i.MX31 SOM-LV Windows Embedded CE 6.0
Lab 2: Application Development
BSP Documentation

Logic // Products
Published: June 2009

Revision History

REV	EDITOR	DESCRIPTION	APPROVAL	DATE
A	MH	Initial release	JCA	06/09/09

Table of Contents

1 Introduction

1.1 Learning Objectives

- Export and install an SDK
- Develop and deploy a managed application
- Interact with native code in a DLL using p/invoke

1.2 Prerequisites

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

Knowledge of C and C++ languages

1.3 Lab Setup

To complete this lab, you must have:

- A development workstation running Windows XP or Vista
- Visual Studio 2005 (Version 8)
- Service Pack 1 for Visual Studio 2005
- Service Pack 1 for Visual Studio 2005 on Vista (if you are running Vista)
- Windows CE 6.0 R2 Platform Builder plug-in with Service Pack 1
- A Zoom i.MX31 LITEKIT
- The i.MX31 SOM-LV Windows CE 6.0 BSP

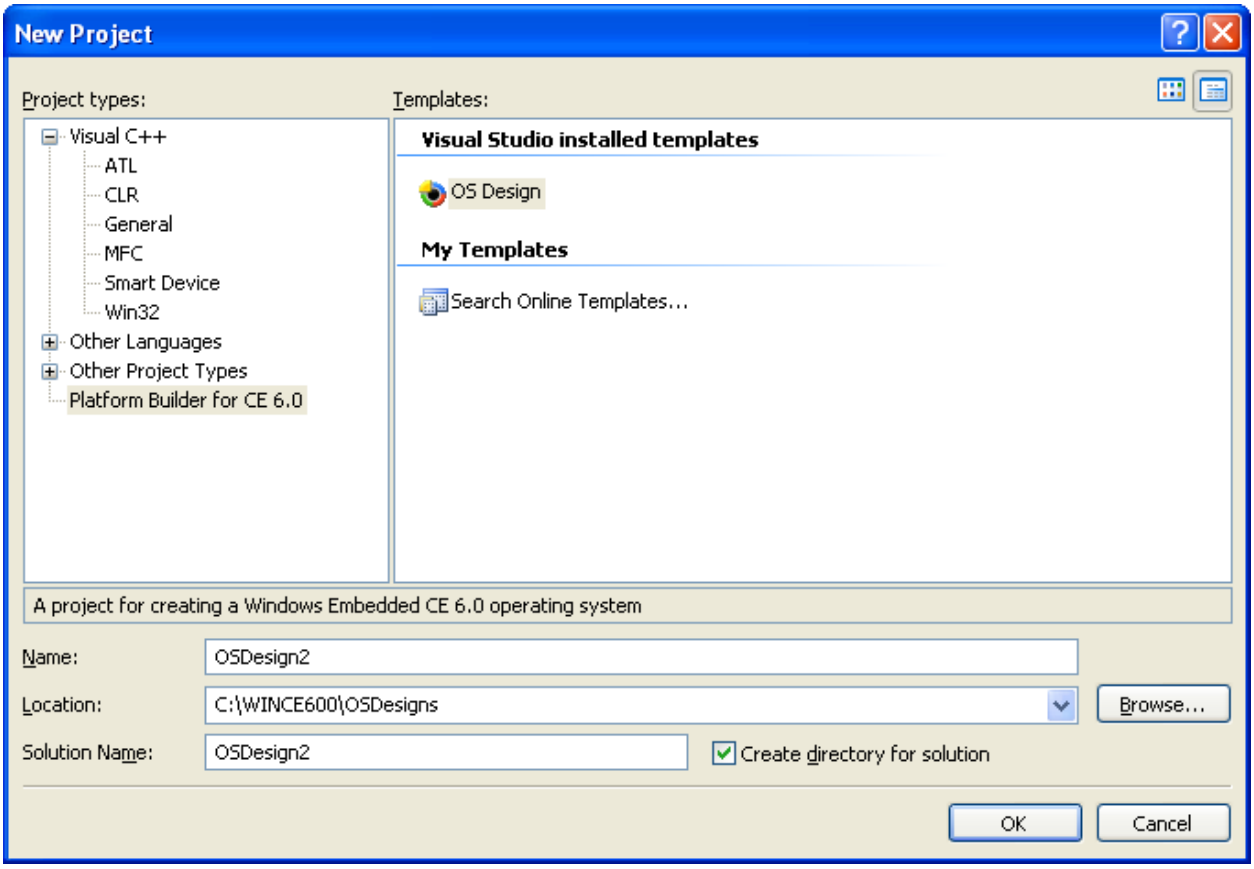
2 Creating the Platform Image

2.1 Create a new OS Design

This OS Design is very similar to that of Lab1, except this time we are going to include the .Net Compact Framework 2.0.

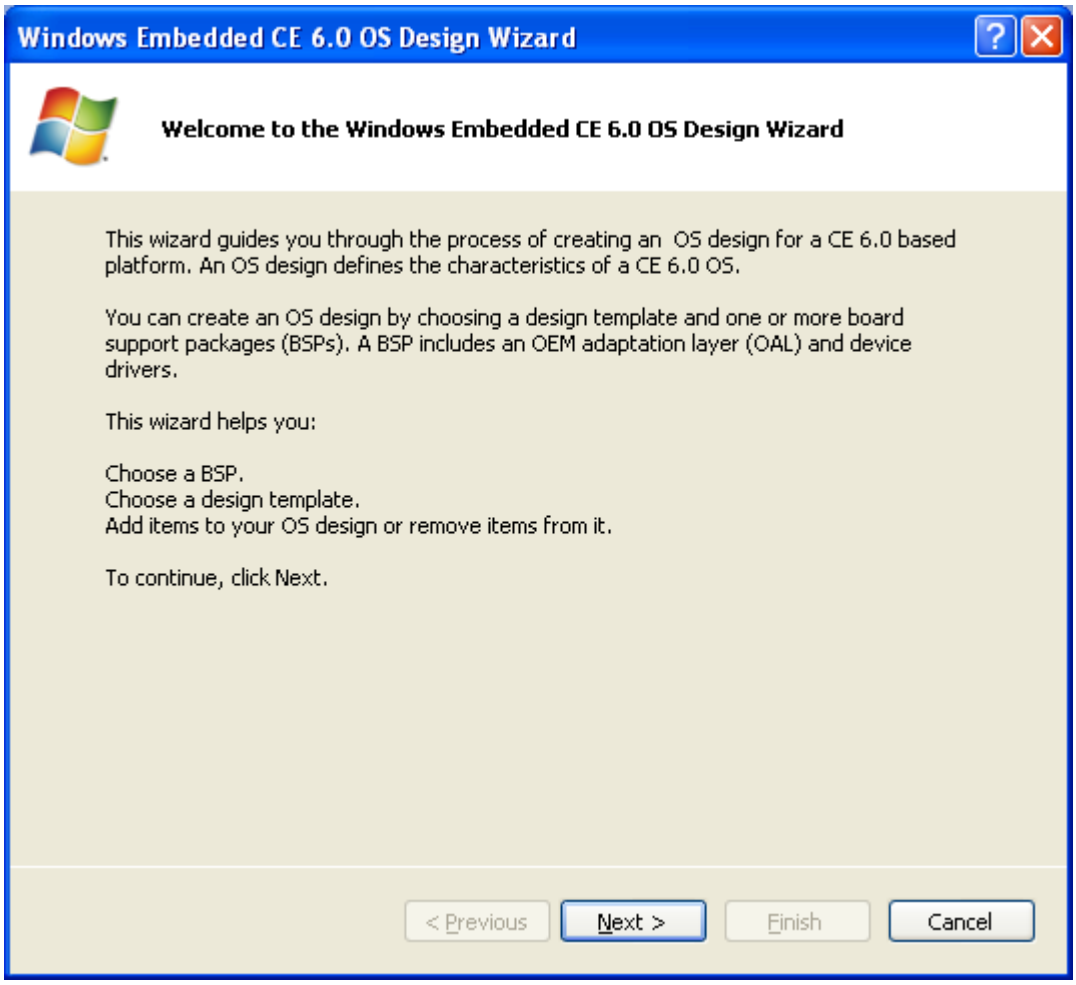
Launch the Visual Studio 2005 New Project dialog.

- Select **File** | **New…** | **Project…**



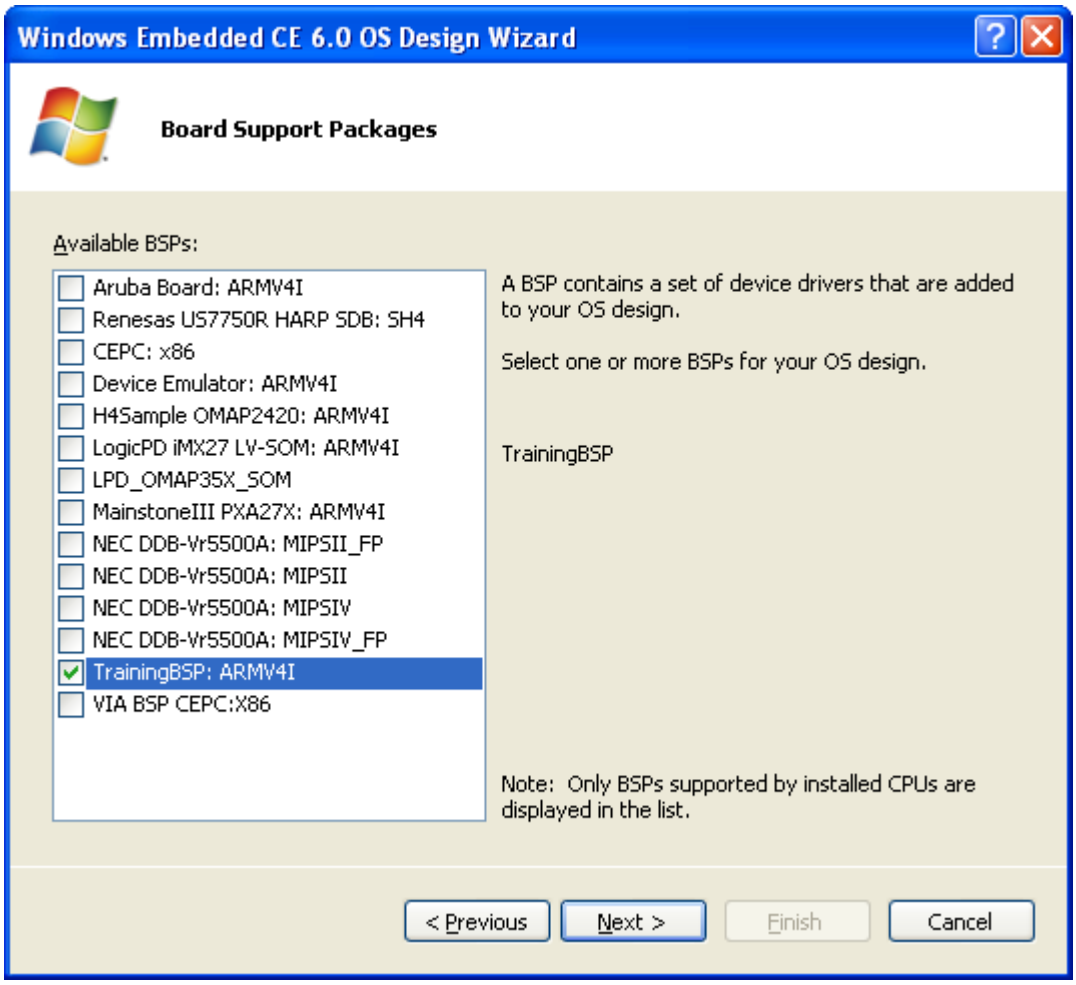
- Select the **Platform Builder for CE 6.0** project type.
- Choose the **OS Design** template
- Enter a name for your project, or go with the default name
- Click **OK**

The initial Dialog that is displayed below outlines the process of creating an OS Design. There are a number of steps involved and you will step through the wizard and make selections as appropriate.



We will build an operating system for the i.MX31 SOM-LV device.

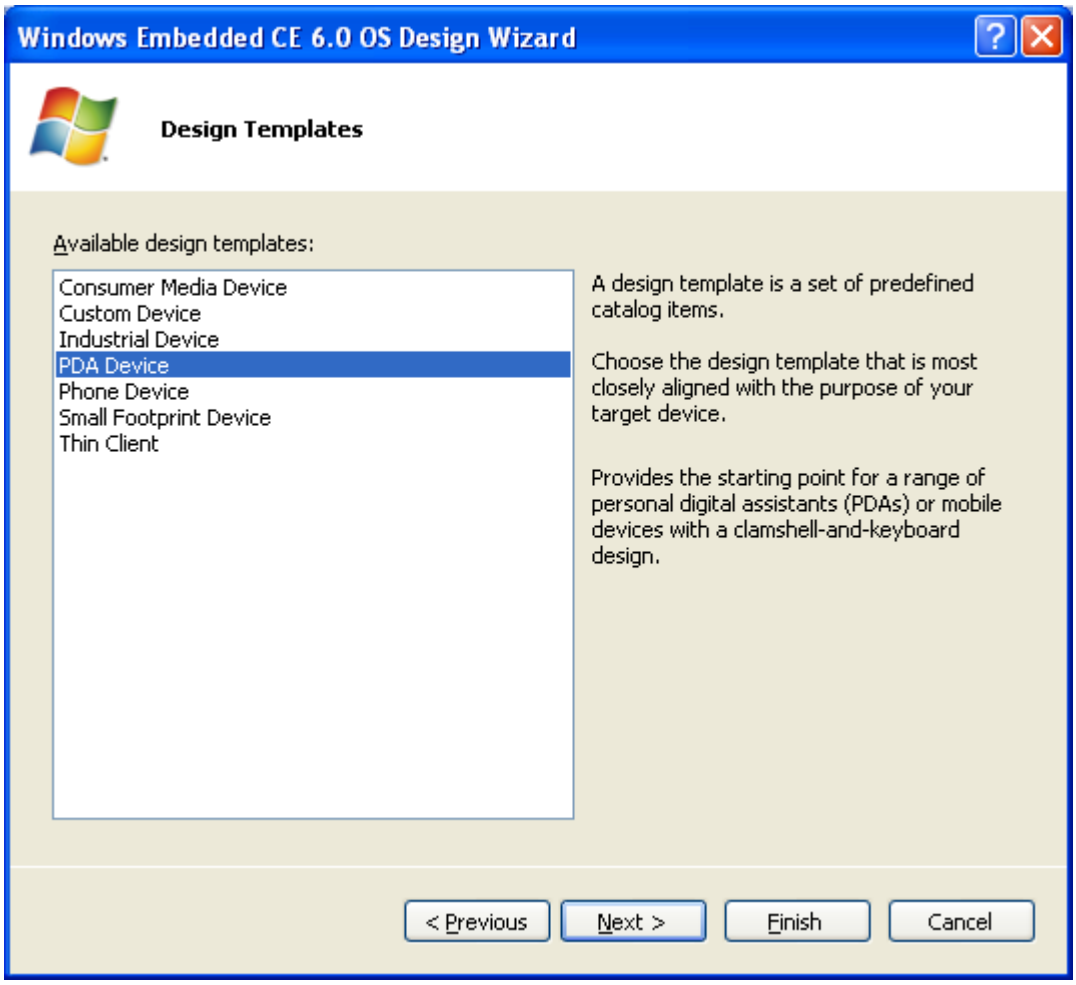
You could select more than one BSP (Board Support Package). For example, it may be useful to select the Device Emulator and a hardware reference platform; the emulation environment can be used by your internal/external application developers to build applications for your device while hardware is still being developed.



