSTR PowerOff()
{
    return g_StrOff;
}
```

- 15. Save and close Power_ON_OFF.cpp.
- 16. Expand the Include files node in the Power Status subproject and open stdafx.h
- 17. Add an include for **<windows.h>** as follows:

// TODO: reference additional headers your program requires here
#include <windows.h>

- 18. Save and close stdafx.h.
- 19. Using the Solution Explorer, locate the **Power_Status.h** file in the **Power_Status** subproject and open it.
- 20. Add the following to Power_Status.h:

```
extern LPCTSTR PowerOff(void);
extern LPCTSTR PowerOn(void);
```

21. Save and close Power_Status.h.

> Build the library

22. Right click the **Power_Status** subproject in the Solution Explorer and select **Build**.

Exercise 2 Create a dynamic library (DLL)

In this exercise you will create a dynamic library that will link with the static library you created previously.

Note This exercise involves a bit of typing; if you perfer you may copy the text from the Student files.

> Create the dynamic library subproject

- 1. Select Project | Add New Subproject... from the Visual Studio menu.
- 2. Select the WCE Dynamic-Link Library template.
- 3. Set the Subproject name to ScanBarcode and click Next.
- 4. Select A simple Windows Embedded CE DLL subproject and click Finish.
- 5. Configure the **ScanBarcode** subproject to be **excluded from the image** and **always build and link as debug**, as documented in Lab 2-2.

> Edit source files for DLL project

- 6. In the Solution Explorer, right-click on the **ScanBarcode** subproject, and then select **Add** | **New Item...**
- 7. Select the **Code** category, the **Header File** (.h) template, and then type **ScanBarcode** in the name field.
- 8. Using the Solution Explorer, open the **ScanBarcode.h** file from the **ScanBarcode** subproject.
- 9. Add the following code to the ScanBarcode.h file:

```
#include <windows.h>
EXTERN_C LPCTSTR ScanBarcode(void);
EXTERN_C LPCTSTR ScanPowerOff(void);
EXTERN C LPCTSTR ScanPowerOn(void);
```

- 10. Using the Solution Explorer, open the **ScanBarcode.def** file from the **Parameter files** node in the ScanBarcode subproject.
- 11. Add the following to the DEF file:

LIBRARY ScanBarcode

```
EXPORTS
ScanPowerOff
ScanPowerOn
ScanBarcode
```

- 12. Save and close ScanBarcode.def.
- 13. Using the Solution Explorer, open the **ScanBarcode.cpp** file from the ScanBarcode subproject.
- 14. Add an include statement for Power_Status.h as follows:

15. Add the following code snippet to ScanBarcode.cpp after the inclusion of Power Status.h.

```
LPCTSTR g_StrScan = L"123456789ABC";
EXTERN_C LPCTSTR ScanBarcode(void)
{
    return g_StrScan;
}
EXTERN_C LPCTSTR ScanPowerOn(void)
{
    return PowerOn();
}
EXTERN_C LPCTSTR ScanPowerOff(void)
{
    return PowerOff();
}
```

16. Save and close ScanBarcode.cpp.

Link to static library

- 17. Right click on the **ScanBarcode** subproject node in the Solution Explorer, and then select **Open**. The ScanBarcode **SOURCES** file will open.
- 18. Locate the section of the file containing TARGETLIBS.
- 19. Add a reference to the Power_Status.lib static library by modifying this section as follows:

```
TARGETLIBS= \
  $ (_PROJECTROOT) \cesysgen \sdk\lib \$ (_CPUINDPATH) \coredll.lib \
  $ (PBWORKSPACEROOT) \Power_Status \obj \$ (_CPUINDPATH) \Power_Status.lib \
```

- **Note** The trailing backslash characters on each line are line continuation characters. Ensure that there is no white space after them. Also, ensure that there is a blank line following the last line.
 - 20. Add the path to directory containing the Power_Status.h header file by adding the following to the bottom of the **SOURCES** file:

```
INCLUDES= \
  $ (PBWORKSPACEROOT) \Power Status \
```

- **Note** Ensure that there is at least one blank line prior to the line containing the INCLUDES directive. This ensures that there are no line continuation characters prior to this statement that are still in effect.
 - 21. Save and close the SOURCES file.
 - 22. Right click on the **ScanBarcode** subproject in the Solution Explorer and select **Properties**.
 - 23. Select the C/C++ tab and observe the **Include Directories** entry. Notice that the directory you just added with the INCLUDES directive in the SOURCES file is listed here.
 - 24. Select the **Link** tab and observe the **Additional Libraries** entry. Notice that the library you just added with the TARGETLIBS directive in the SOURCES file is listed at the end of this line.
- **Note** The SOURCES file itself controls the build rules for the subproject. The graphical user interface shown here provides an alternate way to view and modify this file.

25. Select **Cancel** to close this dialog without making any changes.

> Build the library

26. Right click the **ScanBarcode** subproject in the Solution Explorer and select **Build**.
Exercise 3 Create an executable (EXE)

In this exercise you will create an executable that uses functionality from the dynamic library you just created.

- Adding existing application subproject
 - 1. Copy the **BarcodeDllTest** subproject from the Student files to your OS Design at C:\WINCE600\OSDesigns\EVMOSDesign\EVMOSDesign.
 - 2. Right click on the **Subprojects** node in the Solution Explorer and select **Add Existing Subproject.**
 - 3. Select the **BarcodeDllTest.pbpxml** file from the **BarcodeDllTest** folder.
 - 4. Configure the **BarcodeDllTest** subproject to be **excluded from the image** and **always build and link as debug**, as documented in Lab 2-2.

Add reference to dll

- 5. Right click on the **BarcodeDllTest** subproject in the Solution Explorer and select **Open**.
- 6. Add the following to the bottom of the file:

```
INCLUDES= \
  $ (PBWORKSPACEROOT) \ScanBarcode \
```

Note Ensure that there is a blank line preceding the INCLUDES directive. Ensure there is no whitespace after the trailing backslashes.

7. Locate the TARGETLIBS directive and add a reference to **ScanBarcode.lib** as follows:

```
TARGETLIBS= \
  $ (_PROJECTROOT)\cesysgen\sdk\lib\$ (_CPUINDPATH)\coredll.lib \
  $ (PBWORKSPACEROOT)\ScanBarcode\obj\$ ( CPUINDPATH)\ScanBarcode.lib \
```

Note Ensure that there is a blank line after the line containing ScanBarcode.lib. Ensure there is no white space after the trailing backslashes.

- 8. Save and close the sources file.
- 9. Right click on the BarcodeDllTest subproject and select Build.

Run the BarcodeDllTest application

- 10. Launch **BarcodeDllTest.exe** using **Target** | **Run Programs** from the Visual Studio menu.
- 11. The BarcodeDllTest.exe application will present the following user interface. You can exercise it by clicking on the various buttons.



This simple application makes calls into the linked Scanbarcode.dll dynamic library, which includes functionality from the Power_Status.lib static library. You may wish to set breakpoints on functions in these modules and view the call stacks to see how they are eventually called from the application.

Lab 5-2: Command Line Build

Objectives

- Learn how some of the build commands available in the Visual Studio IDE map to command line actions
- Compare IDE and command line build mechanisms

Prerequisites

• Completed Lab 2-2

Estimated time to complete this lab: 20 minutes

Lab Setup

To complete this lab, you must have:

- A development workstation running Windows XP
- Visual Studio 2005 (Version 8) with Platform Builder plug-in
- A development workstation running Windows XP
- Visual Studio 2005 (Version 8) with Platform Builder plug-in
- CE 6.0
- CE 6.0 2006 Roll up
- CE 6.0 Service Pack 1
- CE 2007 Updates upto month 9th month {QFEs}
- CE R2
- CE 2007 Updates 11th and 12th month {QFES}
- CE 2008 Updates
- A running image from Lab 2-2

Exercise 1 Project build in command line

This exercise will demonstrate how some of the build commands available in the Visual Studio IDE map to command line actions. All of the build commands that are accessible from the Visual Studio IDE eventually resolve to command line actions. You will see how to determine what these mappings are.

We will use the MyHelloWorldApp subproject that we created at the beginning of this course to point out some of the build commands.

> Build the MyHelloWorldApp subproject from the IDE

- 1. Detach the target if attached.
- 2. Right click on the MyHelloWorldApp subproject and choose **Build**. Observe the Build Output window.

Output						
Show output from: Build 🔹 😽 😽 🗐 🗐 🗐						
Build started: Project: EVM_3530, Configuration: TI_EVM_3530_ARMV4I Release Platform Buil E:\WINCE600\0SDesigne \Sample0SDesign \EVM_3530\MyHelloWorldApp\sources Starting Build: set WINCEREL=166build						
BUILD: [Thrd:Sequence:Type] Message BUILD: [OO:OOOOOOOOOO:PROGE] Build started with narameters:						
BUILD: [00:0000000001:PROGC] Build started in directory: E:\WINCE600\OSDesigns\SampleOSDesign\E						
BUILD: [00:000000002:PROGC] Checking for E:\WINCE600\sdk\bin\i386\srccheck.exe.						
BUILD: [00:000000003:PROGC] Running passes WCEFILESO, MIDL, MC, ASN, THUNK, PRECOMPHEADER, COM						
BUILD: [00:000000005:PROGC] Computing Include file dependencies. BUILD: [00:0000000005:PROGC] Checking for SDK include directory: E:\WINCE600\sdk\CE\inc.						
BUILD: [00:000000006:PROGC] Scan E:\WINCE600\OSDesigns\SampleOSDesign\EVM_3530\MyHelloWorldApp						
BUILD: [00:000000007:PROGC] Saving E:\WINCE600\OSDesigns\SampleOSDesign\EVM_3530\MyHelloWorldA						
BUILD: [00:0000000011:PROGC] Building PRECOMPHEADER Pass in E:\WINCE600\OSDesigns\SampleOSDesig BUILD: [0]:00000000026:PROGC] Greate preservited beder Stdift h shi\DruudI\ustail\Stdift shi E:						
BUILD: [01:0000000031:PROGC] Creace precompiled header Schrixth obj(XRNV41(recail(Schrixtob) K. BUILD: [00:0000000031:PROGC] Building COMPILE Pass in K:\WINCE600\OSDesigns\SampleOSDesign\EVM						
BUILD: [01:000000046:PROGC] Resource Compiling .\MyHelloWorldApp.rc						
BUILD: [01:000000051:PROGC] Compiling .\MyHelloWorldApp.cpp						
BUILD: [00:000000058:PROGC] Building LINK Pass in E:\WINCE600\OSDesigns\SampleOSDesign\EVM 353						

- 3. Observe the circled commands. This is a **compound** statement that consists of two commands. First, the environment variable WINCEREL is set to 1 and then the build command is issued.
- 4. Select **Build** | **Targeted Build Settings** | **Make Run-Time Image After Building** on the Visual Studio menu. Selecting this menu item should cause it to become checked.

Note We disabled this option previously because we generally did not want the runtime image to be built after each targeted build. We are enabling it here to show its functionality.

- 5. Clear the Build Output window by right clicking in it and selecting Clear All.
- 6. Right click on the **MyHelloWorldApp** subproject and choose **Rebuild**. Observe the Build Output window.

Dutput 🔀
Show output from: Build 🔹 🕞 🖓 🕼 🖏 🐺 🖃
Build started: Project: EVM_3530, Configuration: TI_EVM_3530_ARMV4I Release Platform Buil 🔥
E:\WINCE600\OSD <u>esigns\Semple65Design\EWH_8530\My</u> HelloWorldApp\sources
Starting Build: set WINCEREL=166build -c66makeing
BUILD: [Thrd:Sequence:Type] Message
BUILD: [00:000000000:PROGC] Build started with parameters: -c
BUILD: [00:0000000001:PROGC] Build started in directory: E:\WINCE600\OSDesigns\SampleOSDesign\E
BUILD: [00:000000002:PROGC] Checking for E:\WINCE600\sdk\bin\i386\srccheck.exe.
BUILD: [00:000000003:PROGC] Running passes WCEFILESO, MIDL, MC, ASN, THUNK, PRECOMPHEADER, COM
BUILD: [00:000000004:PROGC] Ignoring build database (-c specified).
BUILD: [00:000000005:PROGC] Computing include file dependencies:
BUILD: [00:000000006:PROGC] Checking for SDK include directory: E:\WINCE600\sdk\CE\inc.
BUILD: [00:000000007:PROGC] Scan E:\WINCE600\OSDesigns\SampleOSDesign\EVM_3530\MyHelloWorldApp
BUILD: [00:0000000008:PROGC] Saving E:\WINCE600\0SDesigns\SampleOSDesign\EVM_3530\MyHelloWorldA
BUILD: [00:000000012:PROGC] Building PRECOMPHEADER Pass in E:\WINCE600\OSDesigns\SampleOSDesig
BUILD: [01:0000000027:PROGC] Create precompiled header StdAfx.h obj\ARMV4I\retail\StdAfx.obj K:
BUILD: [00:000000032:PROGC] Building COMPILE Pass in E:\WINCE600\OSDesigns\SampleOSDesign\EVM_
BUILD: [01:0000000047:PROGC] Resource Compiling .\MyHelloWorldApp.rc
_BUILD: [01:0000000052:PROGC] Compiling .\MyHelloWorldApp.cpp

7. Observe the circled commands. This time there are three commands. The build command has the -c parameter, causing clean build to be performed. The clean build was performed because we chose **Rebuild** instead of **Build** from the menu. The last command is **makeimg**, which builds the OS run-time image. This command is performed because we selected the option to **Make Run-Time Image After Building** for targeted builds.

> Build the MyHelloWorldApp subproject from the command line

8. Right click on the MyHelloWorldApp subproject and select **Open Build Window**. The following window is displayed:



- **Note** This is not a generic DOS command shell. This command shell has been automatically configured for Windows Embedded CE 6.0 builds. You can not simply launch a generic DOS command shell and be able to do CE builds without configuring the build environment properly.
 - 9. Type **set** at the command prompt and press <Enter>. Observe the many environment variables specific to Windows Embedded CE that have been configured. Notice that WINCEREL is one of those variables.



- **Note** The **WINCEREL** environment variable causes the build mechanism to automatically copy the build output files to the flat release directory. This makes the build output files immediately available to be included in the OS run-time image, or to be loaded from the flat release directory using the debug shell.
 - 10. Type **build** –**c** at the command line.
 - 11. Compare the build output here with the information in the Build Output window. Notice that they are the same.

🛤 EVM_3530 - TI_EVM_3530_ARMV4I Release	- 🗆 🗙
BUILD: [01:0000000075:PROGC] Linking obj\ARMU4I\retail\MyHelloWor BUILD: [00:000000107:PROGC] Saving E:\WINCE600\OSDesigns\Sample0 20.Mullollolue1d0ue3Doild_dot	ldApp.exe SDesign\EVM_35
BUILD: [00:000000109:PROGC] Done.	
BUILD: [00:0000000111:PROGC] Midl 0	js Errors 0 0
BUILD: LUU:UUUUUU112:PROGC J Message U BUILD: [00:000000113:PROGC] Precomp Header 1	0 0
BUILD: [00:000000114:PROGC] Resource 1 BUILD: [00:000000115:PROGC] MASM 0	0 0 0 0
BUILD: [00:000000116:PROGC] SHASM 0 BUILD: [00:000000117:PROGC] ARMASM 0	0 0 0 0
BUILD: [00:000000118:PROGC] MIPSASM 0 BUILD: [00:000000119:PROGC] C++ 1	0 0 0 0
BUILD: [00:000000120:PROGC] C 0 BUILD: [00:000000121:PROGC] Static Libraries 0	0 0 0 0
BUILD: [00:000000122:PROGC] Exe's 1 BUILD: [00:000000122:PROGC] Exe's 1	0 0
BUILD: [00:000000124:PROGC] Preprocess deffile 0	0 0
BUILD: [00:000000125:PROGC] CSharp Compile 0	0 0
BUILD: L00:000000127:PROGC] Other 0 BUILD: L00:000000128:PROGC]	0 0
BUILD: 100:0000000130:PROGC] Total 4 BUILD: 100:0000000130:PROGC]	

- 12. Type **makeimg** at the command line. This command causes the OS run-time image to be built.
- 13. Compare the makeing output here with the corresponding portion in the Build Output window. Notice that they are the same.



EVM_3530 - TI_EVM_3530_	ARMV41 Release	- 🗆 🗙			
Start RAM:	852£0000				
Start of free RAM:	85512000				
End of RAM:	8 b 0 0 0 0 0 0				
Number of Modules:	191				
Number of Copy Sections:	4				
Copy Section Offset:	840faf84				
Kernel Flags:	0000002				
FileSys 4K Chunks/Mbyte:	128 <2Mbyte 128 2-4Mbyte 0 4-6Mbyte 0 >6Mbyte				
CPU Type:	01c2h				
hiscellaneous Flags	0002h				
Extensions Pointer:	84002020				
lotal KVM size:	01205084 (17812484)				
Starting ip: Day files size:	8400acc4 88529-10				
Naw Files Size. Compussed files size.	00527eur				
Compacting hin file	8620ac10				
Done [†]					
makeimg: Check for E:\VII	NCE600\OSDesigns\SamuleOSDesign\EUM_3530\RelDir\TI_F	RUM 3			
S30 ARMU41 Release\PostRomImage_bat to win_					
makeimg: Check for E:\WII	NCE600\OSDesigns\SampleOSDesign\EVM_3530\RelDir\TI_E	EUM_3			
530_ARMV4I_Release\PostMa	akeImg.bat to run.				
makeimg: Change director	y to Ĕ:\WINCE600.				
makeimg: run command: cm	d /C E:\WINCE600\public\common\oak\misc\pbpostmakeim	ıg			
E:\WINCE600\OSDesigns\Sa	npleOSDesign\EVM_3530\MyHelloWorldApp>	-			

- 14. Close the build window.
- 15. Select **Build | Targeted Build Settings | Make Run-Time Image After Building** from the Visual Studio menu to disable this option. We do not want to burden future targeted builds with this build step.

Lab 5-3: Troubleshooting Link Errors

Objectives

- Identify linker errors
- Learn how to determine the correct link library
- Resolve link errors

Prerequisites

• Completed Lab 2-1

Estimated time to complete this lab: 20 minutes

Lab Setup

To complete this lab, you must have:

- A development workstation running Windows XP
- Visual Studio 2005 (Version 8) with Platform Builder plug-in
- A development workstation running Windows XP
- Visual Studio 2005 (Version 8) with Platform Builder plug-in
- CE 6.0
- CE 6.0 2006 Roll up
- CE 6.0 Service Pack 1
- CE 2007 Updates upto month 9th month {QFEs}
- CE R2
- CE 2007 Updates 11th and 12th month {QFES}
- CE 2008 Updates
- A running image from Lab 2-1

Exercise 1 Troubleshoot build errors

In this exercise you will identify a linker error and resolve it.

- > Add the existing TroubleShoot_Build subproject
 - 1. Copy the **TroubleShoot_Build** subproject from the Student files to your OS Design at C:\WINCE600\OSDesigns\EVMOSDesign\EVMOSDesign.
 - 2. Right click on the **Subprojects** node in the Solution Explorer and select **Add Existing Subproject.**
 - 3. Select the **TroubleShoot_Build.pbpxml** file from the **TroubleShoot_Build** folder.

Open					? 🛛
Look <u>i</u> n:	🚞 TroubleShoot_	Build	 G 	🦻 📂 🛄	
My Recent Documents	TroubleShoot_B	uild.pbp×ml			
Desktop					
My Documents					
My Computer					
	File <u>n</u> ame:			~	<u>O</u> pen
My Network	Files of type:	Windows Embedded CE	Subproject Files	: (.pbp: 🔽	Cancel

- 4. Configure the **TroubleShoot_Build** subproject to be **excluded from the image** and **always build and link as debug**, as documented in Lab 2-2.
- 5. Using the Solution Explorer, open the file **TroubleShoot_Build.cpp** from the **TroubleShoot_Build** subproject. Notice that it is simply a call to MessageBox:

MessageBox(NULL, TEXT("Hello World!!"), TEXT("TroubleShoot App. is created!!")
,MB_OK);

> Build the subproject.

6. Right click on the **TroubleShoot_Build** subproject and select **Build**. Note error messages in the Build Output window similar to the following:

Output					
Show output from: Build	- 🖓 🖨 🗄		Ŧ		
BUILD: [01:000000068:PROGC	Linking obj\ARMV4I\ret	ail\Tro	ubleShoot_	Build exe	A
BUILD: [01:000000086:ERRORE]	TroubleShoot_Build.obj	: erro	r LNK2019:	unresolv	ed external symbol MessageBoxW referenced in fu
BUILD: [01:000000087:ERRORE]	corelib0.lib(cexit.obj) : err	or LNK2019	unresol	ved external symbol TerminateProcess referenced
BUILD: [01:000000088:ERRORE]	obj\ARMV4I\retail\Trou	bleShoc	t_Build.ex	e . fatal	error LNK1120. 2 unresolved externals
BUILD: [01:000000093:ERRORE]	EDITBIN : fatal error	LNK1104	: cannot c	pen file	'obj\ARMV4I\retail\TroubleShoot_Build.exe'
BUILD: [00:000000102:PROGC]		Files	Warnings	Errors	
BUILD: [00:000000103:PROGC]	Midl	0	0	0	
BUILD: [00:000000104:PROGC]	Message	0	0	0	
BUILD: [00:000000105:PROGC]	Precomp Header	1	0	0	
BUILD: [00:000000106:PROGC]	Resource	0	0	0	
BUILD: [00:000000107:PROGC]	MASM	0	0	0	
BUILD: [00:000000108:PROGC]	SHASM	0	0	0	
BUILD: [00:000000109:PROGC]	ARMASM	0	0	0	
BUILD: [00:000000110:PROGC]	MIPSASM	0	0	0	
BUILD: [00:000000111:PROGC]	C++	1	0	0	
BUILD: [00:000000112:PROGC]	l C	0	0	0	
BUILD: [00:000000113:PROGC]	Static Libraries	0	0	0	
BUILD: [00:000000114:PROGC]	Exe's	1	0	4	
BUILD: [00:000000115:PROGC]	Dll's	0	0	0	
BUILD: [00:000000116:PROGC]	Preprocess deffile	0	0	0	
BUILD: [00:000000117:PROGC]	Resx	0	0	0	
BUILD: [00:000000118:PROGC]	CSharp Compile	0	0	0	
BUILD: [00:000000119:PROGC]	Other	0	0	1	
BUILD: [00:000000120:PROGC]					
BUILD: [00:000000121:PROGC]	Total	3	0	5	
BUILD: [00:000000122:PROGC]	Province and the second state				
BUILD: [00:000000123:PROGC]	0 Warnings, 5 Errors				
BUILD: [00:000000124:PROGC]	GetSystemTimes (second	is): Idl	le: 0 B	Cernel: 0	User: 1
		NN 2			
× I	100				

7. Observe that the error occurred during the link phase. The functions **MessageBoxW** and **TerminateProcess** could not be resolved. These functions are contained in an external library. The problem with this build is not in the application source code, but with the subproject settings that determine the external libraries that are used. We need to determine the correct library to resolve these functions.

Identify library

- 8. Select Edit | Find and Replace | Find in Files from the Visual Studio menu. This will bring up the Find and Replace dialog.
- 9. Type MessageBoxW in the Find what box.
- 10. Type C:\WINCE600\PUBLIC\COMMON\OAK\LIB\ARMV4I\RETAIL in the Look in: box. You can also navigate to this folder and select it using the Choose Search Folders button on the far right hand side of this box.
- 11. Expand the **Find options** box and enter ***.def** in the **Look at these file types:** box.
- 12. Click **Find All** to perform the search.

- Observe that the MessageBoxW function is exported by coredll.def and k.coredll.def. Therefore we need to link against coredll in order to resolve this link error.
- 14. Repeat this process for TerminateProcess.
- 15. Observe that **TerminateProcess** is also exported by coredll. Coredll will resolve both link errors for us.
- **Note** This same general procedure can be followed for link errors that arise due to functionality contained in dynamic link libraries. The .def files will indicate which dll contains the desired functionality. However, not all .def files are located in the Common subtree. Each of the primary subtrees under \PUBLIC contain their own functionality, and their own .def files. You may need to search those trees as well in order to find the correct dll.

Add the correct link library

- 16. Right click on the **TroubleShoot_Build** subproject and select **Open**. The **SOURCES** file for this subproject will open.
- 17. Observe that there is no **TARGETLIBS** directive. We will add one with an entry for coredll.
- 18. Add the following to the bottom of the SOURCES file:

```
TARGETLIBS= \
   $(_PROJECTROOT)\cesysgen\sdk\lib\$(_CPUINDPATH)\coredll.lib \
```

Note Ensure that there is a blank line prior to the TARGETLIBS directive. Ensure that there is no what space after the trailing backslashes.

Also note that the path used to resolve coredll.lib is not the original location in the \PUBLIC\COMMON tree. Instead, it resolves to the filtered version of coredll.lib located in the project directory. This ensures that we link against the version of coredll.lib that was componentized for our OS Design.

- 19. Save and close the **SOURCES** file.
- 20. Right click on the **TroubleShoot_Build** subproject and select **Build**. Observe that the errors have been resolved, the build is successful.

Lab 6-1: Registry Initialization

Objectives

- Understand that multiple sources provide initial registry content
- Understand registry source precedence when conflicts occur
- Use the Run-Time Image Viewer to observe content in the OS Run-Time image

Prerequisites

• Completed Lab 2-1

Estimated time to complete this lab: 20 minutes

Lab Setup

To complete this lab, you must have:

- A development workstation running Windows XP
- Visual Studio 2005 (Version 8) with Platform Builder plug-in
- A development workstation running Windows XP
- Visual Studio 2005 (Version 8) with Platform Builder plug-in
- CE 6.0
- CE 6.0 2006 Roll up
- CE 6.0 Service Pack 1
- CE 2007 Updates upto month 9th month {QFEs}
- CE R2
- CE 2007 Updates 11th and 12th month {QFES}
- CE 2008 Updates
- A running image from Lab 2-1

Exercise 1

In this exercise you will learn how the various registry files in the OS Design are combined to create the final device registry. You will observe how conflicting registry entries are resolved.

- Add a new key to platform.reg
 - 1. Expand the C:/WINCE600 node under EVMOSDesign in the Solution Explorer. Navigate to Platform/EVMBSP/Parameter Files.
 - 2. Double click on platform.reg to open it in the Visual Studio editor.
 - 3. Navigate to **HKEY_LOCAL_MACHINE** then right-click on the **Software** key and select **New** | **Key**.
 - 4. Name the new key HelloWorld.



🏶 EVM_3530 - Microsoft Visual Sti	ıdio		- 7 🛛				
File Edit View Project Build De	bug Target Data Tools Window Comm	unity Help					
🖥 • 🔤 • 📂 🛃 🗿 🗼 🛍 🛍	🛅 🕶 🔤 🖌 🚰 🕌 🔏 🖄 👘 😤 🖉 – 🔍 – 🚚 – 🖳 🕨 TI_EVM_3530_ – Platform Builder (_TGTCPU) 📼 🙆						
· · · · ·	0, Al 42 44 - 🕴 🖪 🥆 k	. A2 🗱 🗱 🗏 🤮	🗖 🖓 🗣 🖓 🖧 🖗 🤤 🗧				
Device: CE Device 🔹 🗣 😓	• • •						
Solution Explorer - Solution 'EV $~~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ $	platform.reg StdAfx.h	→ ×	Properties 🗸 🗸 🗙				
	HKEY_CLASSES_ROOT	Name	HKEY_LOCAL_MACHINE\Software 🕶				
EVM_3530 E:/WINCE600 PLATFORM PLATFORM PLATFORM CEPC COMMON SCEPC COMMON SCEPC MAINSTONEIII SITMP3780 MAINSTONEIII SITMP3780 MAINSTONEIII SITMP3780 Parameter Files platform.bb platform.db platform.db platform.db platform.reg SRC PRIVATE PUBLIC PRIVATE PUBLIC Catalo CCass View Re Output	HKEY_CURRENT_USER HKEY_LOCAL_MACHINE Drivers HARDWARE Init OMAPPMX Software HeloWorld System HKEY_USERS	(Default)	Name HKEY_LOCAL_MACHI Name HKEY_LOCAL_MACHI Name of object. X				
🛃 start 📄 6 Windo 🕞	🛠 2 Microso 👻 🛄 Tera Term 🔞	Microsoft A 🕎 Lab	6-1_EV 🔇 🕖 🧐 🕲 5:43 PM				

5. Right click on the **HelloWorld** key and select **New** | **String Value**. Name the new value **PlatformMessage**.

4 Lab 6-1 Registry Initialization

🍘 EVM_3530 - Microsoft Visual Stu	dio		- 7 🛛
File Edit View Project Build Deb	oug Target Data Tools Window Commu	nity Help	
🔚 • 🕮 • 💕 🛃 🥥 🐰 🖻 🛍	🔊 - (* - 🚚 - 🖳 🕨 TI_EVM_3530	Platform Builder (_TGTC	PU) 🔹 🖄 📮
·	1, p 4: 54 , i 🗈 🔁 🛌	A2 🛱 🛱 🗏 🔮	🗖 🖓 🗣 🖗 🖧 🖗 🖓 🖕
Device: CE Device 🔹 🗣 😓 💆	• • •		
Solution Explorer - Solution 'EV $~~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ $	platform.reg StdAfx.h	▼ X	Properties 🗸 🗸 X
	HKEY_CLASSES_ROOT	Name	HKEY_LOCAL_MACHINE\Software -
	HKEY_LOCAL_MACHINE HKEY_LOCAL_MACHINE Comm Drivers HARDWARE HARDWARE Init OMAPPMX Software Microsoft Hellowere HKEY_USEF Properties Properties E Properties	(Default) (ey String Value WORD Value Sinary Value Aulti-String Value 4UI-String Value (UI-String Value	Image: Second system Image: Second system
Re Output			×
🐉 start 📄 6 Windo 🕞	🛠 2 Microso 👻 🛄 Tera Term 🔞 M	1icrosoft A 🛛 🖳 Lat	0 6-1_EV 🔇 🔋 🐯 🔕 - 5:44 PM -

- 6. Right click on the **PlatformMessage** value and select **Properties**.
- 7. Type Hello from platform.reg in the Data field.
- 8. Save and close **platform.reg**.

Properties	×
PlatformMessage	e. Registry Value 🛛 🚽
2↓ □	
Misc	
Data	Hello
Key	HKEY_LOCAL_MACHI
Name	PlatformMessage.
Туре	REG_SZ
Data Data of registry val	ue.

- > Build the modified BSP
 - 9. **Detach** from the device if connected.
 - 10. Select Build | Advanced Build Commands| Build Current BSP and Subprojects from the Visual Studio menu.

🥙 EVM_3530 - Microsoft Visual Studio 📃 🖃 🔀					
File Edit View Project Build	d Debug Target Data Tools Windo	w Community Help			
i 🛅 • 🛅 • 💕 🛃 🥥 i 👹	Build Solution F7	1_3530_ 🝷 Platform Builder (_TGTCPU) 🔹 🏄	··		
	Rebuild Solution Ctrl+Alt+F7	N N H H H H H H H H H H H H H H H H H H			
Device: CE Device +	Clean Solution				
Solution Explorer - Solution 'EV	Build Selection		• X		
	Rebuild Selection	~	~		
- 💿 EVM_3530	Clean Selection				
E:/WINCE600	Project Only				
	Advanced Build Commands	Sysgen	444553540000 :		
E CEPC	Build All Subprojects	Clean Sysgen	13340000_INCLOI		
	Rebuild All Subprojects	Build and Sysgen			
H4SAMPLE	Build All SDKs	Rebuild and Clean Sysgen			
ISTMP3780	Copy Files to Release Directory	Build Current BSP and Subprojects			
🖃 🔂 TI_EVM_353	Make Run-Time Image	Rebuild Current BSP and Subprojects	rarely-used stu		
🖃 🔤 Paramete	Open Release Directory in Build Window				
platfi	Global Build Settings	-			
platfi	Targeted Build Settings	dditional beaders your program	requires here		
⊞ ·· 🛐 SRC	Batch Build				
	Configuration Manager	ION})			
	Deploy	_C++ Will insert additional dec.	larations immed:		
	Compile Ctrl+F7	FV STRAFY W ADDROSDR ADER 11D0	PED1 444553540		
Show output from: Build	. B B A =				
Build started: Pro	piect: KVM 3530. Configuration: 1	, 💶 NI RVM 3530 ARMV4I Release Platform 1	Buil NS		
start De Window.	😵 2 Microsof 🛄 Tera Term	Microsoft A Lab 6-1 EV	< 1 2 5:48 PM		

Output
Show output from: Build 🔹 💀 💀 😨
Build started: Project: EVM_3530, Configuration: TI_EVM_3530_ARMV4I Release Platform Buil 📈
Starting Build: blddemo -qbsp
BLDDEMO: Generating OS Design Folders
BLDDEMO: Done Generating OS Design Folders
BLDDEMO: Generating OS Design Files to E:\WINCE600\OSDesigns\SampleOSDesign\EVM_3530\Wince600\TI
BLDDEMO: Done Generating OS Design Files
CEBUILD: Deleting old build logs
CEBUILD: QUick BSP build and sysgen
Sysgening platform E:\WINCE600\platform\TI_EVM_3530
CEBUILD: Building (E:\WINCE600\platform\common)
BUILD: [Thrd:Sequence:Type] Message
BUILD: [00:000000000:PROGC] Build started with parameters:
BUILD: [00:0000000001:PROGC] Build started in directory: E:\WINCE600\PLATFORM\COMMON
BUILD: [00:000000002:PROGC] Checking for E:\WINCE600\sdk\bin\i386\srccheck.exe.
BUILD: [00:0000000003:PROGC] Running passes WCEFILESO, MIDL, MC, ASN, THUNK, PRECOMPHEADER, COM
BUILD: [00:000000004:PROGC] Loading database "E:\WINCE600\PLATFORM\COMMON\Build.dat".
BUILD: [00:000000005:PROGC] Done.
BUILD: [00:000000006:PROGC] Computing include file dependencies:
BUILD: [00:000000007:PROGC] Checking for SDK include directory: E:\WINCE600\sdk\CE\inc.
BUILD: [00:000000008:PROGC] Scan E:\WINCE600\PLATFORM\COMMON\SRC\COMMON\CACHE\
BUILD: [00:000000009:PROGC] Scan E:\WINCE600\PLATFORM\COMMON\SRC\COMMON\I0\BASE\
BUILD: [00:0000000010:PROGC] Scan E:\WINCE600\PLATFORM\COMMON\SRC\COMMON\I0\PCMCIA\
BUILD: [00:0000000011:PROGC] Scan E:\WINCE600\PLATFORM\COMMON\SRC\COMMON\I0\PCI\
BUILD: [00:0000000012:PROGC] Scan E:\WINCE600\PLATFORM\COMMON\SRC\COMMON\I0\PCI_PCMCIA\
BUILD: [00:000000013:PROGC] Scan E:\WINCE600\PLATFORM\COMMON\SRC\COMMON\INTR\COMMON\
BUILD: [00:0000000014:PROGC] Scan E:\WINCEGOO\PLATFORM\COMMON\SRC\COMMON\INTR\BASE\
🖊 Start 📄 🍋 6. Window 🔹 🐲 2. Microsof 🔹 🕮 Tera Term 💿 Microsoft A 👘 Lab 6-1. EV 🔍 👔

Note This command will also cause the OS Run-Time Image to be rebuilt due to the setting in **Build | Global Build Settings | Make Run-Time Image After Build**.

> Observe the change using the Run-Time Image Viewer

- 11. Select File | Open | File... from the Visual Studio menu.
- 12. Select **Windows Embedded CE Run-Time Image** from the **Files of Type** drop down box.
- Navigate to your flat release directory at C:\WINCE600\EVMOSDesign\EVMOSDesign\RelDir\EVMBSP_ARMV4I_Release and open nk.bin. The OS run-time image will open in the Run-Time Image Viewer.
- 14. Select NK, then double click on Registry. Navigate to HKEY_LOCAL_MACHINE\Software and verify the HelloWorld key exists in the image with the PlatformMessage value.

15. Close the file.

> Add a new key to project.reg in the OS Design

- 16. Expand the **Parameter Files** node under the **EVMOSDesign** project in the Solution Explorer.
- 17. Expand the EVMBSP: ARMV4I (Active) node and open the project.reg file.
- 18. Right-click on **HKEY_LOCAL_MACHINE** and add a new key called **Software**.
- 19. Right click on the Software key and add a new key called HelloWorld.
- 20. Right click on the **HelloWorld** key and add a **String Value** with the name **ProjectMessage**.
- 21. Set the value of ProjectMessage to Hello from project.reg.
- 22. Save and close the file.

Build the OS run-time image

23. Select Build | Advanced Build Commands| Build Current BSP and Subprojects from the Visual Studio menu.

> Observe the change using the Run-Time Image Viewer

- 24. Select File | Open | File... from the Visual Studio menu. In the file open box, select Windows CE Run-Time Image from the Files of Type drop down box.
- 25. Open **nk.bin** from the flat release directory. The OS run-time image will open in the Run-Time Image Viewer.
- 26. Select NK, then Registry. Navigate to HKEY_LOCAL_MACHINE\Software and verify the HelloWorld key exists in the image with the ProjectMessage value.
- 27. Close the file.

Dueling Registry Entries

- 28. Open platform.reg and navigate to the [HKEY_LOCAL_MACHINE\Software\HelloWorld] key.
- 29. Create a new String value named Conflicting with the value Platform.reg wins!.
- 30. Open project.reg and navigate to the [HKEY_LOCAL_MACHINES\Software\HelloWorld] key.
- 31. Create a new String value named Conflicting with the value Project.reg wins!.
- 32. Save and close both files.
- 33. Select Build | Advanced Build Commands| Build Current BSP and Subprojects from the Visual Studio menu.
- 34. Open the OS run-time image file from your flat release directory
- 35. Navigate to [HKEY_LOCAL_MACHINE\Software\HelloWorld] and view the Conflicting value. This will tell you which file has precedence in the build process.

Note There are more registry files involved in the build process than just project.reg and platform.reg. Each of the OS subtrees also provides registry content, as do the subprojects that might be added to an OS Design. There is a defined order with all of these potential registry sources.

36. Close the run-time image file.

Lab 6-2: Adding a New IOCTL to the OAL

Objectives

- Understand architecture of OAL IOCTL library in the Common code
- Understand how to add a new IOCTL to the OAL based on the Common code

Prerequisites

• Completed Lab 2-1

Estimated time to complete this lab: 20 minutes

Lab Setup

To complete this lab, you must have:

- A development workstation running Windows XP
- Visual Studio 2005 (Version 8) with Platform Builder plug-in

Exercise 1 Adding an IOCTL to the OAL

In this exercise you will add a new IOCTL to the OAL and demonstrate that it is working. The OAL exposes IOCTLs via the OEMIoControl() function. The libraries that are provided in the Platform\Common subdirectory include an implementation for the OEMIoControl() function. If the OAL in your BSP is based on the PQOAL architecture, or just uses this particular library from the Platform\Common code base you will use this method to add IOCTL support to your OAL.

We will first examine the implementation of the OEMIoControl() function in the Common code, then we will implement IOCTL_HAL_POSTINIT and verify that our implementation was successful.

> Review Common code implementation of OEMIoControl

- Open the file ioctl.c located in C:\WINCE600\PLATFORM\COMMON\SRC\COMMON\IOCTL
- 2. Observe that this file contains the required function **OEMIoControl().** This function is called by the kernel to implement all of the IOCTLs that are supported by the OAL. BSPs that link against the library containing this source code will use this implementation of OEMIoControl().
- 3. Observe that this function uses a global data structure named **g_oalIoCtlTable** containing the IOCTL codes and function pointers to implement them. BSPs that use the Common code to implement OEMIoControl() configure the function using this global data structure.
- 4. Close the file **ioctl.c**.
- Open the file ioctl.c located in C:\WINCE600\PLATFORM\EVMBSP\SRC\OAL\OALLIB
- 6. Observe that this file contains the data structure **g_oalloCtlTable** near the bottom of the file. This data structure is the one referenced by the OEMIoControl() function in the Common code. Note that this data structure is implemented using the header file **ioctl_tab.h**.
- 7. Also observe that this file contains routines that implement individual IOCTLs.
- Open the file ioctl_tab.h located in C:\WINCE600\PLATFORM\EVMBSP\SRC\INC

😂 inc			- 7 ×
File Edit View Favorites Tools	Help		A.
🚱 Back 🝷 🕥 🚽 🏂 🔎 Se	arch 😥 Folders 🛄 🕶		
Address 🛅 E:\WINCE600\PLATFORM\TI	_EVM_3530\SRC\INC		💌 🄁 Go
File and Folder Tasks	2 KB		<u>^</u>
 Rename this file Move this file 	bsp_base_regs.h C/C++ Header 2 KB	bsp_cfg.h C/C++ Header 53 KB	
 Copy this file Publish this file to the Web E-mail this file 	bsp_cfg.inc Include File 1 KB	bsp_keypad.h C/C++ Header 2 KB	
X Delete this file	h bsp_kitl_cfg.h C/C++ Header 2 KB	bsp_logo.h C/C++ Header 1 KB	
Other Places	h bsp_oalserdrv.h C/C++ Header 2 KB	bsp_opp_map.h C/C++ Header 7 KB	
Shared Documents My Computer	h Dsp_twl4030.h C/C++ Header 2 KB	bsp_version.h C/C++ Header 1 KB	
Details 😵	image_cfg.h C/C++ Header 6 KB	image_cfg.inc Include File 2 KB	
	ioctl_tab.h C/C++ Header 6 KB	kit_cfg.h C/C++ Header 2 KB	
Start 6 Window	oal_kitlseria C/C++ Hea Size: 5.14 KB	leader 9/9/2008 5:14 PM C+++ Header TH ···· ♥ Paul osoft A ♥ Lab 6-2_EV	. 🔇 🕽 🎯 5:55 PM

- 9. Observe that this file contains the pairing of IOCTL codes and the function pointers that implement them. The routines listed in this file are implemented directly in the BSP in the ioctl.c file mentioned above.
- 10. Observe that this file also includes the file **oal_ioctl_tab.h**. This file contains a list of IOCTL codes and function pointers for common IOCTLs that are already implemented in the Common code base. IOCTLs listed in that file do not have to be implemented in the BSP unless different functionality is needed.

> Add IOCTL_HAL_POSTINIT handler to the BSP

11. Add the following code snippet to the file **ioctl.c** located at C:\WINCE600\PLATFORM\EVMBSP\SRC\OAL\OALLIB. The routine should be added just above the g_oalIoCtlTable data structure.

4 Lab 6-2 Adding a New IOCTL to the OAL





12. Add the following line to the file ioct_tab.h located at C:\WINCE600\PLATFORM\EVMBSP\SRC\INC. The line should be added just below the { IOCTL_HAL_POSTINIT, 0, OALIOCTHAlPOSTINIT }, line.

#include <oal ioctl tab.h>

🗁 INC		- 2 ×
File Edit View Favorites Tools	Help	A.
🚱 Back 🝷 🕥 🚽 🏂 🔎 Se	arch 🎼 Folders 🛄 -	
Address 🗁 E:\WINCE600\PLATFORM\TI	_EVM_3530\SRC\INC	💌 🄁 Go
File and Folder Tasks	h bsp_base_regs.h C/C++ Header 2 KB bsp_cfg.h C/C++ Header 53 KB	<u>^</u>
Kontains data file Move this file Copy this file Publish this file to the Web	bsp_cfg.inc Include File 1 KB Include File 2 KB	
 F-mail this file Delete this file 	bsp_kitl_cfg.h C/C++ Header 2 KB C/C++ Header	
Other Places (*)	bsr Type: C/C++ Header 2 Date Modified: 3/13/2008 5:10 PM Size: 1 2 X B	
 SRC My Documents Shared Documents 	h bsp_twl4030.h c/C++ Header 2 KB h C/C++ Header 1 KB	
9 My Computer 9 My Network Places	h image_cfg.h C/C++ Header 6 KB Include File 2 KB	
Details 🛛 📚	h c/C++ Header 6 KB KB KB KB	
	h oal_kitlserial.h C/C++ Header 3 KB C/C++ Header 2 KB	~
🛃 start 🛛 🕺 2 Microsoft	🔹 🧰 3 Windows 👻 🛄 Tera Term 💿 Microsoft Ac 📑 Lab 6-2_6	EVM 🔇 🕲 4:59 PM

- 13. Select Build | Advanced Build Commands | Build Current BSP and Subprojects from the Visual Studio menu.
- 14. Attach the device and view the output. You should see the following message printed on the Debug Output window during the boot process.

Hello World from IOCTL_HAL _POSTINIT!!!!

Lab 7-1: Integrating a Device Driver

Objectives

Prerequisites

• Completed Lab 2-1

Estimated time to complete this lab: 30 minutes

Lab Setup

To complete this lab, you must have:

- A development workstation running Windows XP
- Visual Studio 2005 (Version 8) with Platform Builder plug-in
- A development workstation running Windows XP
- Visual Studio 2005 (Version 8) with Platform Builder plug-in
- CE 6.0
- CE 6.0 2006 Roll up
- CE 6.0 Service Pack 1
- CE 2007 Updates upto month 9th month {QFEs}
- CE R2
- CE 2007 Updates 11th and 12th month {QFES}
- CE 2008 Updates
- A running image from Lab 2-1

Exercise 1 Integrate barcode scanner driver into BSP

The purpose of this exercise is to integrate a driver into the BSP. In this exercise you will

- Add the driver subdirectory containing the driver source code to the BSP
- Add the appropriate bib entry to cause the driver to be included in the OS image
- Add the appropriate registry entries to cause the drive to be loaded at boot
- Update the BSP catalog file to support the new driver
- Build a debug OS run-time image that we will use in future labs
- > Add driver source code to BSP directory
 - 1. **Detach** from the device if connected.
 - Copy the BARCODE directory from Student files to the C:\WINCE600\PLATFORM\EVMBSP\SRC\DRIVERS directory.



3. In Visual Studio, double click the

C:\WINCE600\PLATFORM\EVMBSP\src\drivers node in the Solution Explorer. This will open the Dirs file.

4. Add the following line to the end of the **Dirs** file directly after the # @CESYSGEN ENDIF CE MODULES DEVICE line:

barcode \

5. Save and close the Dirs file.



Add Driver to image

- 1. Open the **platform.bib** file in the **Parameter Files** node of the **EVMBSP** in the Solution Explorer.
- 2. Add the following lines near the top of **platform.bib** as the first entry in the MODULES section.