- Select **TrainingBSP: ARMV4I**
- Click **Next**

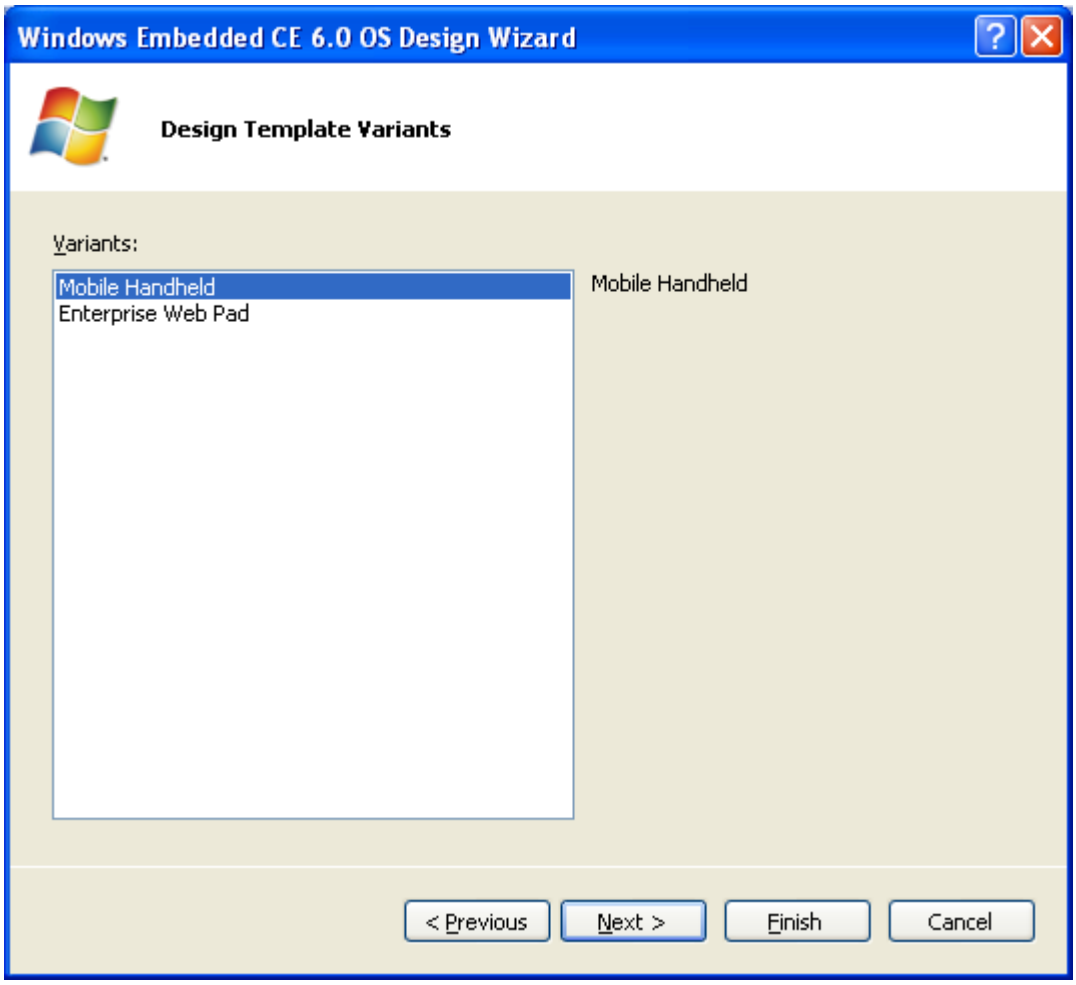
You are now given a number of Design Templates to choose from as the starting point for this project. If none of the options match your needs you can simply select **'Custom Device'** and build the image based on the components you select from the catalog.

For the purposes of this tutorial you will be using the **PDA Device** template.



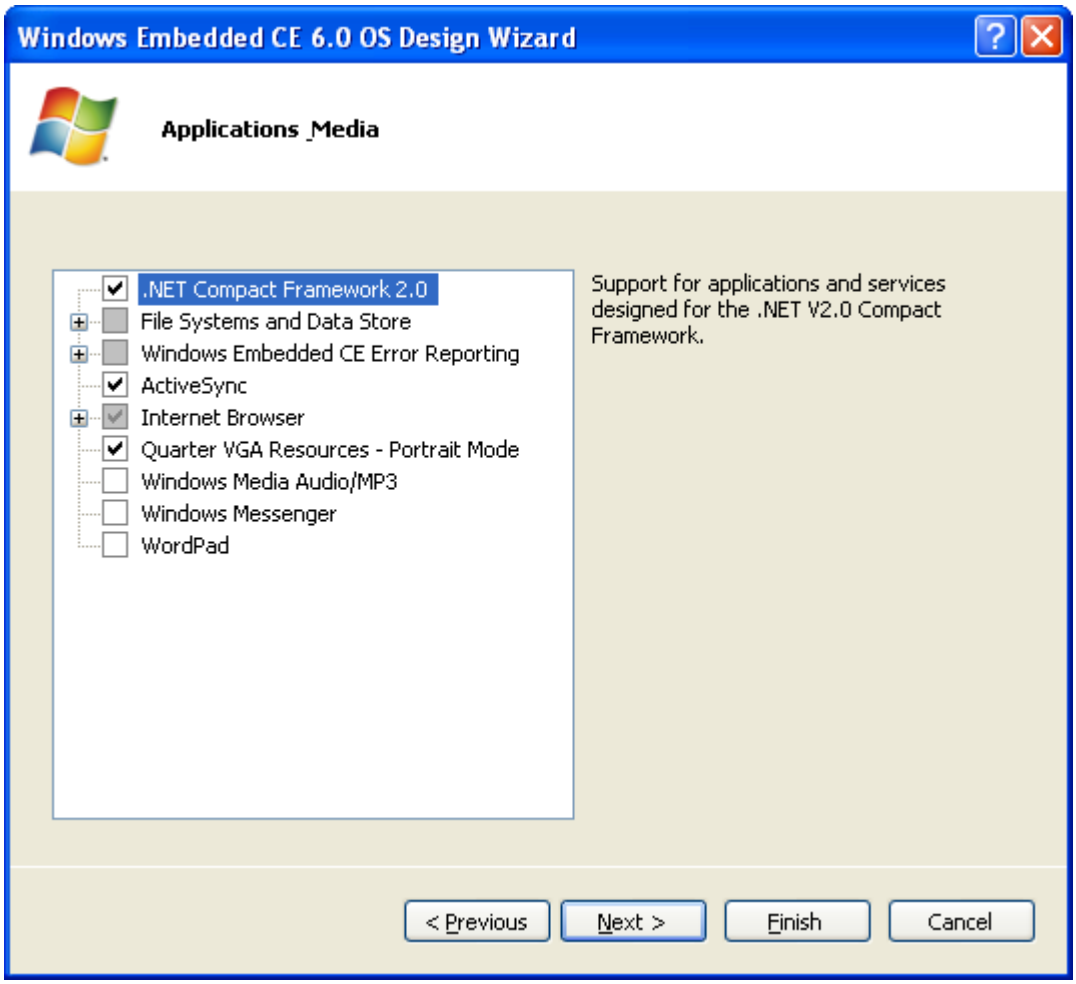
- Select **PDA Device** from the list of Design Templates
- Click **Next**

The Wizard now gives you the option of choosing a variant of the Design Template. For this tutorial you will be using the Mobile Handheld variant.



- Select **Mobile Handheld** from the list of Design Template Variants
- Click **Next**

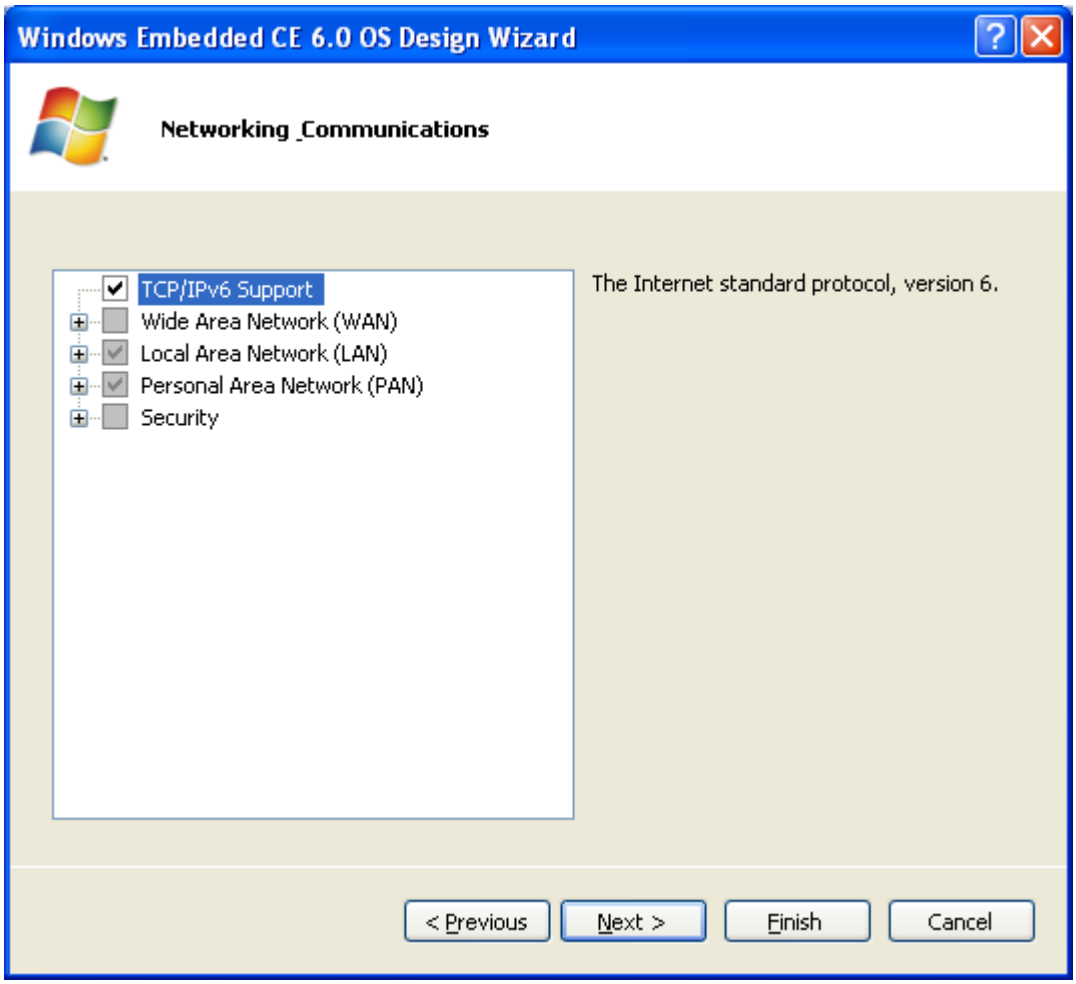
There are a number of options which can be selected for each of the sample platforms, the OS Design Wizard will only show options that are relevant to the platform you are building. For example, it would not make sense to include Internet Explorer or WordPad in a headless device such as a Gateway. The Mobile Handheld device can include applications such as Internet Explorer, WordPad, and Windows Messenger.



- Make sure that .NET Compact Framework 2.0 is selected
- Click **Next**

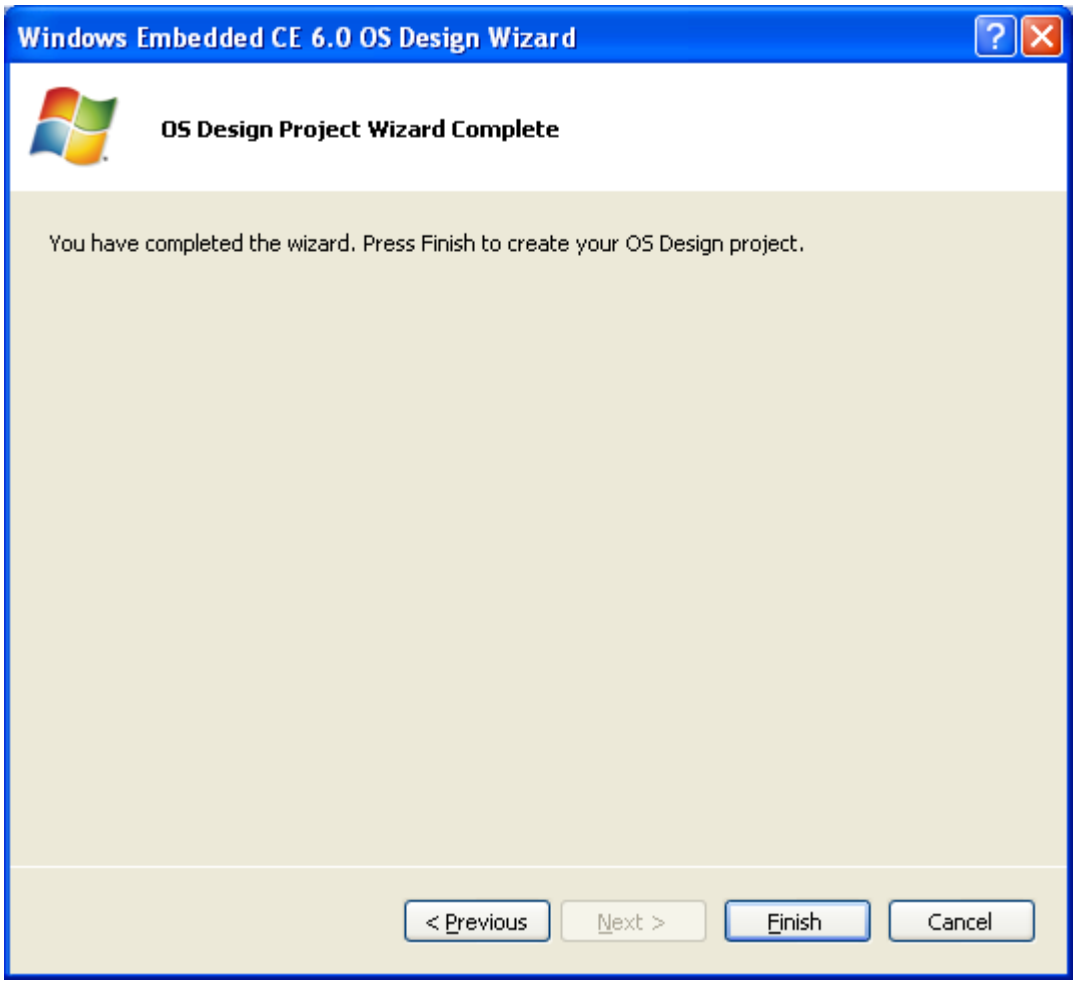
Next comes networking and communications, you can see that Windows CE provides support for Personal, local, and wide area networking through technologies such as Bluetooth, IrDA, Wired and Wireless networking, and VPN.

In our case the default options are fine.



- Click **Next**

We're done! – The wizard is complete. You've configured the OS Design and can now further customize it by adding and/or removing components from the Catalog.



- Click **Finish**

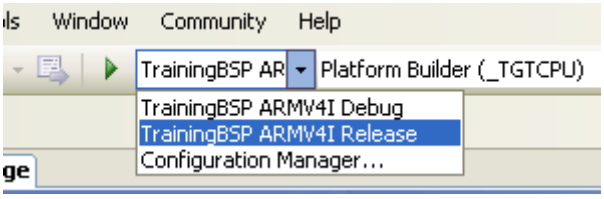
The Wizard will now prompt you with any security warnings or other notifications related to the components you selected. Acknowledge these notifications.

- Click **Acknowledge**

At this point you have an OS Design containing all of the OS components which have been selected from the Wizard.

Now we will set the project's build configuration to **Release**.

- Verify that the active build configuration is set to **TrainingBSP ARMV4I Release** using the pulldown menu in the main toolbar or by using **Build** | **Configuration Manager**



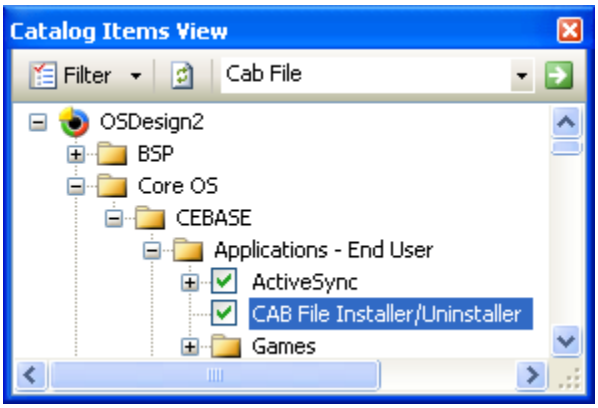
2.2 Customize the Design

You will now make changes to the items included in your OS Design using the Catalog View.

First, ensure that the Catalog View is visible.

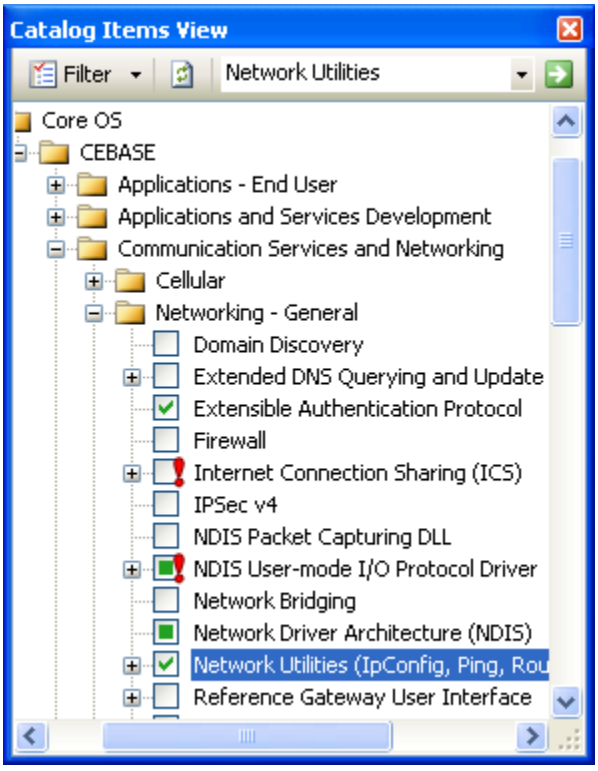
- Select the **View** menu
- Choose **Other Windows** | **Catalog Items View**

You want to ensure that cab file support is included in the run-time image that you build. Search for "Cab File Installer/Uninstaller" to locate the appropriate catalog item.



- Type **Cab File** in the search box
- Press the **Enter** key to search
- **Check** the box next to the **Cab File Installer/Uninstaller** item to include it in your OS Design

You want to ensure that network utilities are included in the run-time image that you build. Search for “Network Utilities” to locate the appropriate catalog item.

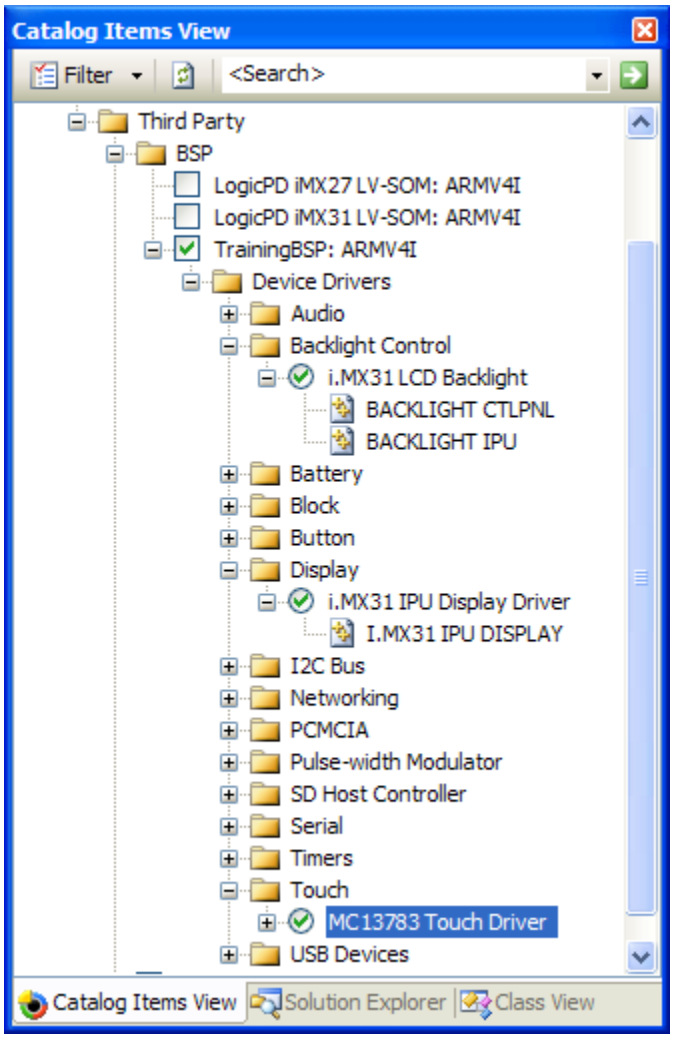


- Type **Network Utilities** in the search box
- Press the **Enter** key to search
- **Check** the box next to the **Network Utilities (IpConfig, Ping, Route)** item to include it in your OS Design

We need to select a few other components for our i.MX31 SOM-LV device.

- In the search box, type **i.MX31 LCD Backlight**
- Select the **i.MX31 LCD Backlight** component

- Note that there are other device drivers that are a part of this BSP.
- Under **Display**, select **i.MX31 IPU Display Driver**
- Under **Touch**, select **MC13783 Touch Driver**
- Your selections should match the following image:



The Catalog is an important tool for configuring your OS Design. If you choose to explore this view further, you will find advanced features for the analysis of dependencies between the various items.

2.3 Add the CoreCon Helper Files

Windows Embedded CE 6.0 R2's Core Connectivity suite enables application developers to very quickly deploy their applications to a remote device and test them through Visual Studio 2005 SP1. To deploy to a remote device through Visual Studio 2005, you must have certain Core Connectivity helper files on the remote device, which are installed on a development PC by Platform Builder in $(CommonProgramFiles)\Microsoft Shared\CoreCon\1.0\Target\wce400\ .

A subproject has already been created for this lab which will automatically incorporate the required files into the run-time image. To use this subproject:

- In **Windows Explorer**, navigate to **C:\Labs\Lab2\LabFiles** and copy the **CoreCon_ARMV4I** folder into your OS Design folder, in **C:\WINCE600\OSDesigns\OSDesign2\OSDesign2**

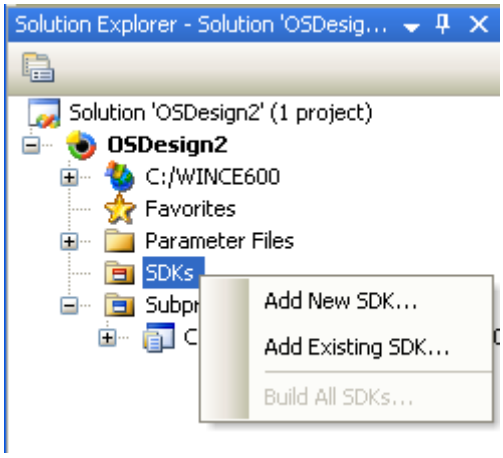
- Return to **Visual Studio 2005** and open the **Solution Explorer** view by selecting **View** | **Solution Explorer**
- Right click **Subprojects** and select **Add Existing Subproject**
- Navigate to the **CoreCon_ARMV4I** subfolder you just copied into the OS Design and select the pbxml file to add CoreCon_ARMV4I to the OS Design.
- Build a new run-time image by using **Build** | **Build Solution**

Note: Adding Files to a Run-Time Image. The subproject we added just includes files that were already installed on our development workstation into the image. If time permits later on, look at this subproject's **postlink.bat** and **project.bib** files to see a straightforward way to include files into a run-time image.

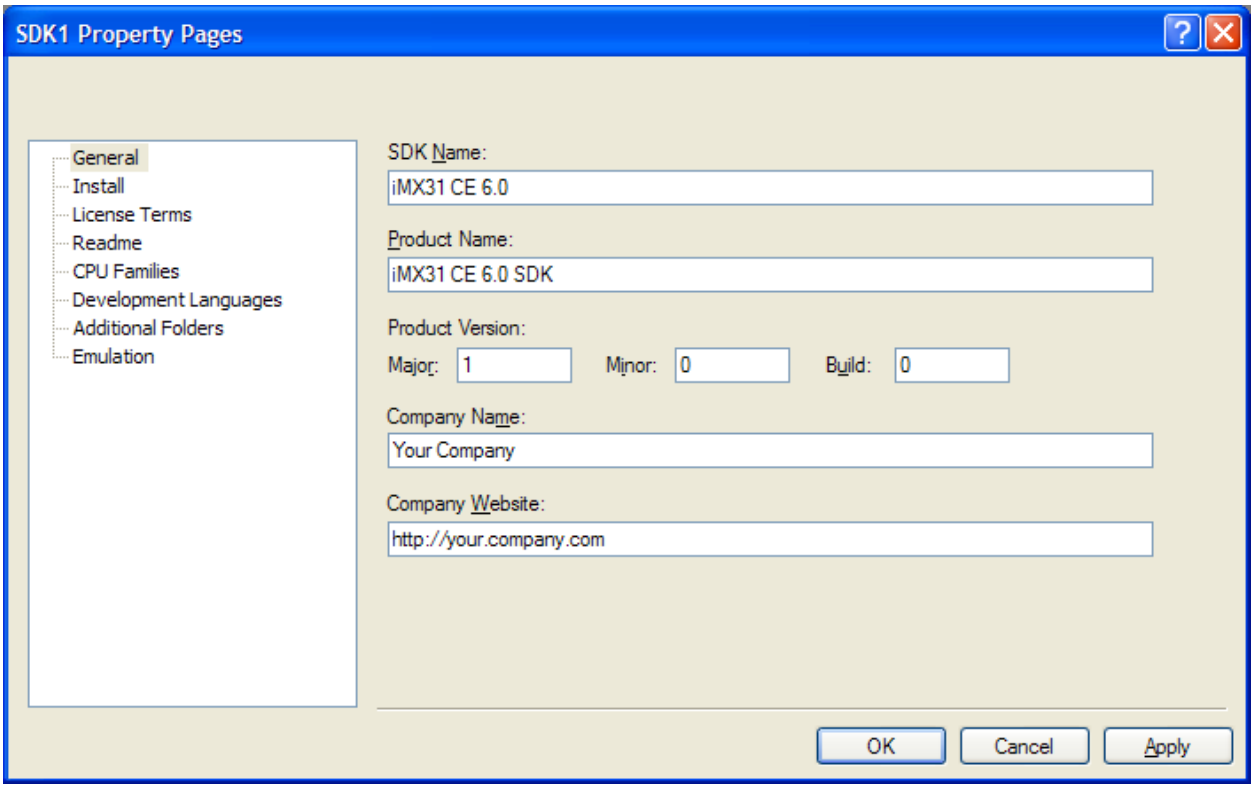
2.4 Create and Install an SDK

To develop with native code on a target device, you must have an SDK for the device. In Windows Embedded CE 6.0 R2, developers must create an SDK for their OS Design in order to develop applications for it.

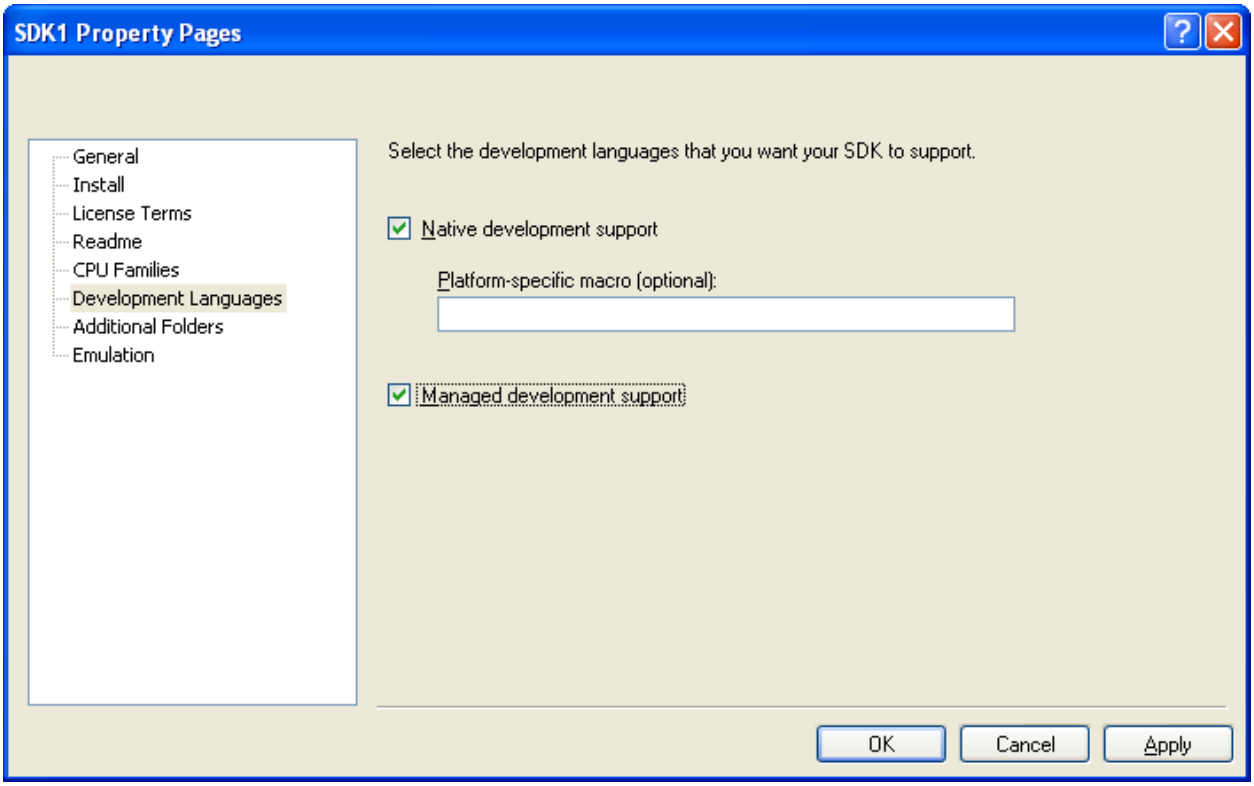
- In the **Solution Explorer** view, right click **SDKs** and select **Add New SDK**



- Fill in all of the fields in the **General** tab. Choose something descriptive such as **iMX31 CE 6.0** for the Name field. (Remember this name, as we'll use it later)

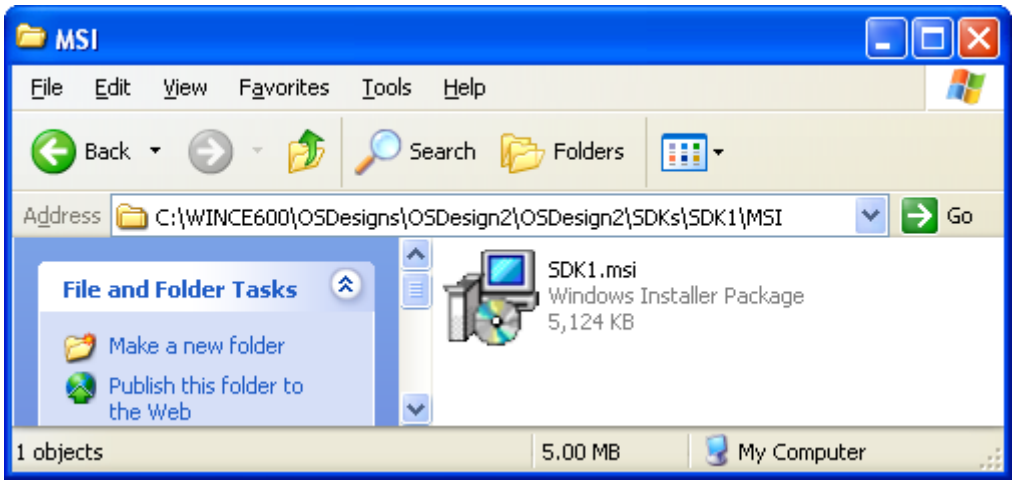


- In the **Development Languages** tab check the boxes next to both **Native** and **Managed** development.