```
IF BSP_BARCODE
barcode.dll $(_FLATRELEASEDIR)\barcode.dll NK SHK
ENDIF
```

4 Lab 7-1 Integrating a Device Driver

🏶 EVM_3530 - Microsoft Visual Stu	ıdio	_ 7 🗙
File Edit View Project Build De	bug Target Data Tools Window Community Help	
📴 • 🔤 - 💕 🖬 🗿 🗼 🗈 🛍	💌 🔹 🖓 🔹 📳 🔹 📕 🕨 TI_EVM_3530_ 🔹 Platform Builder (_TGTCPU) 💽 🌌	
· · · · · · · · · · · · · · · · · · ·	ы, Я. I.A. <mark>- (13, 13, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14</mark>	2 \Lambda 🔍 🚽
Device: CE Device 🗸 🗣 😓 🕴		
Solution Explorer - Solution 'EV 👻 🗜 🗙	platform.bib* StdAfx.h	▼ X
B	;	
isitMP3780 MAINSTONEIII MAINSTONEIII Sitmer Files Parameter Files platform.bib platform.dat platform.reg platform.reg SRC SRC Ssc Ssc	<pre> ; MODULES ;Name Path ; IF BSP_BARCODE barcode.dll \$(_FLATRELEASEDIR)\barcode.dll ENDIF ; @CESYSGEN IF CE_MODULES_SHELL </pre>	
	<pre>; Register OMAP specific shell extension omap_shell.dll \$(_FLATRELEASEDIR)\omap_shell.dll ; @CESYSGEN ENDIF CE_MODULES_SHELL ;</pre>	NK
Solutio 👌 Catalo 🐼 Class View		>
Ready	Ln 21 Col 1 Ch 1	INS
🛃 start 📄 🛅 4 Window 🕞	🔗 2 Microsof 🔹 🛄 Tera Term 🛞 Microsoft A 🔛 Lab 7-1_EV 🔇 🕑 🔇	🦻 6:04 PM

When you are done, the top of the file should look similar to the following:

```
; Copyright (c) Microsoft Corporation. All rights reserved.
; Use of this source code is subject to the terms of the Microsoft end-user
; license agreement (EULA) under which you licensed this SOFTWARE PRODUCT.
; If you did not accept the terms of the EULA, you are not authorized to use
; this source code. For a copy of the EULA, please see the LICENSE.RTF on your
; install media.
MODULES
  Name Path
; Name
                                                         Memorv Tvpe
                                                          _____
IF BSP_BARCODE
barcode.dll
                      $( FLATRELEASEDIR) \barcode.dll
                                                            NK SHK
ENDIF
; @CESYSGEN IF CE MODULES DISPLAY
IF BSP NODISPLAY !
  TrainingBSP_lcd.dll $([FLATRELEASEDIR)\EVM 3530 lcd.dll NK SHK
; @CESYSGEN IF SHELLW MODULES GX
; @XIPREGION IF MISC TRAININGESP BIB
. . .
```

3. Save and **close** the file.

> Add registry settings

- 1. Open the file **Platform.reg** from the Parameter Files node of the **EVMBSP** using the Solution Explorer.
- 2. Right click on the [HKEY_LOCAL_MACHINE\Drivers\BuiltIn] key and add a new key with the name Barcode.
- 3. Add a **String Value** to the **Barcode** key with the name **Dll** and value **Barcode.dll**.
- 4. Add a second **String Value** to the **Barcode** key with the name **Prefix** and value **BAR**.
- 5. Save and close the file.

Add driver to the catalog

- 1. From the Visual Studio menu, select File | Open | File ... and navigate to the C:\WINCE600\PLATFORM\EVMBSP\CATALOG folder.
- 2. Change the file mask to show Files of type: All Files (*.*)
- 3. Open the EVMBSP.pbcxml file.

Note If no nodes are visible underneath **Catalog** in the Catalog Editor, click the **Show All Catalog Files** button.

- 4. Expand the catalog tree to show the **Device Drivers** node.
- 5. Right click on the **Device Drivers** node and select **Add Catalog Item**. The new item will be placed in the **Third Party** node.
- 6. Set the **Description** field to **Barcode Scanner**.
- 7. Set the **Title** field to **Barcode Scanner**.
- 8. Set the Unique Id field to Item:GeneriCo:BarcodeScanner.
- 9. Set the Additional Variables field to BSP_BARCODE.
- 10. Set the **Modules** field to **barcode.dll**.
- 11. Save and close the file.

> Add barcode scanner driver to image

- 1. Switch to the Catalog Items View and refresh the Catalog.
- 2. Expand the EVMBSP node under Third Party.
- **Note** Ensure that the Filter option is set to All Items in the Catalog. The Filter option is a drop down box in the upper right hand corner of the Catalog Items View.
 - 3. Select the **Barcode Scanner** item under **Device Drivers**. Refresh the Catalog View if necessary to see the Barcode Scanner.
 - 4. Select Build | Advanced Build Commands | Build Current BSP and Subprojects from the Visual Studio menu.

Verify integration using Image Viewer

- 1. Open the NK.bin file located in the flat release directory using Visual Studio. This will bring up the **Run-Time Image Viewer**.
- 2. Click on the (All Files) node in the Image Explorer. This shows all files that are built into the OS run time image.
- 3. Verify that **barcode.dll** is listed.

- 4. Verify that the **[HKLM\Drivers\BuiltIn\Barcode]** key exists under the registry node.
- 5. Close the Image Viewer.

* Build a Debug OS image

- 1. Select Build | Configuration Manager... using the Visual Studio menu.
- 2. Set the Active solution configuration to EVMBSP ARMV4I Debug.
- 3. Remove the following subprojects from the OS Design by right-clicking on the subproject in the Solution Explorer view and selecting **Remove**:
 - MyHelloWorldApp
 - HeapTest1
 - LeakingMemory
 - ThreadSynchronization
 - MutexDemo
 - EventDemo
 - SemaphoreDemo
 - Power_Management
 - Troubleshoot_Build
- 4. Right-click EVMOSDesign in the Solution Explorer view and select Properties.
- 5. Under the Configuration Properties section, select Environment
- 6. Click New and set the environment Variable Name to BSP_DSPLINK and the Variable Value to 0

Environment Variable	<
⊻ariable name:	
BSP_DSPLINK	
V <u>a</u> riable value:	
0	
OK Cancel)

- 7. Click **OK** to close the **Environment Variable** dialog box and **OK** to close the **EVMOSDesign Property Pages** window
- 8. Select **Build** | **Build Solution** using the Visual Studio menu. This will build a debug configuration that we will use in future labs.

Lab 7-2: Debugging the Scanner Device Driver

Objectives

- Understand driver interaction with application
- Use kernel debugger to investigate call stack

Prerequisites

- Completed Lab 2-1
- Completed Lab 7-1

Estimated time to complete this lab: 30 minutes

Lab Setup

To complete this lab, you must have:

- A development workstation running Windows XP
- Visual Studio 2005 (Version 8) with Platform Builder plug-in
- A development workstation running Windows XP
- Visual Studio 2005 (Version 8) with Platform Builder plug-in
- CE 6.0
- CE 6.0 2006 Roll up
- CE 6.0 Service Pack 1 with CE 2007 Updates upto month 9th month {QFEs}
- CE R2
- CE 2007 Updates 11th and 12th month {QFES}
- CE 2008 Updates
- A running image from Lab 2-1
- Image from Lab 7-1
- Debug build for the EVMOSDesign

Exercise 1 Application and Driver Integration

In this exercise you will add an application that communicates with the barcode scanner device driver. You will exercise the functionality of the driver and function call tree that results when the application calls into the driver.

- > Add BarcodeTest1 application subproject to your OSDesign
 - 1. Copy the **BarcodeTest1** folder from your Student files to C:\WINCE600\OSDesigns\EVMOSDesign\EVMOSDesign
 - 2. Right click on the **Subprojects** node in the Solution Explorer and select **Add Existing Subproject...**



3. Add the **BarcodeTest1** subproject to your OS Design.

- 4. Configure the **BarcodeTest1** subproject to be **excluded from the image** and **always build and link as debug**, as documented in Lab 2-2.
- 5. Right click on the **BarcodeTest1** subproject in the Solution Explorer and select **Build.**

Output 🛛
Show output from: Build 💽 😽 🚽 🖓 🖓 😨
Build started: Project: EVM_3530, Configuration: TI_EVM_3530_ARMV4I Release Platform Buil E:\WINCK600\OSDesigns\SampleOSDesign\BarcodeTestl\sources
starting Build: set WINLEREL=166Duild66makeimg
BUILD: [Thrd:Sequence:Type] Message
BUILD: [00:000000000:PROGC] Build started with parameters:
BUILD: [00:0000000001:PROGC] Build started in directory: E:\WINCE600\0SDesigns\Sample0SDesign\B
BUILD: [00:000000002:PROGC] Checking for E:\WINCE600\sdk\bin\i386\srccheck.exe.
BUILD: [00:000000003:PROGC] Running passes WCEFILESO, MIDL, MC, ASN, THUNK, PRECOMPHEADER, COM
BUILD: [00:000000004:PROGC] Computing include file dependencies:
BUILD: [00:000000005:PROGC] Checking for SDK include directory: E:\WINCE600\sdk\CE\inc.
BUILD: [00:000000006:PROGC] Scan E:\WINCE600\0SDesigns\Sample0SDesign\BarcodeTestl\
BUILD: [00:000000007:PROGC] Saving E:\WINCE600\05Designs\Sample05Design\BarcodeTestl\Build.dat
BUILD: [00:0000000011:PROGC] Building PRECOMPHEADER Pass in E:\WINCE600\0SDesigns\Sample0SDesig
BUILD: [01:000000026:PROGC] Create precompiled header StdAfx.h obj\ARMV4I\retail\StdAfx.obj E: 🗸

Run test application on OS image

6. Attach the device by selecting **Target** | **Attach Device** from the Visual Studio menu.

Note This lab uses an updated version of the OS run time image. You will need to first detach from the existing device instance if it is still running.

- 7. Open the **BarcodeTest1.cpp** file in the BarcodeTest1 subproject using the Solution Explorer.
- 8. Set a breakpoint on the call to **DeviceIoControl**().
- 9. Run the **BarcodeTest1** application using **Target** | **Run Programs...** from the Visual Studio menu. The debugger will halt execution at the breakpoint.
- 10. Select **Debug** | **Windows** | **Call Stack** from the Visual Studio menu to show the call stack. This window shows the sequence of calls that resulted in the statement containing the breakpoint. You can double click any of the calling functions to view the source code file containing each function.
- **Note** The source code for the functions listed in this window is only available if you have installed the Shared Source. Only the disassembly view is available if the source code is not installed.
 - 11. Step through the application by pressing **F10** through completion.
Add additional functionality to test application and retest

12. Locate the comment // Turn on power and add the following function call:

```
// Turn on power
DeviceIoControl(hBARPort, BARCODE_IOCTL_POWER_ON, NULL, 0, NULL,
0, &dwNumBytesRead, NULL);
```

13. Locate the comment // Check to make sure power is on and add the following function call:

```
// Check to make sure power is on
DeviceIoControl(hBARPort, BARCODE_IOCTL_QUERY_POWER_STATE, NULL,
0, &dwResult, sizeof(DWORD), &dwNumBytesRead, NULL);
_tprintf(_T("Power Status = %d.\n"),dwResult);
```

- 14. Right click on the BarcodeTest1 subproject and select Build.
- 15. Run the **BarcodeTest1** application using **Target** | **Run Programs...** from the Visual Studio menu. The debugger will halt execution at the breakpoint.

16. Press F5

17. Observe debug messages in the Output window similar to the following:

Test BAR1: driver open/close. Barcode.DLL: +BAR Open Barcode.DLL: -BAR Open CreateFile returned a valid handle. Barcode.DLL: +BAR IOControl Barcode.DLL: IOCTL - Set Power Management Barcode.DLL: -BAR IOControl Barcode.DLL: +BAR IOControl Barcode.DLL: IOCTL - Power On Command. Barcode.DLL: +BAR PowerUp Barcode.DLL: -BAR PowerUp Barcode.DLL: -BAR IOControl Barcode.DLL: +BAR_IOControl Barcode.DLL: IOCTL - Query Power State Barcode.DLL: -BAR IOControl Power Status = 1. Barcode.DLL: +BAR IOControl Barcode.DLL: IOCTL - Read Barcode. Barcode.DLL: -BAR IOControl Driver: bytes read=7. Driver: buffer=' 0 0 2 5 2 3 Barcode.DLL: +BAR Close Barcode.DLL: -BAR Close

Lab 7-3: Using Debug Zones in a DLL

Objectives

• Learn to implement debug zones in a dll

Prerequisites

- Completed Lab 2-1
- Completed Lab 5-1
- Completed Lab 7-1
- Completed Lab 7-2

Estimated time to complete this lab: 30 minutes

Lab Setup

To complete this lab, you must have:

- A development workstation running Windows XP
- Visual Studio 2005 (Version 8) with Platform Builder plug-in
- A development workstation running Windows XP
- Visual Studio 2005 (Version 8) with Platform Builder plug-in
- CE 6.0
- CE 6.0 2006 Roll up
- CE 6.0 Service Pack 1 with CE 2007 Updates upto month 9th month {QFEs}
- CE R2
- CE 2007 Updates 11th and 12th month {QFES}
- CE 2008 Updates
- A running debug image from Lab 7-2

Exercise 1 Integrate debug zones

In this exercise, you will implement debug zones in the ScanBarcode dll, and test the implementation using the BarcodeDllTest application. This exercise requires a debug OS run-time image on the EVM Board.

- Create the debug zones
 - 1. Right click the ScanBarcode subproject in the Solution Explorer View and select Add | New Item....
 - 2. In the Add New Item Dialog box, select Header File(.h) and name the file DbgZones.h.
 - 3. Add the following code snippet to the new DbgZones.h file:

#include	e <dbgapi.h></dbgapi.h>	
#define	DEBUGMASK(n)	(0x0000001< <n)< th=""></n)<>
#define	MASK INIT	DEBUGMASK(0)
#define	MASK ON	DEBUGMASK(1)
#define	MASK OFF	DEBUGMASK(2)
#define	MASK SCAN	DEBUGMASK(3)
#define	MASK_WARN	DEBUGMASK(14)
#define	MASK ERROR	DEBUGMASK(15)
#define	ZONE INIT	DEBUGZONE(0)
#define	ZONEON	DEBUGZONE(1)
#define	ZONE_OFF	DEBUGZONE (2)
#define	ZONE_SCAN	DEBUGZONE(3)
#define	ZONE WARN	DEBUGZONE(14)
#define	ZONE_ERROR	DEBUGZONE (15)

Instantiate the DBGPARAM structure

- 4. Open the ScanBarcode.cpp file in the ScanBarcode subproject.
- 5. Add the following code snippet just after the #include Power_Status.h.

```
#include "DbgZones.h"
DBGPARAM dpCurSettings =
{
    TEXT("ScanBarcode"),
    {
        TEXT("Init"), TEXT("PwrOn"), TEXT("PwrOff"), TEXT("Scan"),
        TEXT("na"), TEXT("na"), TEXT("na"), TEXT("na"),
        TEXT("na"), TEXT("na"), TEXT("na"),
        TEXT("na"), TEXT("na"), TEXT("na"),
        TEXT("na"), TEXT("na"), TEXT("Na"), TEXT("PwrOff"), TEXT("PwrOff"), TEXT("Na"),
        TEXT("na"), TEXT("Na"), TEXT("Na"),
        TEXT("Na"), TEXT("Na"),
        TEXT("Na"), TEXT("Na"),
        TEXT("Na"), TEXT("Na"),
        TEXT("Na"), TEXT("Na"),
        TEXT("Na"), TEXT("Na"),
        TEXT("Na"), TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na"),
        TEXT("Na")
```

Register the Debug Zones

6. Add the following code snippet to the **DllMain()** function just before the return statement.

```
if(ul_reason_for_call==DLL_PROCESS_ATTACH)
{
     DEBUGREGISTER((HMODULE)hModule);
}
```

- Add debug messages to the dll
 - 7. Add the following debug message to the **DllMain()** function just after the DEBUGREGISTER macro:

```
DEBUGMSG(ZONE_INIT,(_T("ScanBarcode: Initialized!!\r\n")));
```

8. Add the following debug message to the **ScanBarcode()** function just before the return statement:

```
DEBUGMSG(ZONE_SCAN, (_T("ScanBarcode: Scanned!!\r\n")));
```

9. Add the following debug message to the **ScanPowerOn()** function just before the return statement:

```
DEBUGMSG(ZONE_ON,(_T("ScanBarcode: Power ON!!\r\n")));
```

10. Add the following debug message to the **ScanPowerOff()** function just before the return statement:

```
DEBUGMSG(ZONE_OFF,(_T("ScanBarcode: Power OFF!!\r\n")));
```

11. Save and close ScanBarcode.cpp and DbgZones.h.