- Click **OK** to finish creating the SDK.

- Right click the new SDK in the **Solution Explorer** and select **Build**.
- Navigate to the newly created MSI and install it on your development workstation. Just click **OK/Next** through the steps, and select **Complete** when given a choice of which components to install

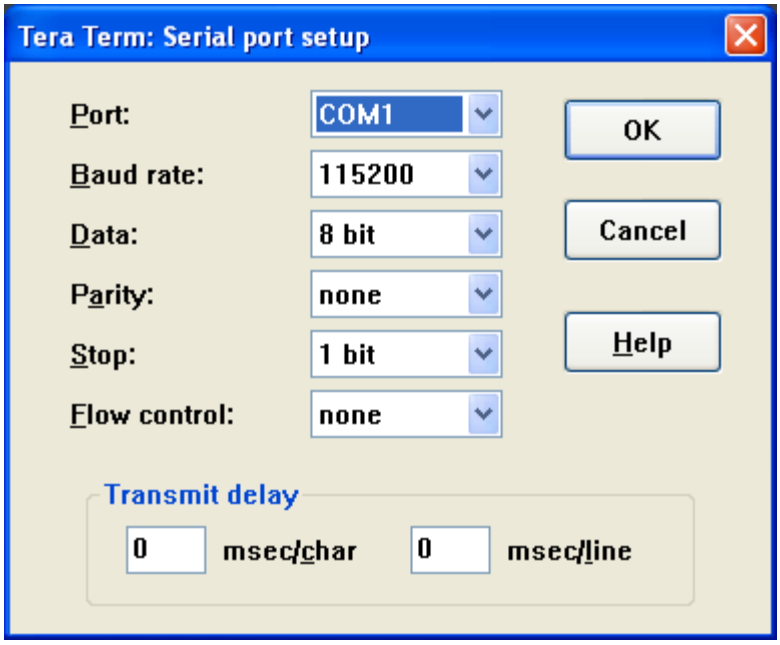


3 Download the Platform Image

3.1 Configure Download/Debug Transport

You are now ready to download and debug the Windows CE operating system image – before you download let's check to make sure the debug and kernel transports are configured and that we can view debug information over the serial terminal.

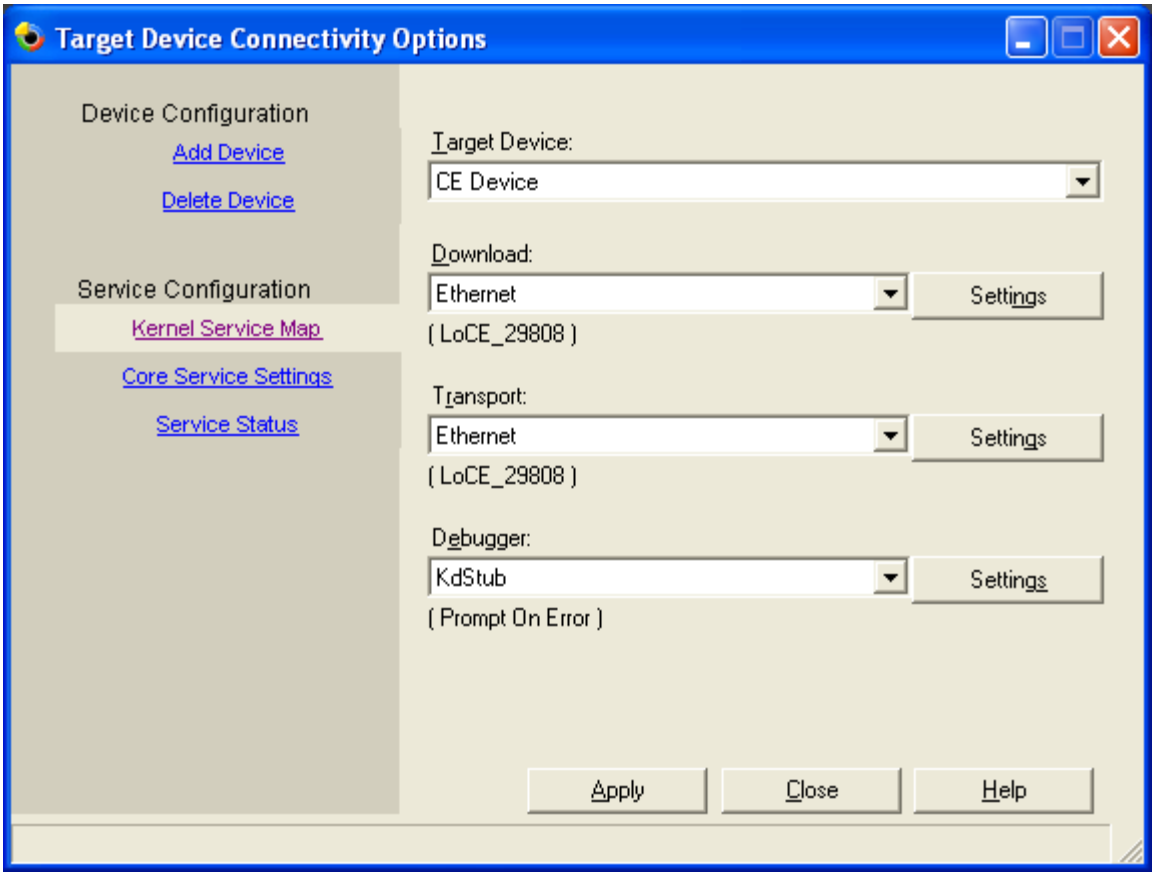
- Open **TeraTerm** by double clicking the icon on the desktop.
- Verify the serial port settings by clicking the **Setup** | **Serial Port** menu item (Your physical Port may differ):



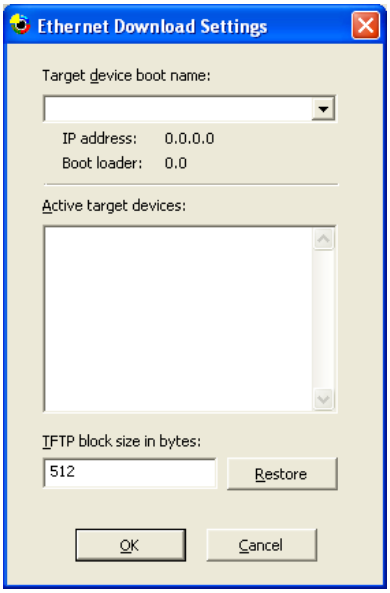
Ok now that you have configured your serial terminal we can continue to setup the device and Platform Builder. Switch back to your Visual Studio 2005 instance.

- Select **Target** | **Connectivity Options**

The following dialog will be displayed:



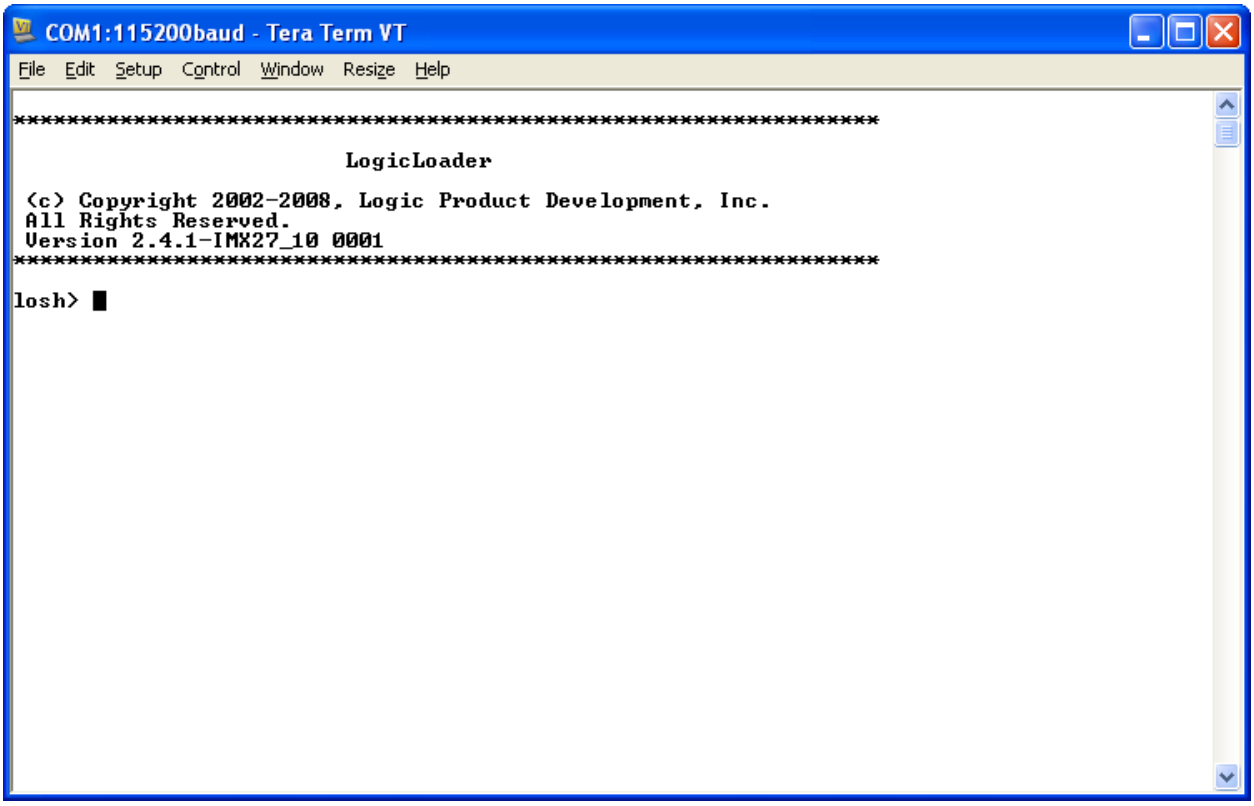
- In the **Download** box select **Ethernet**
- In the **Transport** box select **Ethernet**
- In the **Debugger** box select **KdStub**
- Click on the **Settings** box next to the **Download**: dropdown box. A box like this will appear:



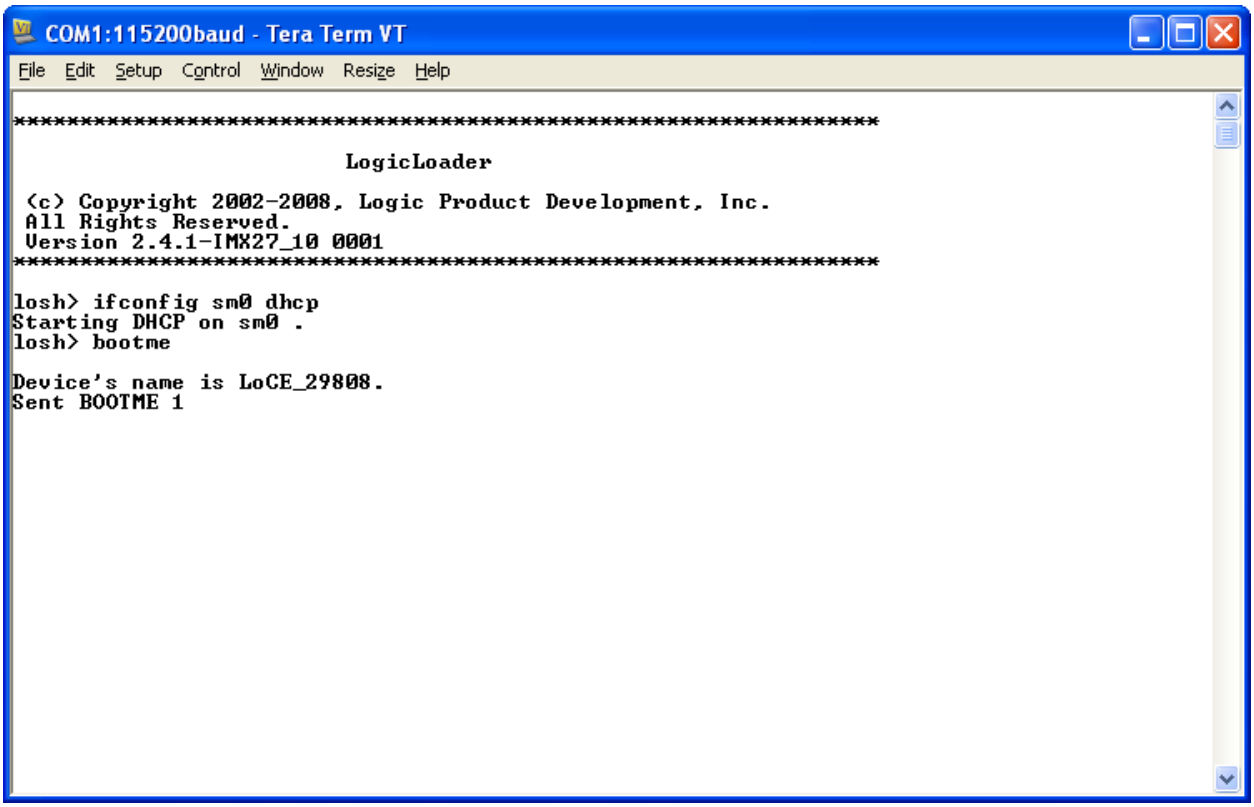
- Leave this box open and move on to the next step

Now we will switch to the **Tera Term** window to configure our device.

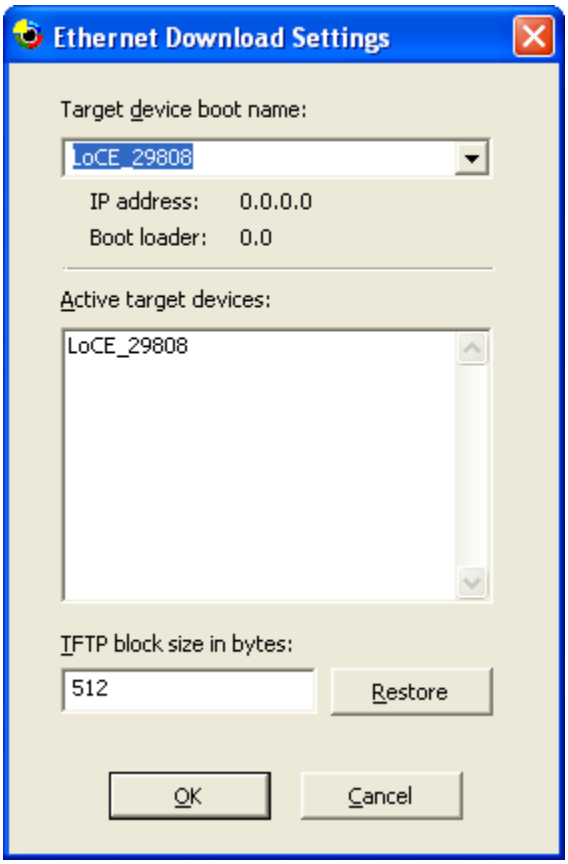
- Power on the device.
- You should see a **losh** prompt display.



- Configure the network interface using either DHCP or a statically assigned address:
  - For DHCP, type: **ifconfig sm0 dhcp**
  - For a static ip address, use the following command: **ifconfig <interface> <ip> <netmask> <gw>**
    - Example: **ifconfig sm0 192.168.1.1 255.255.255.0 192.168.1.0**
- Type the command **bootme** and press the Enter key. This will allow visual studio to connect to this device. Note the name of the device that is displayed:



- Switch back to Visual Studio and the box you left open should now be populated with a device. If there are multiple devices in the list, click on them and view the IP address. Make sure it matches up with the one that you wrote down.
- **Note**: if the device does not appear in Platform Builder's "Ethernet Download Settings" window, please check your firewall settings and anti-virus software. The "bootme" protocol uses raw UDP packets for communication between the device and the tools.



- Click **OK** in the small box, and then click **Apply**. Once you have done this, click the **Close** button.

3.2 Download the Operating System

- Select **Target** | **Attach Device**
- The device should still be executing the **bootme** command, but if it is not, type **bootme** at the **losh** prompt
- When the image is finished transferring (a **losh** prompt will display), type the following command to boot the image (all on one line):
  exec rtc:rtc_imx31_pmic:dbg_leds:1:disp_num:3:kitl:true:share_eth:1:

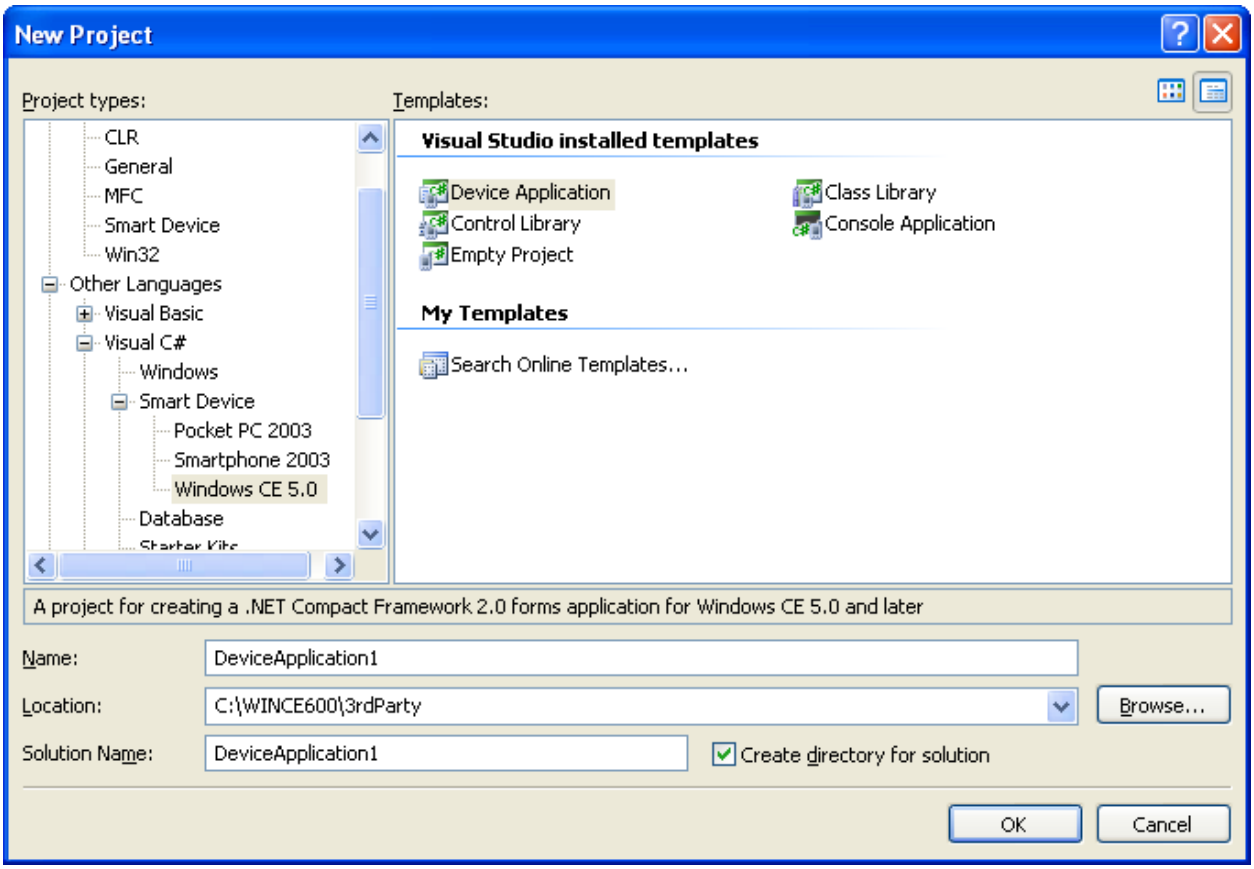
4 Developing a Managed Application

Now we'll develop a simple managed application for the device. Note that this lab is slightly different from the other labs, so make sure you followed the customization directions earlier in this lab. Otherwise you won't be able to deploy this application to the device.

4.1 Create a Project

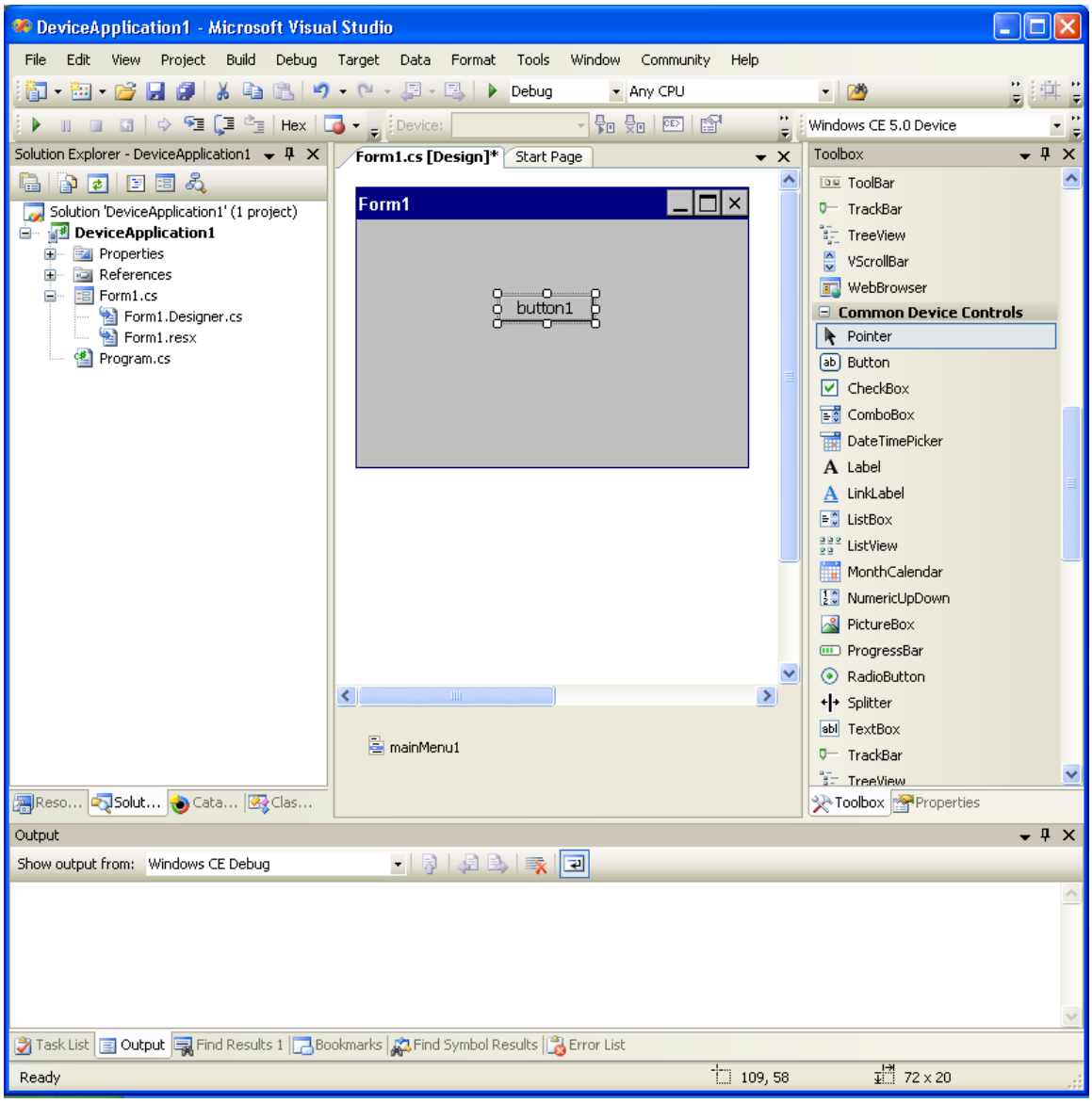
- Open a new instance of Visual Studio (keeping the previous instance with your OS design open!)
- Create a new managed application by selecting **File** | **New** | **Project**
- In the New Project screen select on the left side select **Other Languages** | **Visual C#** | **Smart Device** | **Windows CE 5.0** (Windows CE 5.0 applications created this way are compatible with CE 6.0)

- Select **Device Application** as the template
- Choose a name such as **HelloWorldApp** or just use the default name
- Create the application wherever you wish; it does not have to be in any particular folder. Third party applications for Windows CE devices are often stored in the WINCE600\3rdParty folder, as it is relatively straightforward to create catalog items for components in that directory.



4.2 Create the UI and Add Code

- In the newly created project, open the Toolbox by selecting **View** | **Toolbox**
- Add a button by dragging a Button from the toolbox onto the form view



- Double click the button to create an onclick event handler for the button and jump to that event handler's code.
- Add code to open a message box when this button is clicked.

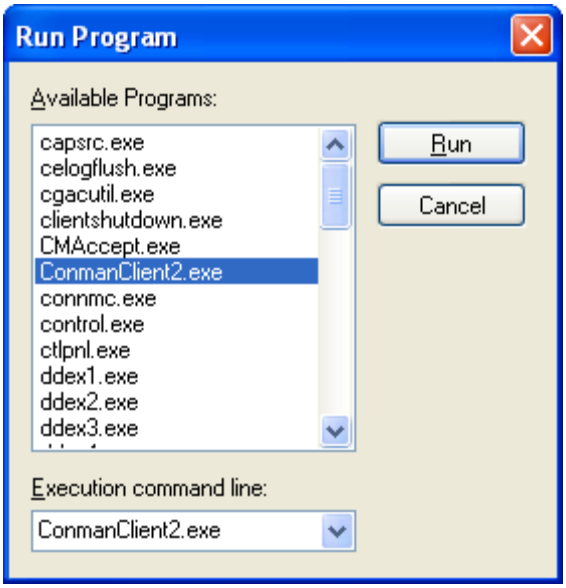
```
MessageBox.Show("Hello, World");
```

- Build the project by selecting the **Build** menu and **Build Solution** at the top. It should build without any errors.

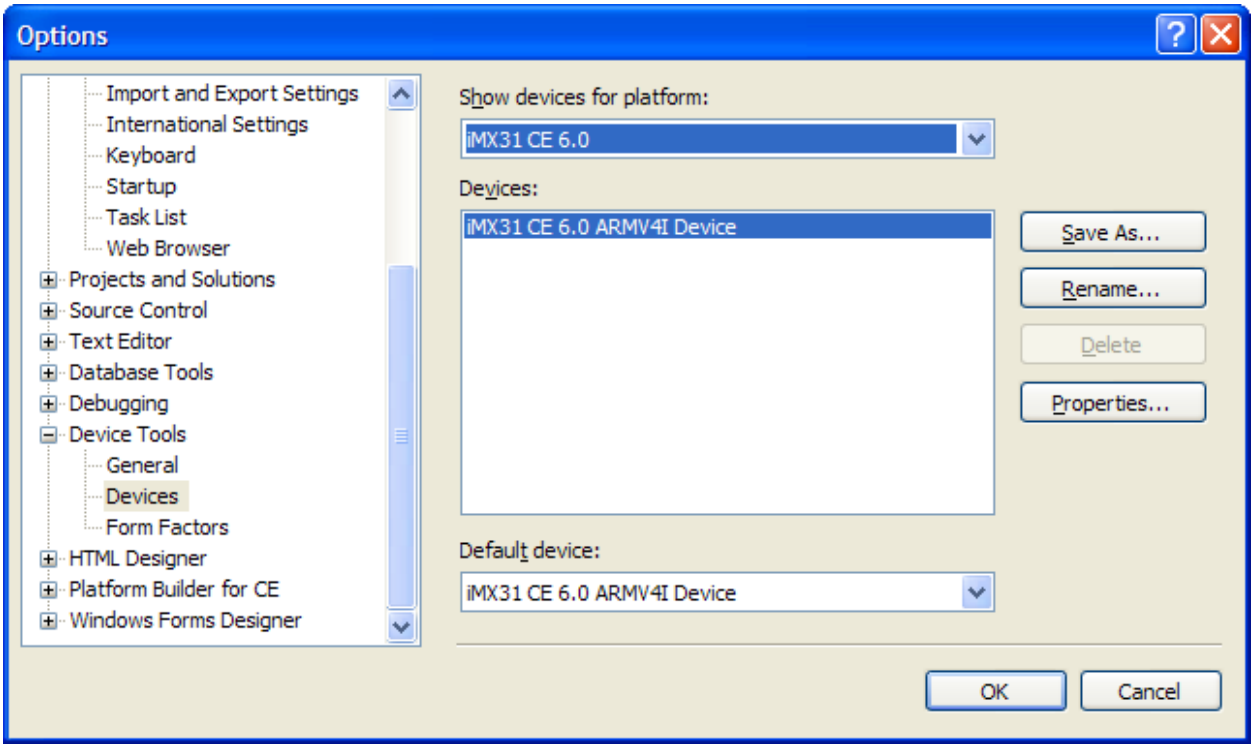
4.3 Deploying to the Remote Device

Now we'll deploy the code to the i.MX31 SOM-LV device using the Core Connectivity helper files we added earlier before building the runtime image.