```
> Build the DLL
```

12. Right click on the **ScanBarcode** subproject in the Solution Explorer and select **Build.**

Test the application

- 13. Launch the **BarcodeDllTest.exe** application using **Target** | **Run Programs** from the Visual Studio menu.
- 14. Observe the debug message output when you use the Power and Scan buttons.
- 15. Select Target | CE Debug Zones... from the Visual Studio menu.

- 4 Lab 7-3 Using Debug Zones in a DLL
 - 16. Scroll down and click on scanbarcode.dll.
 - 17. Click on the Scan check box to remove the check, and click OK.
 - 18. Select the Scan button again in the BarcodeDllTest application.
 - 19. Observe that the **ScanBarcode: Scanned!!** debug message is no longer being displayed. This demonstrates the ability to control message output using debug zones.
- **Note** If you do not have the ability to configure debug zones in a particular module when using the Target | CE Debug Zones menu, it is probably because there were no debug zones registered in that particular module.
 - 20. Close the **BarcodeDllTest.exe** application.

Change to Release configuration

- **Note** We will change back to the Release configuration now for better performance in the remaining labs.
 - 21. **Detach** the device.
 - 22. Select Build | Configuration Manager from the Visual Studio menu
 - 23. Select **EVMBSP ARMV4I Release** from the **Active solution configuration** drop down box.
 - 24. Click on Close.

Lab 8-1: Adding a Catalog Item

Objectives

- Understand how the Catalog works in Windows Embedded CE 6.0
- Be able to add items to the catalog

Prerequisites

• Completed Lab 2-1

Estimated time to complete this lab: 20 minutes

Lab Setup

To complete this lab, you must have:

- A development workstation running Windows XP
- Visual Studio 2005 (Version 8) with Platform Builder plug-in
- A development workstation running Windows XP
- Visual Studio 2005 (Version 8) with Platform Builder plug-in
- CE 6.0
- CE 6.0 2006 Roll up
- CE 6.0 Service Pack 1
- CE 2007 Updates upto month 9th month {QFEs}
- CE R2
- CE 2007 Updates 11th and 12th month {QFES}
- CE 2008 Updates
- A running image from Lab 2-1

Exercise 1 Add an item to the catalog

In this exercise you will create and add the CoreCon ARMV4I Files Helper catalog item to your Windows Embedded CE 6.0 installation. This catalog item will activate a subproject that brings in the files necessary to support application development with Visual Studio. The catalog item can be added to any ARMV4I based BSP.

The subproject that implements the component has already been created and is in your Student files. This subproject simply copies the appropriate binaries into the flat release directory using a batch file. It also adds bib file entries so that the binaries are included in the OS run-time image. See the files postlink.bat and CoreCon_Armv4i.bib for details.

> Create 3rdParty area in WINCE600 tree

- 1. Navigate to C:\WINCE600 with Windows Explorer.
- 2. Create a directory called 3rdParty (no spaces) in C:\WINCE600.
- 3. Create a directory called GeneriCo in C:\WINCE600\3rdParty.
- 4. Create a directory called **Catalog** in **C:\WINCE600\3rdParty\GeneriCo**.
- **Note** The 3rdParty area we created above is the standard location for vendors to add their own functionality other than BSPs. In our case, we are the vendor GeneriCo. The Platform Builder plugin for Visual Studio will look in WINCE600\3rdParty*\Catalog for any catalog files that could add items to the catalog. This provides a consistent mechanism for vendors to add their own functionality.

> Copy CoreCon_ARMV4I subproject to 3rdParty area

5. Copy the CoreCon_ARMV4I folder from your Student files to our 3rdParty folder at C:\WINCE600\3rdParty\GeneriCo.

Note This subproject is only implemented for ARMV4I CPUs. It directly includes the ARMV4I binaries. It could be modified to support all CPU types generically.

Create a new Catalog Item

- 6. In Visual Studio, select File | New | File... to open the New File dialog.
- 7. In the Categories tree, select **Platform Builder**.
- 8. In the Templates list, select Platform Builder Catalog File.

- 9. Click Open. A new catalog file will open in the Visual Studio editor.
- 10. Right click on the node **Catalog [Current file]** and select **Add Catalog Item in Subfolder**.



- 11. Name the new folder GeneriCo and select OK.
- 12. The new Item will appear under the **Third Party** | **GeneriCo** node in the Catalog. **Right click** on the item and choose **Properties**. This will bring up the Properties window for this catalog item.



- 13. Type CoreCon Files Helper for ARMV4I in the Comment block in the Identification section.
- 14. Type **CoreCon Files Helper for ARMV4I** in the **Description** block in the **Identification** section.

- 15. Type **CoreCon Files Helper for ARMV4I** in the **Title** block in the **Identification** section.
- 16. Type Item:GeneriCo:CoreCon_ARMV4I in the Unique Id block in the Identification section
- 17. Type **SYSGEN_CORECON_ARMV4I** in the **Sysgen Variable** block in the **Item** section.
- 18. Click in the data area for **Subproject Links** in the **Projects** section, then click on the ... button on the right hand side. This will bring up the **PbpXml Project Links** dialog.
- 19. Click Add
- 20. Navigate to the C:\WINCE600\3rdParty\GeneriCo\CoreCon_Armv4i folder and select CoreCon_Armv4i.pbpxml.
- 21. Click **OK** to close the dialog. The final Properties window should look like the following:

Properties		×
CoreCon Files Helper for A	ARMV4I Catalog Item	-
₽		
Compatibility		-
Supported CPUs		
Type	General	-
E General	UCHCHI	
Help Link		
Sdk Help File Attribute		_
Sdk Help Files		_
Size	0	-
Size Is Scaled	True	-
Identification		
Comment	CoreCon Files Helper for ARMV41	-
Description	CoreCon Files Helper for ARMV41	-
Title	CoreCon Files Helper for ARMV41	_
Unique Id	Item:GeneriCo:CoreCon_ARMV4I	
E Item		
Additional Variables		
Modules		_
Notification Html		
Notification Title		
Systen Variable	SYSGEN CORECON ARMV41	_
Choose One Group	False	
Locations	(\GeneriCo.)	
Projects	((contract))	
Subproject Links	\$(_WINCEROOT)\3RDPARTY\GENERICO\CORECON_ARMV4I\CORECON_ARMV4I.PBPXML	
Subproject Links Zero or more. Path to *.pbpx variables.	ml files that should be built and included in the image if this catalog item is selected. May contain environmen	t

- 22. Save the Catalog File to C:\WINCE600\3rdParty\GeneriCo\Catalog, with the name CoreCon_ARMV4I.PbcXml.
- 23. Switch to the Catalog Items View if it is not already open.
- 24. Refresh the **Catalog Items View** by clicking on the refresh button located on the command bar.

Catalog Items View	×
Filter - 😰 <search></search>	- 🗈
 EVM_3530 BSP Core OS Device Drivers Third Party BSP 	
🖏 Solution Explorer 💩 Catalog Items View 🐼 Class View	

- 25. Expand the new GeneriCo node under Third Party, and observe the new CoreCon Files Helper for ARMV4I item.
- 26. Select the **CoreCon Files Helper for ARMV4I**. A green check mark should appear in the box indicating the item has been added to your OS Design.
- 27. Switch to the Solution Explorer view.
- 28. Observe that the **CoreCon_ARMV4I** subproject has been automatically added to your design. This OS Design will now include the binaries necessary to support application debugging with Visual Studio.

Note Do not exclude this subproject from the build. Its purpose is to include files into the build.

29. Select **Build** | **Advanced Build Commands** | **Build Current BSP and Subprojects** from the Visual Studio menu. The new OS run-time image will include the CoreCon helper files.

Lab 8-2: Replace the Standard Explorer Shell with IESHELL

Objectives

• Understand how to implement a custom shell

Prerequisites

• Completed Lab 2-1

Estimated time to complete this lab: 45 minutes

Lab Setup

To complete this lab, you must have:

- A development workstation running Windows XP
- Visual Studio 2005 (Version 8) with Platform Builder plug-in
- A development workstation running Windows XP
- Visual Studio 2005 (Version 8) with Platform Builder plug-in
- CE 6.0
- CE 6.0 2006 Roll up
- CE 6.0 Service Pack 1
- CE 2007 Updates upto month 9th month {QFEs}
- CE R2
- CE 2007 Updates 11th and 12th month {QFES}
- CE 2008 Updates
- A running image from Lab 2-1

Exercise 1 IESHELL

In this exercise you will clone the IESimple browser application into a new subproject and call it IESHELL. The IESimple application is a simple container around the IE browser object. This application is sometimes used as the starting point to write custom browser based shells.

You will run the application and verify its functionality. You will use this application in the next exercise as a replacement for the Standard Shell.

Clone IESIMPLE

- 1. Using the Solution Explorer, create an empty subproject of type WCE Application with the name IESHELL.
- Navigate to C:\WINCE600\PUBLIC\IE\OAK\IESIMPLE and copy all the files except sources and makefile to your new IESHELL subproject directory at C:\WINCE600\OSDesigns\EVMOSDesign\EVMOSDesign\IESHELL.
- 3. Rename **iesimple.rc** to **ieshell.rc** in your new subproject directory.
- 4. Add the newly copied files by right clicking on the **IESHELL** subproject and selecting **Add** | **Existing Item...** Add the files **mainwnd.h**, **resource.h**, **ieshell.rc** and **mainwnd.cpp**.
- 5. Right click on the **IESHELL** subproject and select **Open**. The **SOURCES** file will open in the Visual Studio editor.
- 6. Add the following INCLUDES directive to the bottom of the file. Ensure there is a blank line between the INCLUDES directive and the line above it.

```
INCLUDES= \
  $ (_WINCEROOT) \PUBLIC\IE\SDK\INC; \
  $ ( WINCEROOT) \PUBLIC\COMMON\OAK\INC \
```

7. Add the following libraries immediately after the last library listed in the TARGETLIBS directive. Ensure there is a blank line after the last entry.

```
$(_PROJECTROOT)\cesysgen\sdk\lib\$(_CPUINDPATH)\wininet.lib \
$(_PROJECTROOT)\cesysgen\sdk\lib\$(_CPUINDPATH)\commctrl.lib \
$(_PROJECTROOT)\cesysgen\sdk\lib\$(_CPUINDPATH)\uuid.lib \
$(_PROJECTROOT)\cesysgen\sdk\lib\$(_CPUINDPATH)\ole32.lib \
$(_PROJECTROOT)\cesysgen\sdk\lib\$(_CPUINDPATH)\oleaut32.lib \
```

- 8. Save and close the file.
- 9. Right click on the IESHELL subproject and select Build.

> Test IESHELL

- 10. Attach to the device if not already attached.
- 11. Launch **ieshell.exe** using **Target** | **Run Programs** from the Visual Studio menu. The default home page will appear.
- **Note** You may not be able to access the internet using the IESHELL browser application. There are a number of issues that can limit connectivity. The device must be configured for Internet access, there must be a virtual network driver installed on your development system, you must have Internet connectivity available at your location etc.
 - 12. Press **Ctrl + G** to bring up an address dialog box. You may type other web addresses in this dialog to navigate to other sites.

> Terminate IESHELL

- 13. Select **Target | Target Control** from the Visual Studio menu to bring up the Target Control utility.
- 14. Type **gi proc** at the Windows CE prompt to determine the ID of the ieshell.exe process.
- 15. Terminate the ieshell process using the **kp** command at the Windows CE prompt.

Exercise 2 Configure IESHELL as the shell

In this exercise you will configure IESHELL to run as the default shell application instead of the Standard Shell. We will rebuild the OS run-time image to include this new component.

Note You should normally remove the Standard Shell from your OS design if you are going to use a different application as the shell. We are not going to remove the Standard Shell in this exercise so that we do not have to rebuild the OS design.

Detach the device

1. Select **Target** | **Detach Device** from the Visual Studio menu.

Configure IESHELL to launch at boot

- 2. Open the ieshell.reg file in the IESHELL subproject using the Solution Explorer.
- 3. Add a new key called **Init** to **HKEY_LOCAL_MACHINE**.
- 4. Add a new **String Value** to the Init key called **Launch50** with the value **ieshell.exe**.
- 5. Add a new **Binary Value** to the Init key called **Depend50** with the value **14 00 1e 00**.



Note These registry settings will override existing settings that are provided by the Standard Shell. They will cause ieshell.exe to be launched automatically during the boot process. The settings we provide here take precedence because registry entries from subprojects are processed last during the build.

Add SignalStarted() to ieshell

6. Open mainwin.cpp from the IESHELL subproject

7. Add the following code near line 171 to handle the **SignalStarted()** call. There should be a PeekMessage statement followed by an "if" logic statement. The code will need to go between these to points.