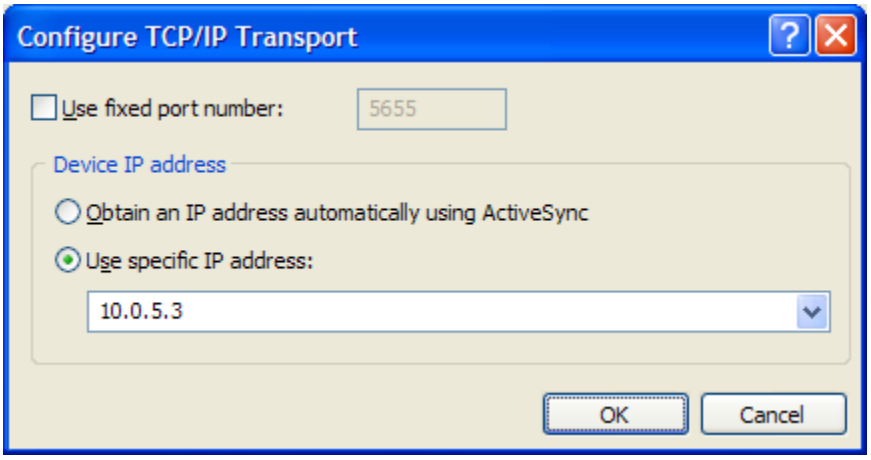
- Return to the first instance of Visual Studio which is running your OS Design.
- Run the Core Connectivity helper application on the remote device by using **Target** | **Run Programs** and selecting **ConManClient2.exe**



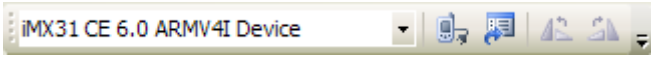
- Next open the Run Program screen again and run **CMAccept.exe**. This will instruct ConManClient2 to accept an incoming TCP/IP connection
- Retrieve your device's IP address. If you don't remember it, you can enter the **Target Control** screen by pressing **ALT+1** in the OS Design instance of Visual Studio, then type **s ipconfig** at this screen to run ipconfig, which will print out your IP address to the debug output in Visual Studio.
- Return to the instance of Visual Studio with your managed application. Open the Device Options screen by selecting **Tools** | **Options** and then on the left side expand **Device Tools** and select **Devices**



- Select **iMX31 CE 6.0** under “Show devices for platform:” and under “Devices:” select **iMX31 CE 6.0 ARMV4I Device** then click Properties...
- Select **Configure** next to Transport and specify your device’s IP address

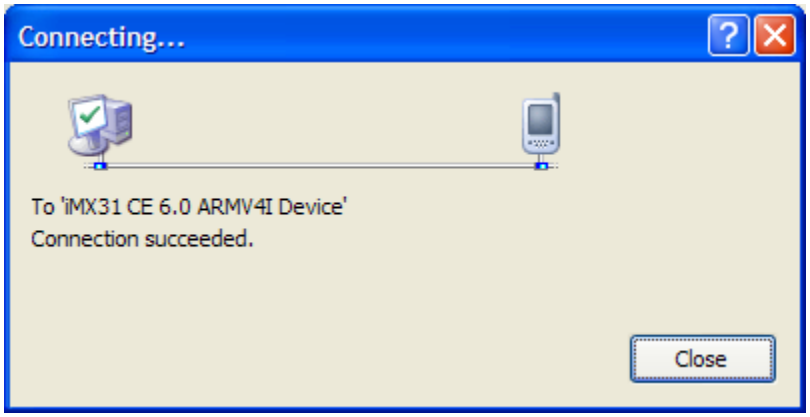


- Click **OK** at all the open windows to save the selected settings.
- Ensure that the **Device** toolbar is visible (see below for an image of it). If it is not visible, enable it by selecting **View** | **Toolbars** | **Device** and make sure the checkbox next to Device is checked.

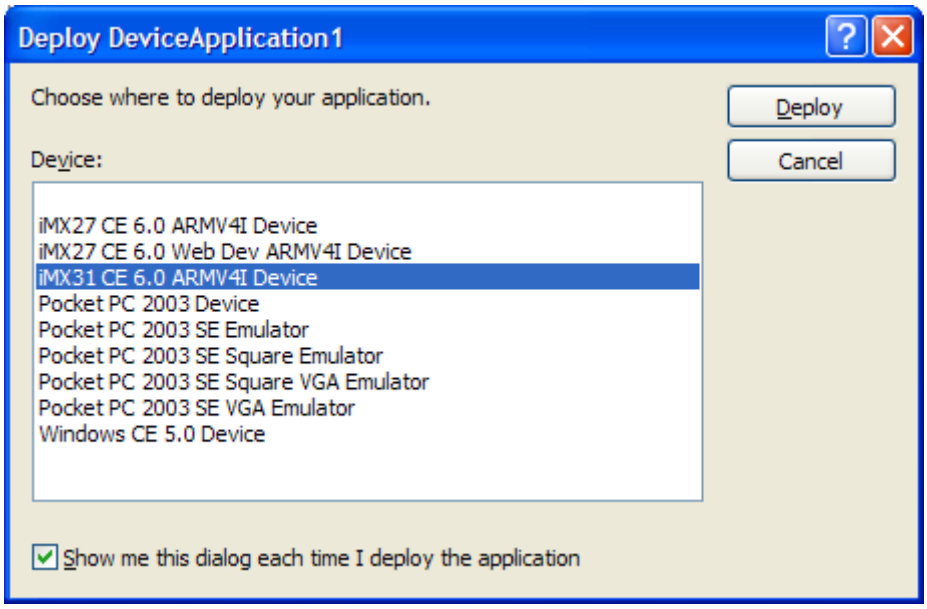


- Select **iMX31 CE 6.0 ARMV4I Device** from the dropdown box and click the button next to the pulldown box to connect to your i.MX31 SOM-LV.
- Assuming **ConManClient2** and **CMAccept** are still running on the VIA device, the connection should succeed. If it fails, return to the first OS Design, enter the **Target Control** screen by

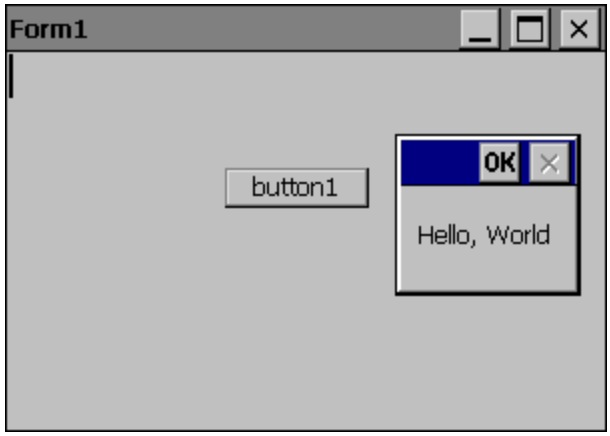
pressing **ALT+1** and then type **gi proc** to list all the running processes and verify that those applications are running. If not, run **ConManClient2** first with either the **Target** | **Run Programs** window or by typing at the **Target Control** screen **s ConManClient2**, then run **CMAccept**



- Now that you've successfully connected, you can deploy to the device by selecting **Debug** | **Start Debugging** or clicking the right-facing green arrow on the debugging toolbar.
- You may get a dialog asking where to deploy your application. Make sure that **iMX31 CE 6.0 ARMV4I Device** is selected and select **Deploy**



- The application should run on your target device. Click the button you created to show a messagebox. From here, it's straightforward to do whatever .NET Compact Framework development you need to do.



- When you're done, just close the application and return to the Visual Studio instance with your managed code.

4.4 Using Breakpoints and Run-time Exceptions

Note: This section does not always work properly. It might not be possible to set the breakpoint and have it be hit while debugging.

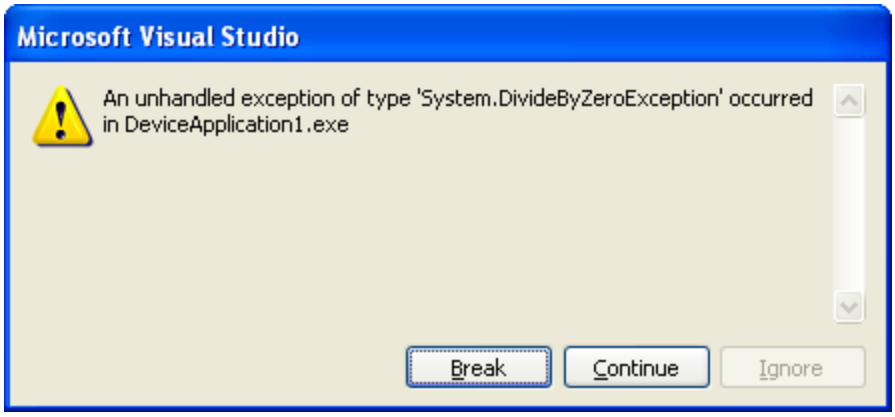
Now we'll see some of the features available when debugging on a remote device with Visual Studio through the Core Connectivity suite.

- Return to your application code and add another MessageBox.Show statement after the first
- Add a breakpoint before the second statement by moving the cursor the second statement and pressing the **F9** key.

```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("Hello, World.  More after the break");
    MessageBox.Show("Let's continue.");

    int zero = 0;
    int answer = 1 / zero;
}
```

- Add code which will generate an exception after the second messagebox statement. A good example would be to create an int, assign the value 0 to it, then divide another number by the 0-assigned integer. Note that the compiler will not allow you to divide by a constant zero.
- Run the application again. After closing the first messagebox, visual studio will reach the breakpoint and stop, allowing you to step through the code. Press the **F10** key to run the next statement
- Press **F10** again to run the next two lines.
- An exception will be triggered when you divide 1 by 0:



5 Interfacing with Native Code

Now we'll engage in a somewhat more difficult activity: Interfacing with a DLL written in native (unmanaged) C++. This is necessary for a number of tasks, especially when it comes to writing code for embedded devices.

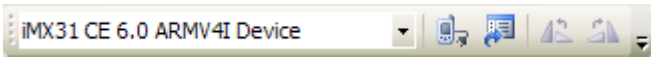
5.1 Open Pre-created Windows Forms Project

Go to the instance of Visual Studio 2005 running your new managed application.

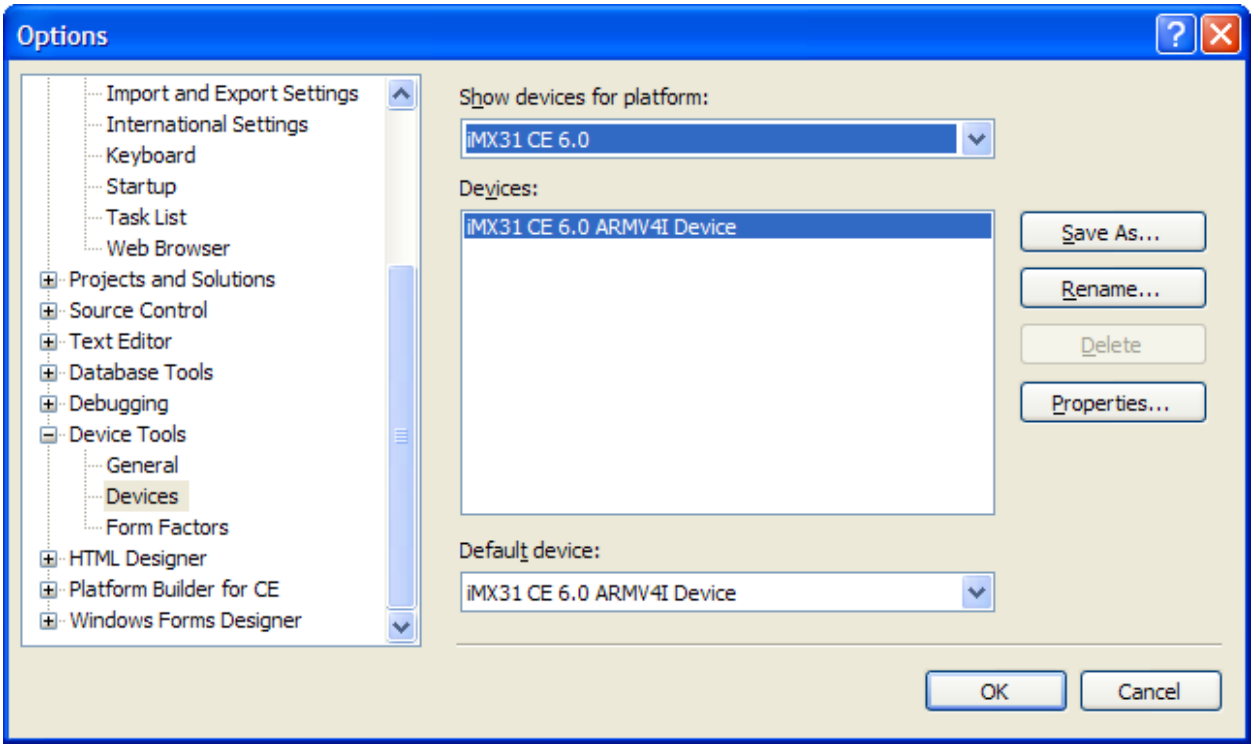
- Close the current application by selecting **File** | **Close Solution**. Save changes if you wish.
- In the same instance of Visual Studio, open the LogoApp application at **C:\Labs\Lab2\LabFiles\LogoApp\LogoApp.sln**
- Set the project's active build configuration to **Release** mode by using the main toolbar or by entering **Build** | **Configuration Manager**

5.2 Deploy to the Target Device

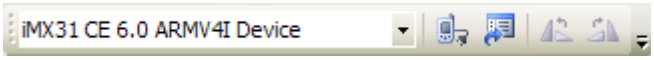
- Configure your target device connectivity options as with the previous part. First open the **Device** toolbar if it isn't open already with **View** | **Toolbars** | **Device**, then select **iMX31 CE 6.0 ARMV4I Device** from the pulldown menu in this toolbar as shown



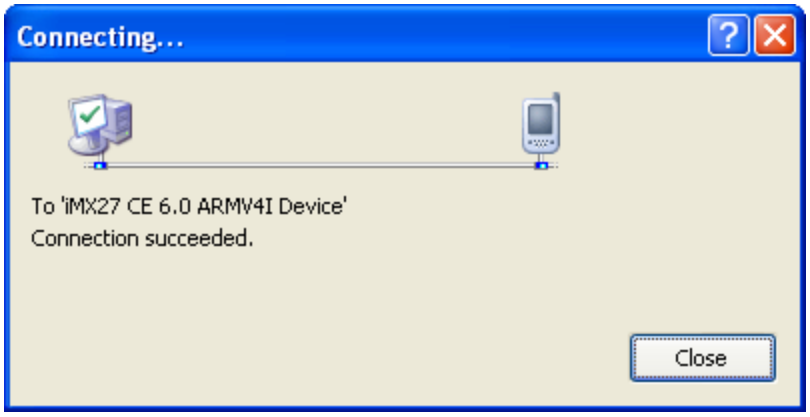
- Click the Device Options ( ) button and at the screen select the **iMX31 CE 6.0 ARMV4I Device** again



- Open the **Properties** screen after selecting **iMX31 CE 6.0 ARMV4I Device**, then select **Configure** and enter your device's IP address
- Click OK at each open window to save the current settings.
- Return to the instance of Visual Studio with your OS Design
- Run **ConManClient2** and then **CMAccept** by using **Target** | **Run Programs** and double clicking them in the list. (Note that if ConManClient2.exe is still running from earlier, you can just run CMAccept again. Running it again doesn't cause any problems, though).
- Return again to the instance of Visual Studio with the LogoApp solution open
- Verify that **iMX31 CE 6.0 ARMV4I Device** is selected in the Target toolbar as in the image below:



- Click the Connect to Device () button to connect to the device

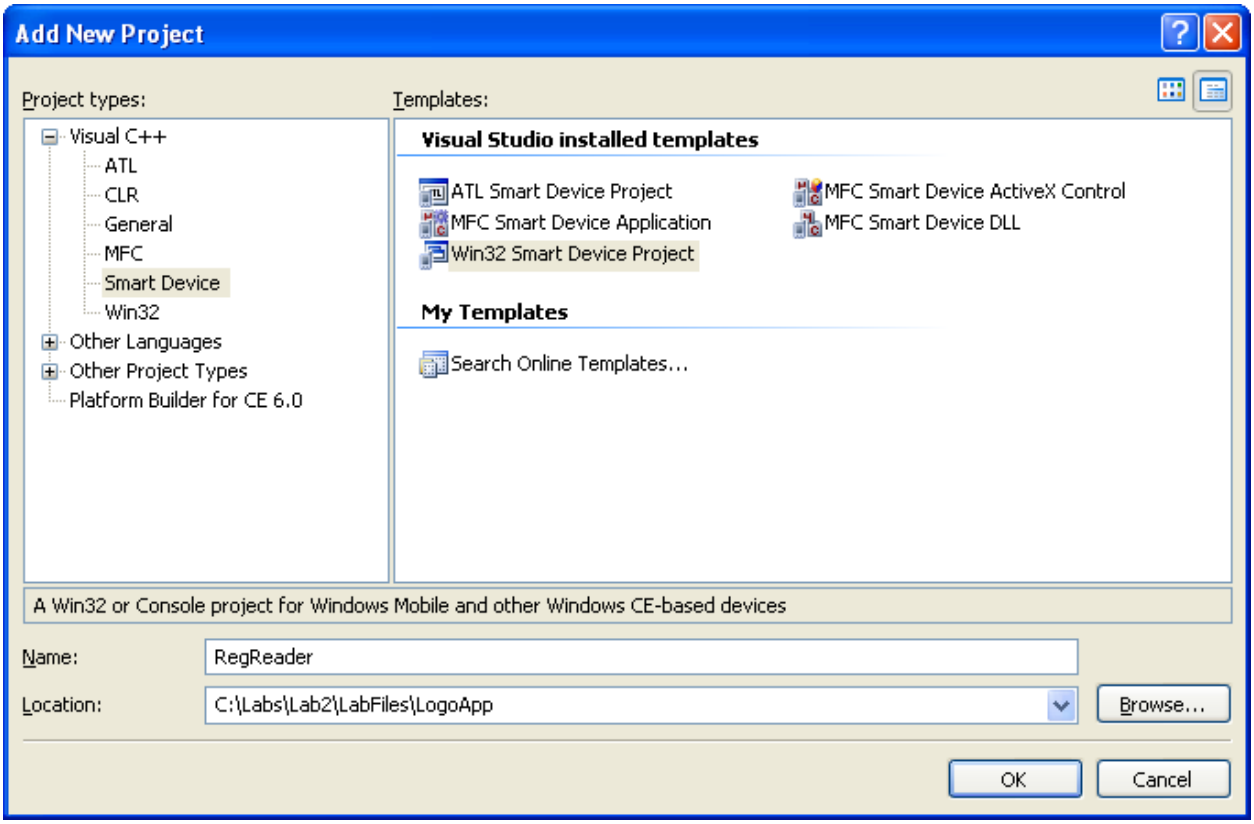


- Select **Debug** | **Start Debugging** to deploy the application to the device and run it
- It should look similar to the following image:

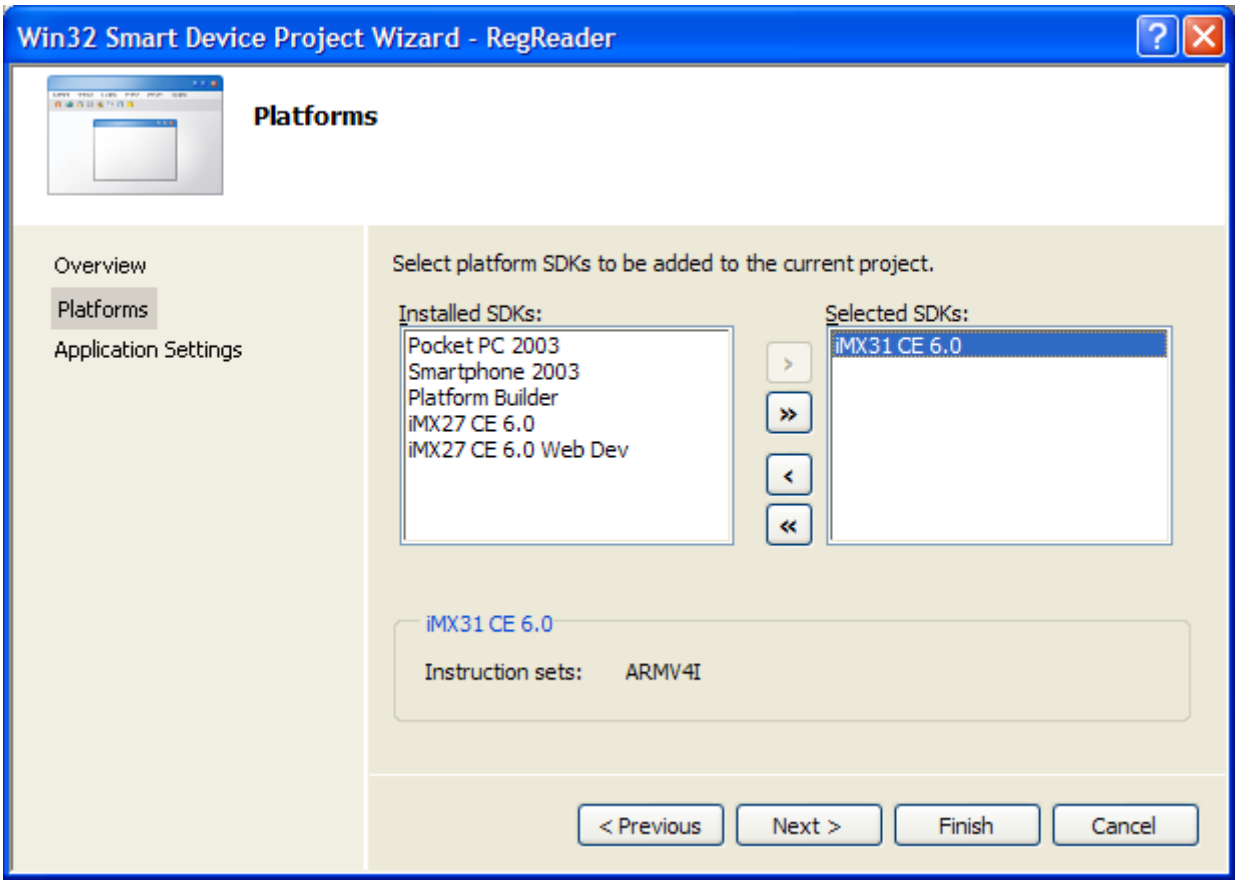
5.3 Create a Native DLL Using a Custom SDK

Now we'll create an unmanaged DLL in Native C++ capable of performing some tasks that are awkward to do directly through C#. The purpose of this section is just to provide a basic overview of how this is done.

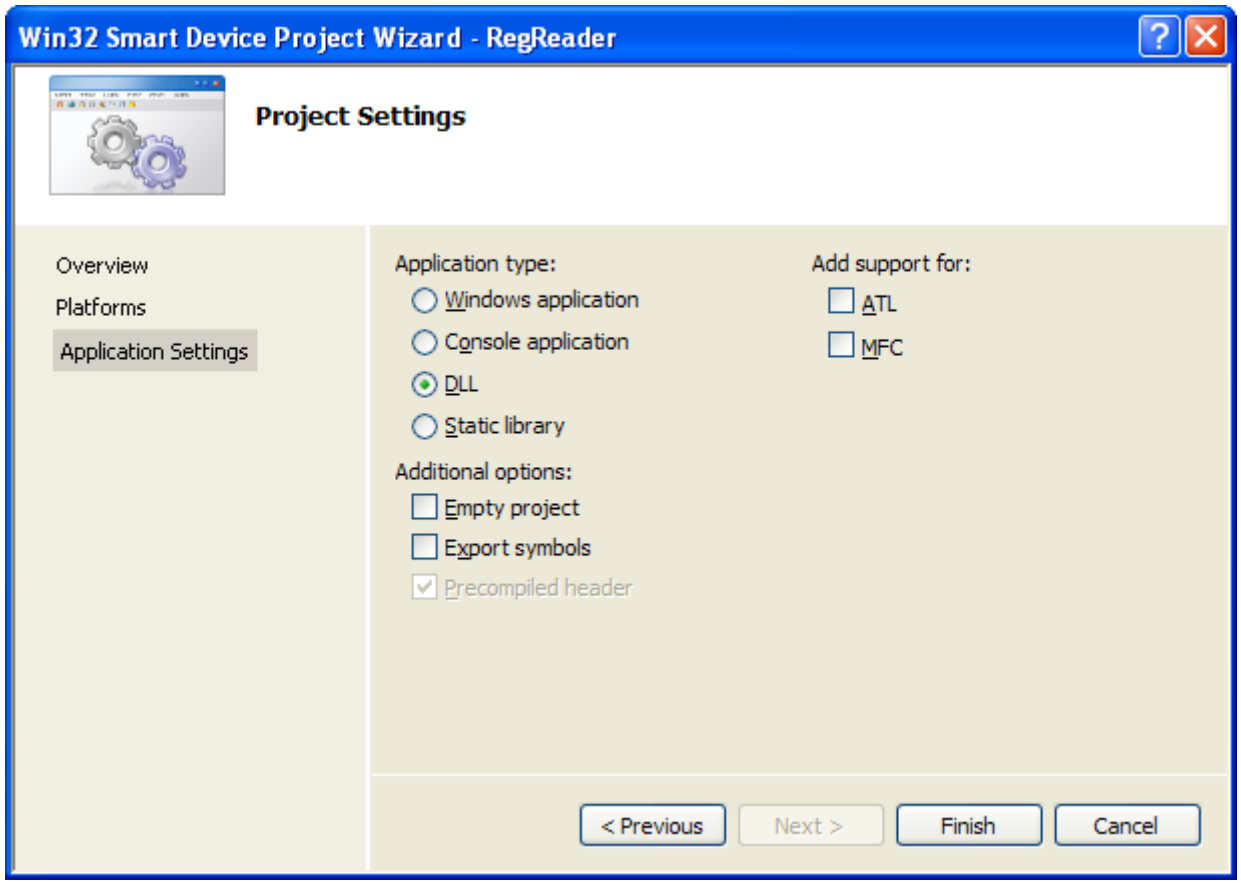
- Stop the running application by selecting **Debug** | **Stop Debugging**
- Right click **Solution 'LogoApp'** and select **Add** | **New Project** from the context menu that pops up.



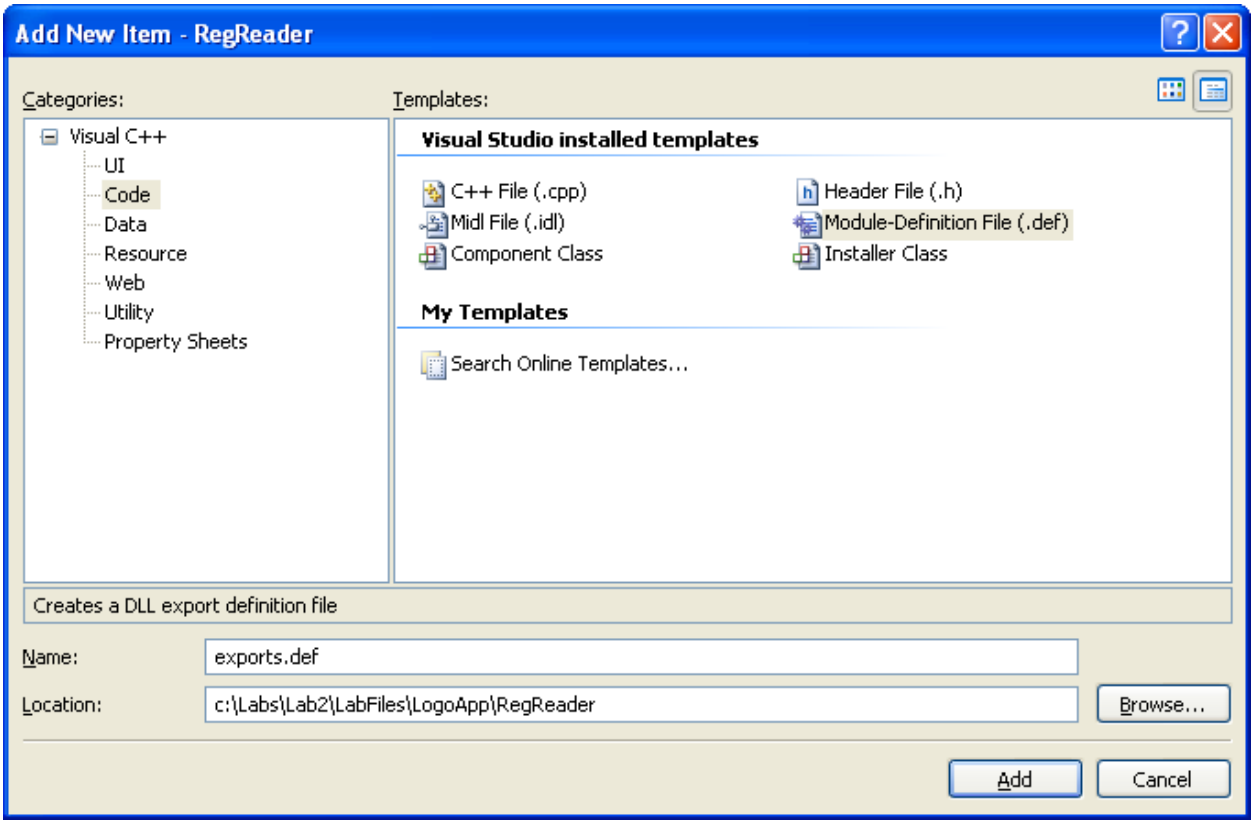
- Create a **Win32 Smart Device Project** under **Visual C++** | **Smart Device** named **RegReader**
- The new project wizard will pop up. Select **Next** at the Overview screen that pops up to go to the Platform Selection screen
- At the Platform Selection screen, select the default platform in the right list and click the left arrow button to remove it from the new project.
- Select the **iMX31 CE 6.0** SDK we created earlier and installed in the left window and click the right arrow button to include it into the new project.
- Afterward it should look like this:



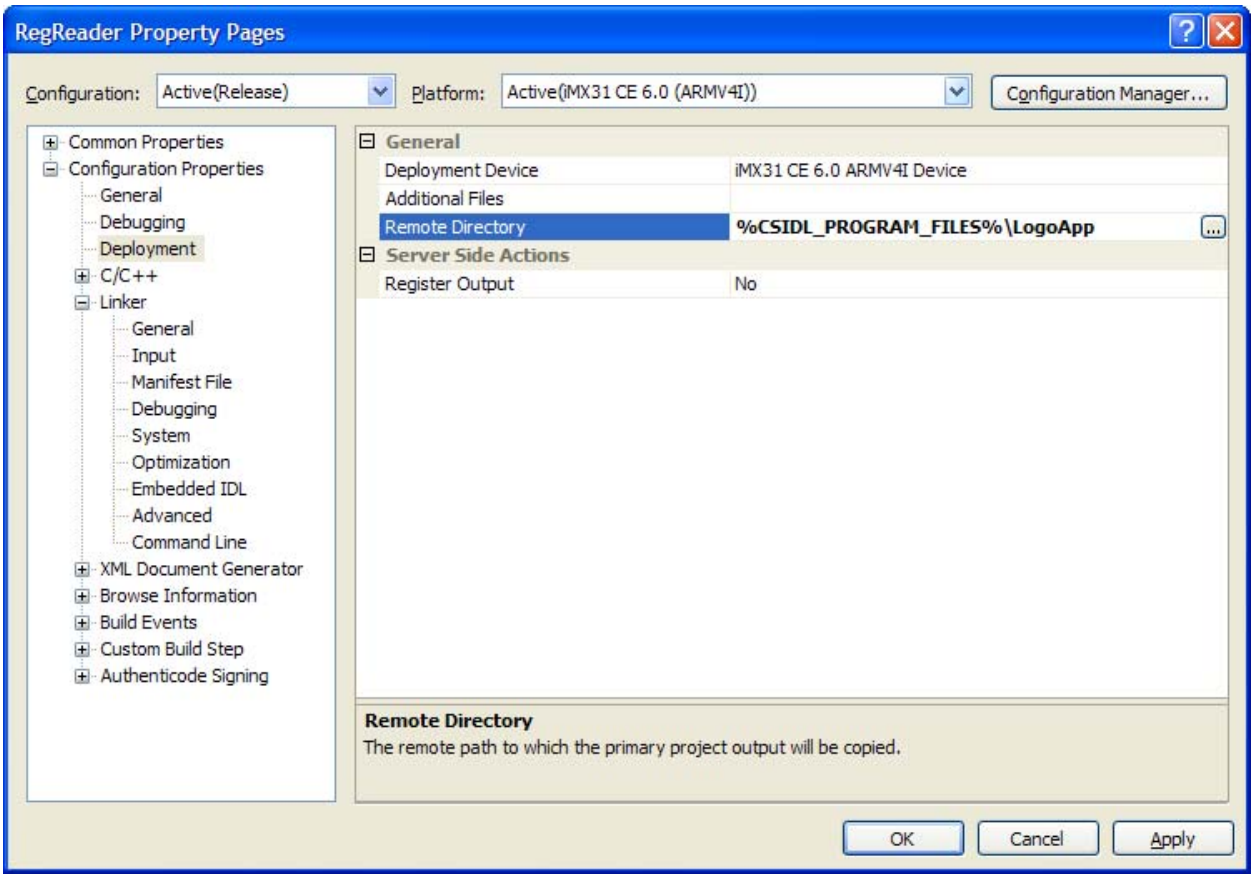
- Click **Next** to continue to the final step of the wizard
- At the final window create a **DLL**, but do **NOT** select “Export symbols”. The template which exports symbols uses the C++ technique for doing so, which is straightforward to import into a C++ application but difficult to make compatible with C# p/invoke techniques



- Click **Finish** to create the DLL project.
- First we'll add an exports.def file to export a function. Right click the new **RegReader** DLL subproject and select **Add** | **New Item** to add a new file
- Create a file of type **Code** | **Module Definitions File** named **exports.def** (or whatever name you wish, just remember to use it in the next step)



- Right click the **RegReader** project again and select **Properties**. In the property pages under **Linker** | **Input** verify that the **Module Definition File** is set to the same name as the .def file you just created
- In the Property Pages under **Deployment** set the **Remote Directory** to the same directory LogoApp.exe is being deployed to, which is **%CSIDL_PROGRAM_FILES%\LogoApp** (just change the RegReader to LogoApp in this field)



- Click **OK** to return to the project
- Double click **RegReader.cpp** and add the following code above the **DllMain** function:

```
//Reads HKLM\Software\LogoApp\DefaultSpeed and returns the value
DWORD getDefaultSpeed()
{
	HKEY hKey;
	LONG lRet;
	DWORD dwType,dwSize;
	DWORD dwDefaultSpeed = 5;

	lRet = RegCreateKeyEx(HKEY_LOCAL_MACHINE, L"SOFTWARE\\LogoApp",
		0, NULL, 0, KEY_ALL_ACCESS, 0, &hKey, NULL);

	if (ERROR_SUCCESS == lRet)
	{
		lRet = RegQueryValueEx(hKey,L"DefaultSpeed",NULL,&dwType,
			(LPBYTE)&dwDefaultSpeed, &dwSize);
```



```
            if (ERROR_FILE_NOT_FOUND == lRet)
            {
                  RETAILMSG(1, (L"RegReader: DefaultSpeed not found"));
            }
            else
            {
                  RETAILMSG(1, (L"RegReader: Obtained speed of %d",
dwDefaultSpeed));
            }
      }
      else
      {
            ERRORMSG(1, (L"RegReader.dll: Could not access key"));
      }

      RegCloseKey(hKey);

      return dwDefaultSpeed;
}
```

- Add the following line to RegReader's **exports.def** file:

```
EXPORTS getDefaultSpeed
```

5.4 Interface with the New DLL

Now we'll add code to LogoApp to use the exported function in RegReader

- Add the following line to LogoApp's **Form1.cs** near the top, below the other **using** statements: (this will use the p/invoke library to access unmanaged code)

```
using System.Runtime.InteropServices;
```

- Next add the following code above the **public Form1()** constructor statement

```
[DllImport("RegReader.dll")]

static extern UInt32 getDefaultSpeed();
```

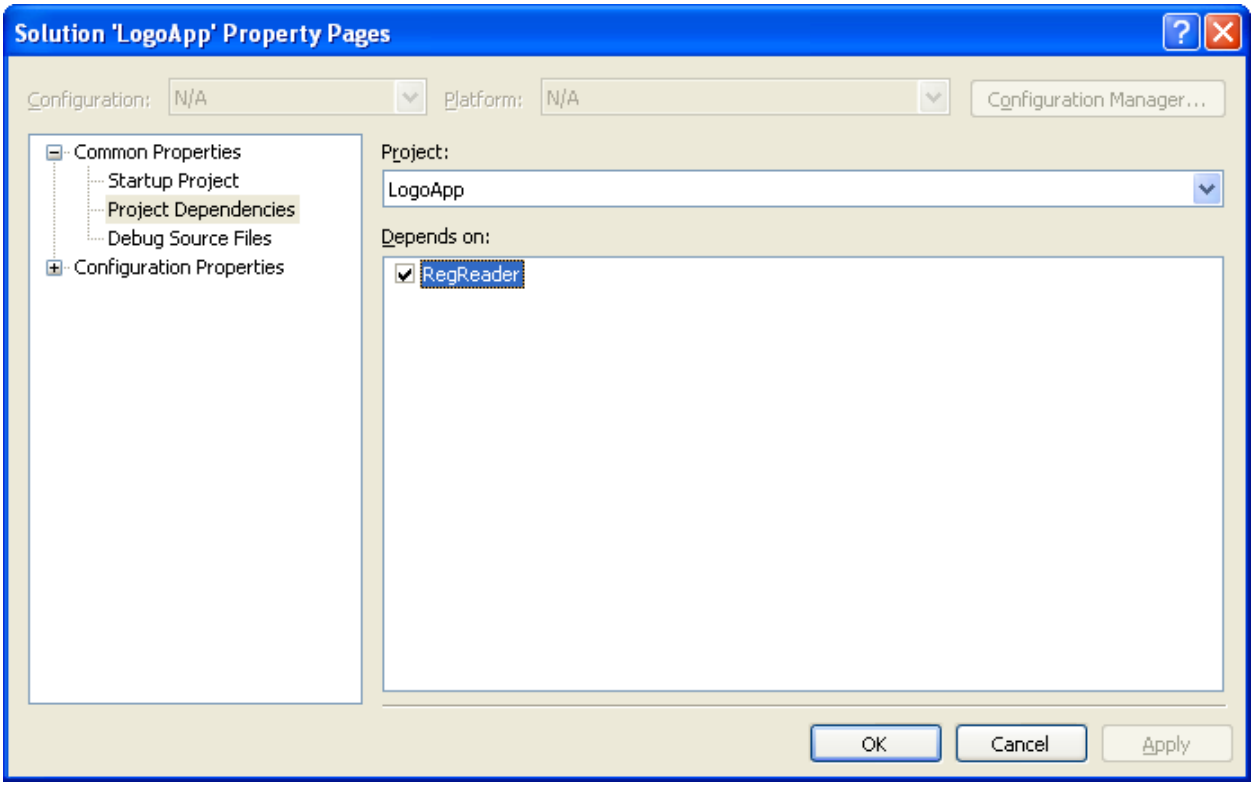
- Now add the following lines to the **Form1**() constructor somewhere between **InitializeComponent()** and the **tmr** code:

```
xVelocity = (int) getDefaultSpeed();

yVelocity = (int) getDefaultSpeed();

MessageBox.Show("Default speed from Registry: " +   getDefaultSpeed());
```

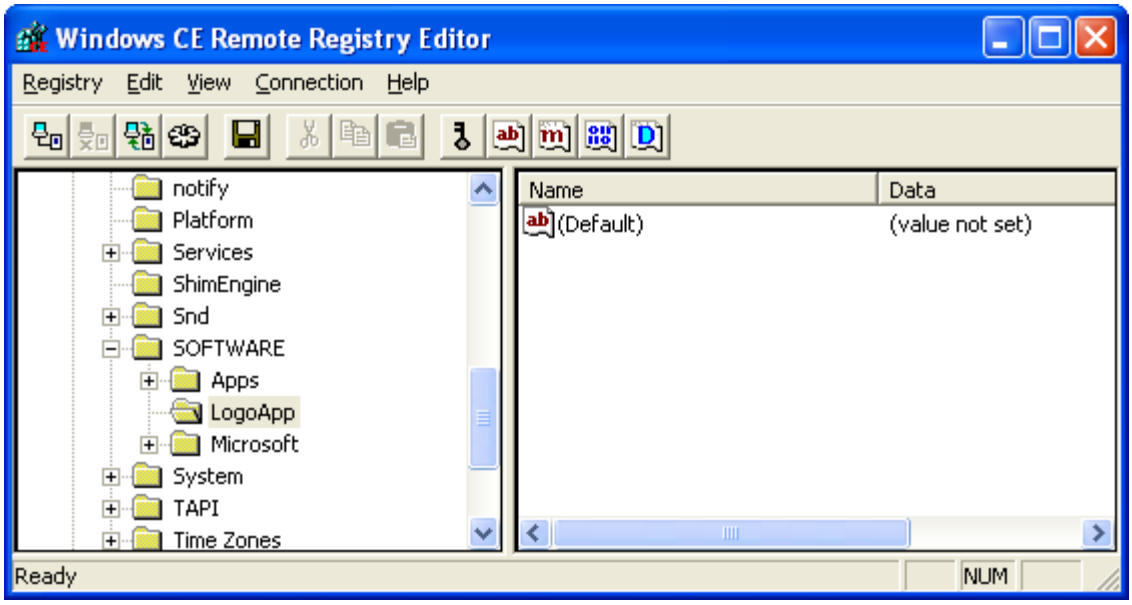

- That's all that should be necessary to call the function from the DLL.
- Now we'll set up the dependencies so that RegReader will be deployed as part of LogoApp. Right click **Solution 'LogoApp'** and select **Properties**
- At the **Common Properties** | **Project Dependencies** screen, select **LogoApp** from the pulldown box and check the box next to **RegReader** to set **RegReader** as a dependency of **LogoApp**, then click OK



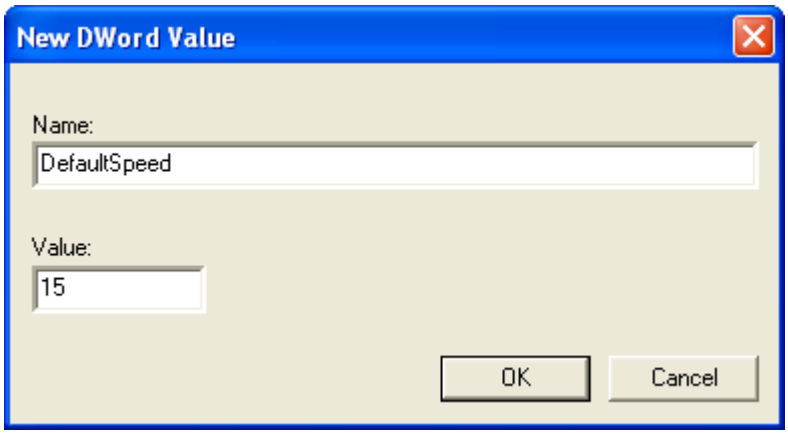
- Run the application again by selecting **Debug** | **Start Debugging**. RegReader should be built and deployed automatically, and then LogoApp built and run again. At startup the application will display a messagebox with **Default speed from registry: 5**
- Observe that the logo moves more quickly now than before
- Close the application by selecting **Debug** | **Stop Debugging**

To verify that our application is reading this value from the registry, we'll use the Registry Editor to modify it and then run the program again.

- Return to the **first instance** of Visual Studio with your OS Design, then start the **Remote Registry Editor** by selecting **Target** | **Remote Tools** | **Registry Editor**
- Navigate to **HKEY_LOCAL_MACHINE\SOFTWARE\LogoApp** and observe that this key was created by the RegReader DLL



- Create a new **DWORD** value by right clicking in the LogoApp key and selecting **New** | **DWORD Value**. Name the new value **DefaultSpeed**



- Set this value to something other than 5, such as 15.
- Return to the instance of **Visual Studio with LogoApp** and run the application again by selecting **Debug** | **Start Debugging**
- Note that the messagebox now says “15”
- Observe that the logo moves even more quickly now than before

Congratulations! You have successfully integrated managed and unmanaged code!

6 Summary

You have now completed all the steps in this guide. Here’s what we have covered:

- Adding the Core Connectivity helper files to an OS Design using a subproject
- Creating and installing a custom SDK based on an OS Design
- Developing with Managed code using the .NET Compact Framework
- Deploying code to a remote device using Visual Studio 2005 and the Core Connectivity framework

- Debugging code on a remote device
- Creating and interfacing with a native C++ DLL from C#