```
int initSignal = wtol(lpCmdLine);
if(initSignal != \overline{0})
{
        SignalStarted(initSignal);
        if (FAILED(HandleNewWindow2(T(""),NULL)))
        {
                 goto Cleanup;
         }
}
else
{
        // EXISTING CODE HERE
        if(FAILED(HandleNewWindow2(lpCmdLine, NULL)))
        {
                goto Cleanup;
        }
}
```

- 8. Select Build | Advanced Build Commands | Build Current BSP and Subprojects from the Visual Studio menu.
- > Test
 - 9. Open the OS run-time image file (NK.BIN) from the flat release directory. The Run-Time Image viewer will load.
 - 10. Verify that the [HKLM\Init] key contains ieshell.exe and not explorer.exe

IK.bin	Name	Туре	Data
NK	Default	REG_SZ	(value not set)
Boot Registry	Depend20	REG_BINARY	0a 00
	Depend30	REG_BINARY	14 00
	Depend50	REG_BINARY	14 00 1e 00
	Depend60	REG_BINARY	14 00
	Depend98	REG_BINARY	14 00
	Launch10	REG_SZ	shell.exe
	Launch20	REG_SZ	device.dll
🗉 🧰 Explorer	🗎 Launch30	REG_SZ	gwes.dll
🗉 🛅 ExtModems	📃 Launch50	REG_SZ	ieshell.exe
🗉 🚞 HARDWARE	📄 Launch60	REG_SZ	servicesStart.exe
Ident	🗐 Launch98	REG_SZ	EmulatorStub.exe
🖨 🗁 init			
BootVars			

- 6 Lab 8-2 Replace the Standard Explorer Shell with IESHELL
 - 11. **Attach** to the device. Observe that the default shell is now ieshell and not the Standard Shell.

Note We want the Standard Shell for future labs. So we'll remove ieshell from the OS design here.

- 12. **Detach** the device
- 13. Right click on the IESHELL subproject in the Solution Explorer and select **Remove**
- 14. Select Build | Advanced Build Commands | Build Current BSP and Subprojects from the Visual Studio menu.

Lab 8-3: Exporting an SDK

Objectives

• Be able to create an SDK for native code development in Visual Studio 2005

Prerequisites

• Completed Lab 2-1

Estimated time to complete this lab: 15 minutes

Lab Setup

To complete this lab, you must have:

- A development workstation running Windows XP
- Visual Studio 2005 (Version 8) with Platform Builder plug-in

Exercise 1

In this exercise you will create an SDK based on your OS Design. The SDK can be installed by application developers using Visual Studio to target applications to your device.

> Add New SDK

- 1. Select **Project** | **Add New SDK...** from the Visual Studio menu.
- 2. Click General in the left window.
- 3. Change SDK Name to myTrainingSDK.
- 4. Fill in **Product Name**, **Company Name**, and **Company Website** with appropriate values.
- 5. Add Major, Minor, and Build numbers.

Note You should increment the build number every time you create a new version of the SDK. The installer uses this version information to compare different installations of the SDK

SDK1 Property Pages		?×
General Install License Terms Readme CPU Families Development Languages Additional Folders Emulation	SDK Name: My TrainingSDK Product Name: My Product Name Product Version: Major: 01 Minor: 00 Build: 0000 Company Name: Generico Company Website: www.generico.com	
	OK Cancel	Apply

6. Select Install in the left window.

- 7. Make note of the **MSI Folder Path**.
- 8. In the MSI File Name box type myTrainingSDK.msi.

SDK1 Property Pages		? 🛛
General Install License Terms Readme CPU Families Development Languages Additional Folders Emulation	MSI Folder Path: E:\WINCE600\DSDesigns\SampleOSDesign\EVM_3530\SDKs\SDK1\MSI MSI File Name: my trainingSDKImsi Locale: U.S. English	
	OK Cancel	

9. Select **CPU Families** in the left window. SDK Property Pages dialog should appear as follows.



10. Select **Development Languages** in the left window. Note that if you have added the .NET Compact Framework to your OS Design, you will have the option to check the **Managed development support** checkbox.

SDK1 Property Pages		? 🛛
General Install License Terms Readme CPU Families Development Languages Additional Folders Emulation	Select the development languages that you want your SDK to support. Native development support Platform-specific macro (optional): Managed development support)	
	OK Ca	ncel <u>A</u> pply

11. Select **Additional Folders** in the left window. This dialog allows you to add custom folders to your SDK.

SDK1 Property Pages				? 🛛
General Install License Terms Readme CPU Families Development Languages Additional Folders Emulation	You can choose to incl fully-qualified path to a installation directory. Include Subfolders	ude additional directories in ocal directory. The target di Source Folder Add	your SDK. The source directory rectory is a path relative to the Target Folder Edit Dele	is a
			OK Cancel	

12. Click OK.

13. The new SDK will appear in the Solution Explorer in the SDKs node.

Build the new SDK

14. In Solution Explorer under the SDKs folder, right-click myTrainingSDK and select **Build**. Once the build is completed, the SDK can be installed from the MSI file created in the MSI Folder Path.



6 Lab 8-3 Exporting an SDK

Output	×
Show output from: Build 🗸 😽 🚽 🖓	
adding E:\WINCE600\0SDesigns\Sample0SDesign\EVM_3530\SDKs\SDKl\obj\My TrainingSDK\Lib\ARMV4 adding E:\WINCE600\0SDesigns\Sample0SDesign\EVM_3530\SDKs\SDKl\obj\My TrainingSDK\Lib\ARMV4 adding E:\WINCE600\0SDesigns\Sample0SDesign\EVM_3530\SDKs\SDKl\obj\My TrainingSDK\MSManifes	^
Cabinet file successfully created: E:\WINCE600\OSDesigns\SampleOSDesign\EVM_3530\SDKs\SDK1\obj\s	
Exported SDK to: E:\WINCE600\OSDesigns\SampleOSDesign\EVM_3530\SDKs\SDK1\MSI\my trainingSDK.msi	
E:\WINCE600\OSDesigns\SampleOSDesign\EVM_3530\SDKs\SDK1\SDK1.sdkcfg - 0 error(s), 0 warning(s) ====================================	~

Lab 9-1: Developing with Managed Code

Objectives

• Learn to develop and debug managed applications in a separate Visual Studio 2005 instance

Prerequisites

- Completed Lab 2-1
- Completed Lab 8-1

Estimated time to complete this lab: 30 minutes

Lab Setup

To complete this lab, you must have:

- A development workstation running Windows XP
- Visual Studio 2005 (Version 8) with Platform Builder plug-in
- Visual Studio 2005 Service Pack 1
- CE 6.0
- CE 6.0 2006 Roll up
- CE 6.0 Service Pack 1
- CE 2007 Updates upto month 9th month {QFEs}
- CE R2
- CE 2007 Updates 11th and 12th month {QFES}
- CE 2008 Updates
- .NET Compact Framework 2.0 Service Pack 1 Patch

Exercise 1 Create a managed application project

In this exercise you will create a managed application project in a separate instance of Visual Studio targeting your Windows Embedded CE 6.0 device. You will deploy this application to the running device and debug it in the next exercise.

Create a new Managed Project

- 1. Start a **new instance** of Visual Studio (NOT the same instance that contains your Windows Embedded CE 6.0 OS Design; leave that instance running).
- 2. Select File | New Project ... from the Visual Studio menu.
- 3. In the New Project window select Visual C# | Smart Device | Windows CE 5.0.
- 4. Select the **Device Application** template.
- 5. Name your project HelloWorld, and click OK.

Project types:	Templates:	
 Visual C++ Other Languages Visual C# Windows Smart Device Pocket PC 2003 Smartphone 2003 Windows CE 5.0 Database Starter Kits Web Distributed System Solutions Other Project Types Platform Builder for CE 6.0 	Visual Studio installed templates Device Application Class Library Control Library Console Application Empty Project Console Application My Templates Search Online Templates	
A project for creating a .NET Compac	t Framework 2.0 forms application for Windows CE 5.0 and later	



- **Note** The Windows CE 5.0 option in the Smart Device category applies to both Windows CE 5.0 and Windows Embedded CE 6.0 development. There is no difference between the two OS versions with regard to managed code development in Visual Studio 2005.
- Add controls to the form
 - 6. Double click on Form1.cs in the HelloWorld project in the Solution Explorer.

4 Lab 9-1 Developing with Managed Code

🍘 HelloWorld - Microsoft Visual St	tudio				-7×
File Edit View Project Build De	ebug Target Data	Format Tools	Window Community	Help	
🔚 • 🛅 • 📂 🛃 🗿 🖉 🗎	L ら - ら - 値 -	🖳 🕨 Debug	 Any CPU 	- 🖄	
単 阜 冬	前 昭 尊 🏎 2	* 약* 약* 출 축	다 음부 🙄 🕴 Pocket PC 2	003 SE Emulator	- 📴 🎫 🗛 🖕
Device: 🔽 🖓 🖓	📼 😭 🖕				
Solution Explorer - Solution 'Hel $ eq \ \ \P \ \ \times$	Form1.cs [Desi	gn] Start Page	Output		- ×
🖶 🗿 👩 🗉 📾 🖧					<u>^</u>
Solution 'HelloWorld' (1 project)	Form1				
en properties					
AssemblyInfo.cs					
Form1.cs					
Program.cs					
	<				
	🖹 mainMenu1				
Beadu					
	TTTTTTTTTTTTT		CEC OL abo		
Start % 2 Microso	lera lerm	Microsoft A	CE6.0 Labs	Eap 9-1_EV	🔨 😈 🕲 3:12 PM

- 7. Delete the **mainMenu1** control at the bottom of the design window. Our application will not have a menu.
- 8. Right click on the form and select **Properties**.
- 9. Change the Text property in the Appearance group to MyManagedApp.
- Expand the Size property in the Layout group. Change the Width and Height to 240.

🌤 HelloWorld - Microsoft Visual Studio						_ 7 🗙
File Edit View Project Build Debug Target	Data Too	Pro	perties		×	
) 🖥 • 🛅 • 💕 🛃 🗿 🐰 🗈 🛍 🔊 • (* •	J - B	Fo	rm1 System.Windows.Forms.Forr	n	-	
	H• → →+		2↓ 🗉 🖋 🖻			p 12 21 📮
Device:		Ξ.	Appearance		^	
Solution Explorer - Solution 'Hel. -1×1	[Decign]*]		BackColor [Control		- ×
	[Design]	±١	Font	Arial, 10pt		• •
			ForeColor	ControlText		<u> </u>
Solution 'HelloWorld' (1 project)			FormBorderStyle I	FixedSingle		
🖃 📲 HelloWorld			Text I	MyManagedApp.		
Properties		Ξ Ι	Behavior		=	
AssemblyInfo.cs			AutoValidate I	EnablePreventFocusChange		
Resources.resx			ContextMenu	(none)		
			Enabled	True		
Second Designer cs			ToolBar	(none)		
Services		Ξ Ι	Data		_	
Program.cs		Ð	(DataBindings)			
			Tag			
			Design			
			(Name) I	Form1		=
			FormFactor	WebPad		=
			Language	(Default)		
			Localizable	False		
			Locked I	False		
			Skin I	False	~	
		He	ight M			~
Ready						
🐮 start 🕺 2 Microso 🔹 🛄 Tera Term	🔞	Micro	osoft A 📄 🗁 CE6.0 Labs	🕎 Lab 9-1_EV 🔇 🜖	0	🔞 3:14 PM

- 11. Close the **Properties** window.
- 12. If the Toolbox is not visible, select **View** | **Toolbox** from the Visual Studio menu.
- 13. Drag a button from the Toolbox onto the center of the form. Size the button to whatever dimensions you wish.
- 14. Right click on the button and select **Properties**.

6 Lab 9-1 Developing with Managed Code

🦇 HelloWorld - Microsoft Visual St	udio			- 7 ×	
File Edit View Project Build De	bug Target Data f	Format Tools Window	Community Help		
🛅 • 🖽 • 💕 🛃 🗿 🐰 🗈 🛍	🖌 🕶 - 🗐 - 🗐	👃 🕨 Debug 🗸 👻	Any CPU	- 🖄 🗒	
[神] [] 후 리 [파 아 프] ; ; ;	[] 昭 中 명 대	않 않 응 찾 당	Pocket PC 2003 SE Em	ulator 🔹 🖳 💻 🗛 🚽	
Device:					
Solution Explorer - Solution 'Hel 4 ×	Form1 cs [Design]	* Start Page Output	- X	Toolbox – 4 ×	
	Torini.cs [Design]			All Device Controls	
Solution HelloWorld' (1 project)				Pointer	
in the source of	0	00		📅 BindingSource	
📮 🗁 Properties	j but	to View Code	1	ab Button	
AssemblyInfo.cs	0			CheckBox	
End Resources.resx		Bring to Front		E ComboBox	
Form1.cs		Send to Back		🗟 ContextMenu	
Form1.Designer.cs		🛱 Align to Grid	ſ	🚰 DataGrid	
- B Form1.resx		Lock Controls		📷 DateTimePicker	
errogram.cs		Colort 'Covert'		🗐 DomainUpDown	
				 ▲> HScrollBar 	
		💑 Cut	=	🗊 ImageList	
		🛅 Сору		📖 InputPanel	
		🖺 Paste		A Label	
		× Delete		A LinkLabel	
				E ListBox	
				222 ListView	
				🛓 MainMenu	
			~	剑 MessageQueue	
	<		>	🛗 MonthCalendar 🗸 🗸	
Ready 1 70, 131 1 72 × 20					
🚜 start 🛛 😻 2 Microso 🔹	🛄 Tera Term (🗕 Microsoft A 🛛 🛅 C	E6.0 Labs 👜 Lab	19-1_EV 🔇 👂 😵 3:15 PM	

- 15. Change the Text property in the Appearance group to Click Me!
- 16. Close the **Properties** window.



17. Double click the button you just added. The **Form1.cs** file will open in the editor with the cursor in the **button1_Click ()** function.

🦇 HelloWorld - Microsoft Visual St	udio	PX					
File Edit View Refactor Project	Build Debug Target Data Tools Window Community Help						
🔚 • 🛅 • 📂 📕 🎒 🗼 🗈 🛍	🔊 🔹 🖓 👻 🖳 🕨 Debug 🔹 Any CPU 🔹 🏄						
Pocket PC 2003 SE Emulator 🔹 🦣 🗿 🛝 🕼 🖕 🔢 📳 💁 🗛 🛊 華 🚊 😫 🗔 🖓 🤜 🚑 👰 🦂							
Device: 🗾 🗸 🗛 🖾 🔛							
Solution Explorer - Solution 'Hel $ eq \ \ \mathbf{q} \ \ \mathbf{X}$	Form1.cs* Form1.cs [Design]* Start Page Output	▼ X					
🖹 📴 🖬 🗉 🖧	HelloWorld.Form1	~					
Solution 'HelloWorld' (1 project) HelloWorld Properties AssemblyInfo.cs References Form1.cs Form1.resx Program.cs	<pre>using System.ComponentModel; using System.Data; using System.Drawing; using System.Text; using System.Windows.Forms; namespace HelloWorld { public partial class Form1 : Form { public Form1() { InitializeComponent(); } private void button1_Click(object sender, EventArgs e) { } private void button1_Click(object sender, EventArgs e) } } } </pre>						
		>					
Ready		INS					
Stant 🕺 2 Microso 👻	🚆 Tera Term 🔞 Microsoft A 🧰 CE6.0 Labs 🕎 Lab 9-1_EV 🔇 🔰 🔇 🔕 S	818 PM					

18. Add the following code snippet to the click handler.

MessageBox.Show ("Hello World!");

19. Right click the **HelloWorld** project in the Solution Explorer and select **Build**. Your managed application is complete.

Exercise 2 Deploy to device

In this exercise you will deploy your application to the device and debug it. The OS runtime image running on the device already contains the helper files necessary to support communication between the device and Visual Studio thanks to the helper component we added in a previous lab.

Determine device IP address

- 1. In the CE6 instance of Visual Studio, **Attach** to the device if not currently attached.
- 2. Open the **Target Control** utility by pressing **Alt+1** in the Platform Builder session of Visual Studio.
- 3. At the Windows CE prompt, type **s ipconfig** /**d**. Note the device IP address.
- **Note** The ipconfig utility was included in our OS Design as a part of the networking utilities. The /d option causes the output of the command to be displayed in the debug Output window where we can easily retrieve it.
- > Configure managed application development environment for deployment
 - 4. In the Visual Studio instance containing your managed application, select **Tools** | **Options** from the menu.
 - 5. In the Options window, expand the **Device Tools** node and select **Devices**.

Options		? 🛛
Import and Export Settings International Settings Keyboard Startup	^	Show devices for platform: Windows CE 5.0
Web Browser		Windows CE 5.0 Device Save As
 Projects and Solutions Source Control Text Editor Database Tools Debugging Device Tools General Devices 	Ξ	Rename Delete Properties
Form Factors HTML Designer Platform Builder for CE Windows Forms Designer		Defaul <u>t</u> device: Pocket PC 2003 SE Emulator
		OK Cancel

- 6. In the Show device for platform: drop down box select Windows CE 5.0.
- 7. Click on Windows CE 5.0 Device and select Properties.
- 8. Click the **Configure** button beside the **Transport** drop down box. We are going to configure the TCP Connect Transport.
- 9. Select the Use specific IP address button, and type in the IP address of the target device.



10. Click **OK** through all of the dialogs.

Prepare the target device

- 11. At the Windows CE prompt in the Target Control utility, type s ConmanClient2.
- 12. Then, type **s cmaccept**. You now have 3 minutes to establish a connection with your managed application.

Note These two utilities were included in the CoreCon File Helper that we previously added to this OS Design.

> Deploy the managed application

- 13. Set a breakpoint in your application on the call to **MessageBox.Show("Hello World!");** in the button1_Click() function in **Form.cs**.
- 14. Select **Debug | Start Debugging** from the Visual Studio menu.
- 15. Select **Windows CE 5.0 Device** from the list of devices in the **Deploy HelloWorld** box and click **Deploy**. Visual Studio will deploy several cab files to the device in addition to your application. Your application will run on the target device.
- 16. Click on the **Click Me!** button in your application, and you will hit the breakpoint you just set. You are now debugging your managed application!

Lab 9-2: Integrating a Managed Application

Objectives

• Learn how to integrate a managed application into the BSP

Prerequisites

- Completed Lab 2-1
- Completed Lab 8-1
- Completed Lab 9-1

Estimated time to complete this lab: 20 minutes

Lab Setup

To complete this lab, you must have:

- A development workstation running Windows XP
- Visual Studio 2005 (Version 8) with Platform Builder plug-in
- Visual Studio 2005 Service Pack 1
- CE 6.0
- CE 6.0 2006 Roll up
- CE 6.0 Service Pack 1
- CE 2007 Updates upto month 9th month {QFEs}
- CE R2
- CE 2007 Updates 11th and 12th month {QFES}
- CE 2008 Updates
- .NET Compact Framework 2.0 Service Pack 1 Patch

Exercise 1 Integrating a managed application

In this exercise you will integrate your managed application directly into the OS run-time image by including it into the BSP instead of deploying it from Visual Studio.

- > Build a Release version of your application
 - 1. Select **Build** | **Configuration Manager** from the Visual Studio menu in the Visual Studio instance that is building your managed HelloWorld project.
 - 2. Select **Release** from the **Active solution configuration** drop down box, and click **Close**.
 - 3. Select **Build | Build Solution** from the Visual Studio menu to build the Release version of your application.
 - 4. Note the output directory for the executable. By default, it will be in your **My Documents** folder in the **Visual Studio 2005\Projects\HelloWorld\bin\Release** subfolder.

Note The project directory for Visual Studio is configurable using the **Options** dialog available from the **Tools** | **Options** menu in Visual Studio.

> Add the managed application to your BSP

 Copy the HelloWorld.exe application from the Visual Studio output directory to the FILES directory of the EVMBSP located at C:\WINCE600\PLATFORM\EVMBSP\FILES.

Note Everything in the **FILES** directory automatically gets copied to the flat release directory during the Build Release Directory phase.

- 6. Open **platform.bib** from the **Parameter Files** node of the EVMBSP using the Solution Explorer.
- 7. Add the following line to the bottom of **platform.bib** in the **FILES** section:

HelloWorld.exe \$(_FLATRELEASEDIR)\HelloWorld.exe NK

Note Managed applications must be included in the FILES section of a .bib file. Do not place a managed application in the MODULES section.

> Add the .NET Compact Framework 2.0 to the OS Design

- **Note** We previously only included the OS dependencies for the .NET Compact Framework 2.0 in our OS Design; we did not include the framework itself. We allowed Visual Studio to deploy the framework to our device during the managed code development process. Now we want the framework on the device so that we can run managed applications without the support of Visual Studio.
 - 8. **Detach** the device. We are going to be rebuilding the OS run-time image.
 - 9. Locate the .NET Compact Framework 2.0 catalog item in the Catalog Items View under Core OS | CEBASE | Applications and Services Development | .NET Compact Framework 2.0.
 - 10. Add the .NET Compact Framework 2.0 catalog item to your OS Design.

Note There are two versions of the .NET Compact Framework 2.0. Be sure to select the one that does **NOT** have the – Headless modifier in the name.

Rebuild the OS Design

11. Select **Build** | **Rebuild EVMOSDesign** from the Visual Studio menu. This will clean our existing design (both Debug and Release) and rebuild the currently selected Release configuration.

Note This will take several minutes to complete, depending on the capabilities of your development workstation.

> Test the managed application

- 12. Attach the device.
- 13. Navigate to the \Windows directory on the device.

14. Double click on HelloWorld.

Your managed application will load and run. You have successfully integrated your managed application into your OS run-time image.
Lab 10-1: Using the CETK

Objectives

- Run automated tests using the Windows Embedded CE Test Kit (CETK)
- Modify the default behavior of the standard tests

Prerequisites

• Completed Lab 2-1

Estimated time to complete this lab: 30 minutes

Lab Setup

To complete this lab, you must have:

- A development workstation running Windows XP
- Visual Studio 2005 (Version 8) with Platform Builder plug-in

Exercise 1 Run a simple CETK test

In this exercise you will learn how to launch the Windows Embedded CE Test Kit. You will run selected tests and observe the results.

- > Launch the Windows Embedded CE Test Kit
 - 1. Ensure that the EVM is attached.
 - 2. Copy the file ktux.dll from the C:\Program Files\Microsoft Platform Builder\6.00\cepb\wcetk\ddtk\armv4i folder on your workstation to the \windows directory on your device.
 - Using the Start Menu on your workstation, select Start | All Programs | Windows Embedded CE 6.0 | Windows Embedded CE 6.0 Test Kit. The Windows Embedded CE Test Kit (CETK) window will appear.
- **Note** The CETK is not available from within the Visual Studio 2005 development environment.
 - 4. Select **Connection** | **Start Client...** from the CETK menu. The Device Connection dialog will appear.

Device Connection	×
Platform Manager	
Click Connect to use Platform Manager to automatically download and start the clientside program.	
Connect	
Use Windows Sockets for the client/server communication.	
Click Settings to configure Windows Embedded CE Platform Manager.	
Settings	
Manual connection Click Manual Connection Info to learn how to manually start the clientside program without using Platform Manager.	
Manual Connection Info	
Close <u>H</u> elp	11

🔥 Windows Embedded CE Tes	t Kit	. ø 🗙
Server Connection View Tests	Device Connection	
Windows Embedded CE Te	Platform Manager Click Connect to use Platform Manages to automaticallu douveload and start the Select a Windows CE Default Platform I us Click S Manual Click N progra OK Cancel	
🐉 start 📄 3 Win 🕙	🔹 😕 2 Micr 🔹 🛄 Tera T 💿 Microso 📑 Lab 10 💽 Windo 🔇 🖲 🧐 🕲	2:28 PM

- 5. Click on Connect.... The Select a Windows CE Device dialog will appear.
- 6. Click **OK** to accept the **Default Device** connection. The CETK server on the development workstation will download client software to the device and connect to it.

🔉 Windows Embedded CE Tes	t Kit	
erver Connection View Tests	Device Connection	
	Platform Manager Cick Connect to use Platform Manager to automatically download and start the clientside program. Connecting to device Device Name: EVM_3530 EVM_3530 Establishing platform manager connection to device Cancel Close Help	

Note The CETK supports multiple connection methods. This allows the test suite to be used in a variety of scenarios. We are using the same connection that we have been using with the Remote Tools. This connection configuration relies on the KITL transport.

> Run selected tests

🗞 Windows Embedded CE Test Kit	_ 7 🗙
Server Connection View Tests Help	
🖃 🚇 Windows Embedded CE Test Kit Server	~
😑 🕵 WindowsCE (ARMY4I)	
🖃 - 🛞 📕 Windows Embedded CE Test Catalog	
₽ <mark></mark> Audio	
Audio Quality Test	
IR Port	
🗉 🦲 Keyboard	
	=
a- 🕞 Mouse	=
🖶 – 🧰 Multimedia	
OAL Cache Tests	
OAL Interrupt Tests	
OAL loct Tests	
Performance Tests	
Exercise Port	
🔤 🕞 Smart Card	
🖬 💽	
🗄 🧰 Touch Panel	~
🛃 Start 🛛 🕺 2 Micro 🔹 🚞 3 Wind 🔹 🔤 Tera Ter 💿 Microsof 💽 Window 📑 Lab 10-1 🤇	🔞 6:23 PM

- 7. Expand the WindowsCE (ARMV4I) node.
- 8. Expand the Windows Embedded CE Test Catalog node to show the test groups.
- 9. Right click on Windows Embedded CE Test Catalog and choose Deselect All Tests.



Note By default, the CETK will select the all the tests it determines are appropriate for the device. We wish to run only a subset of the tests, so it is easier to select them individually.

Performance Tests Printer Serial Port Smart Card Storage Device Touch Panel USB Port USB Port	
	~
WindowsCE (ARMV4I)	

10. Expand the Touch Panel node and select Touch Panel Test.



11. Expand the **Other Tests** node and select **Battery API Test**. Note that the list is not in alphabetical order.



12. Select **Tests** | **Start/Stop Tests** | **WindowsCE (ARMV4I)** from the CETK menu. The CETK will indicate which test is currently running.

Note Some tests are fully automatic, others require user interaction.

- 13. Follow the instructions on the EVM screen to complete the tests that require user input.
- 14. Switch to Visual Studio and view the testing progress in the Output window. You will see details of what the test is doing in addition to the results. These same results will be available from within the CETK once the tests are complete.

Output	X
Show output from: Windows CE Debug 🔹 💀 🎝 🗐 😨	
1865986 PID:411000a TID:412000a ***	~
1865986 PID:411000a TID:412000a *** Test Name: Pen Up / Down Test	_
1865986 PID:411000a TID:412000a *** Test ID: 8012	
1865986 PID:411000a TID:412000a *** Library Path: touchtest	
1865986 PID:411000a TID:412000a *** Command Line:	
1865986 PID:411000a TID:412000a *** Kernel Mode: Yes	
1865986 PID:411000a TID:412000a *** Random Seed: 12562	
1865986 PID:411000a TID:412000a *** Thread Count: 0	
1865986 PID:411000a TID:412000a *** vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv	m
1865986 PID:400002 TID:457000a BEGIN TEST: "Pen Up / Down Test", Thre	a
1865986 PID:400002 TID:28d012e Calling TouchPanelSetMode(TPSM_PRI	ð:
1881090 PID:400002 TID:28d012e Calling TouchPanelSetMode(TPSM_PRI	ð:
1891135 PID:400002 TID:457000a END TEST: "Pen Up / Down Test", FAILED	
1891135 PID:411000a TID:412000a *** ^^^^^^^^^^^^^^^	<u>^.</u>
1891135 PID:411000a TID:412000a *** TEST COMPLETED	-
1891135 PID:411000a TID:412000a ***	~
🗐 Output 📸 Error List	

> View results

- 15. Once the tests are complete switch back to the Windows Embedded CE Test Kit window.
- 16. Select **Tests** | **View Results** | **WindowsCE (ARMV4I)** | **View All Results** from the CETK menu. The **CETKParser** window will appear.

B Windows	Embedd	led CE	Test Kit - CETKDar	sor					
File Edit View	v Help	led et	Test Kit - GLTN- di	361					
	Ba 🖪	A	9						
Filename		esult	o Device	TestDll	Encrypted	Type	Size	Date	Time
Battery API Te	st.log P	assed	WindowsCE (ARMV4I)	hatanitest.dl	No	Tux	8784	2/9/2009	12:6:35
Touch_Panel_Te	est.log P	assed	WindowsCE_(ARMV4I)	touchtest	No	Tux	18853	2/9/2009	12:8:45
Tort Caro Day		ct Norso							
Summary Pas	suit Te: ssed (- Show t	the Summary of the Ove	rall Test Suite	3				
1001 Pas	ssed Gel	tSystem	PowerStatusEx2		/				
1002 Pas	ssed Bat	tteryDrv	rGetLevels						
1003 Pas 1004 Pas	ssed Bat ssed Bat	tteryDry tteryGet	rsupportsChangewotiric LifeTimeInfo	ation					
Call	lina B	atter	uGetLifeTimeIn	fo(x. x. N	ULL)				^
END TES	ST: "B	atter	yGetLifeTimeIn	fo", PASSE	D, Ťime=	0.004	۱		
*** ^^^	ст сом	ооооо DI Г Т Г	Б		~~~~~~	~~~~			
***	51 601	FLEIE	U.						
*** Tes	st Nam	e:	BatteryGetL	ifeTimeInf	0				
*** Teg	st ID:		1004						
	brary mmand	Path: Lino:	\batapitest	.dll					
*** Ker	rnel M	ode:	No						
*** Res	sult:		Passed						
*** Rar	ndom S	eed:	8462						
*** Inr *** Fxr	read C Poutio	ount: n Tim	 						
***				*****	~~~~~~	~~~~			*****
1									M
Boodu									
кеаду									

- 17. Click on the **Battery_API_Test.log** in the top pane. The middle pane will show each of the subtests along with their status.
- 18. Click on the last subtest, **BatteryGetLifeTimeInfo** in the middle pane. The bottom pane will show the detailed test log for that particular subtest.
- 19. Close the **CETKParser** window.

Exercise 2 Modify the command line for CETK tests

In this exercise you will modify the command line of individual tests. Each test typically has a number of configurable parameters that can be changed from within the CETK window. These parameters can be used to target the testing to a particular problem area, speeding up the overall development cycle. The CETK test harness itself also has configurable parameters.

- > Configure Graphics Device Interface Test
 - 1. Right click on the **Graphics Device Interface Test** in the **Display** node and select **Test Information.** The documentation for this specific test will load in the Microsoft Document Explorer. The documentation indicates what parameters are available for this specific test.

Note There are typically several pages in the documentation for each test. You may have to change to a different page to see the command line parameters.

- 2. Right click on the **Graphics Device Interface Test** in the **Display** node and select **Edit Command Line...**
- 3. Add the following command line parameters to the end of the existing command line:

Edit Command Line	X
Graphics Device Interface Test	
<u>C</u> ommand line:	
x -o -d gdiapi -x 301 -c"/Width 240 /Height 240"	
Target device: WindowsCE (ARMV4I)	
Apply command line:	
⊙ <u>I</u> emporarily, to this target device.	
Permanently, to all devices.	
OK Cancel <u>H</u> elp	

-x 301 -c"/Width 240 /Height 240"

Note The –x parameter tells the test harness to run only subtest number 301.

The -c parameter tells the test harness to pass everything in quotes to the actual test dll, in this case gdiapi. The parameters within quotes are interpreted by the individual test and are not consistent among tests.

- 4. Click **OK** to temporarily change the command line.
- 5. Right click on the **Graphics Device Interface Test** and select **Quick Start**. This individual test will run, and no others. This is a convenient way to run targeted tests.

Note For a lab that covers writing custom CETK tests, go to <u>www.microsoft.com</u> and search for **Advanced Automated Test Development with TUX**.