# Introduction to the .NET Micro Framework

**Description:** The .NET Micro Framework (NETMF) extends the .NET programming model to embedded devices. This enables embedded developers to take advantage of desktop application tools and it enables current .NET developers to now work on devices.

This lab introduces the .NET Micro Framework concepts and shows examples of working at both the application level and the hardware level.

## Lab Sections

1. Lab Setup and Simple Hello World .......................................................... 2
2. Using the debugger ...................................................................................... 14
3. Exploring the class libraries ........................................................................ 17
4. Event Handling ............................................................................................ 19
5. User Interface Controls ................................................................................ 25
6. Porting Kit – building and deploying the TinyCLR .................................... 32
7. TinyCLR – adding components to the TinyCLR ......................................... 43
8. Adding Touch To Applications .................................................................... 47
9. Creating your own class libraries ............................................................... 50
10. DPWS: Hello World Client ......................................................................... 55
11. Simple Service ......................................................................................... 57
12. Putting It All Together .............................................................................. 61

## Skill Level

1. Intermediate
2. You should have some familiarity with the Microsoft .NET Framework and Visual Studio environment
3. Programming experience with C++, C# or other object oriented language

## Lab Materials

Please verify you have the following materials at your lab station.

- Microsoft Visual Studio 2010 Professional Edition
- .NET Micro Framework Platform SDK
- .NET Micro Framework Porting Kit
- Renesas High-performance Embedded Workshop
- SH7264
- Power Cord
- Serial Cable
- Ethernet Cable

## Time to Complete Lab

4 hours
1 Lab Setup and Simple Hello World

Overview:
The first lab session will set up your system and verify you have all the parts required for the rest of the lab.

Procedural Steps

Step 1.1 Board Setup: Verify you have the following components:
- SH7624 board
- 5V/4A DC Power Supply
- Ethernet Cable
- Null modem cable
Step 1.2 Debugger Setup: Verify you have the following components for the debugger:
- E10A USB Emulator
- A/Mini-B USB Cable

Step 1.3 Software Setup: Verify you have the following software components installed on your PC:
- Microsoft Visual Studio 2010 Professional Edition
- .NET Micro Framework Platform SDK
- .NET Micro Framework Porting Kit
- Renesas High-performance Embedded Workshop

On your PC, go to **Start** → **Control Panel** → **Programs** → **Programs and Features**

Look for the following 4 entries:

- **Microsoft Visual Studio 2010 Professional Edition**
  - Microsoft Visual Studio 2010 Professional - ENU
  - Microsoft Visual Studio 2010 Tools for Office Runtime

- **.NET Micro Framework Platform SDK**
  - Microsoft .NET Micro Framework Porting Kit v4.1
  - Microsoft .NET Micro Framework SDK 4.1

- **.NET Micro Framework Porting Kit**
  - Microsoft .NET Micro Framework Porting Kit v4.1
Now that we have verified that the lab setup is complete – let’s start!

For the remainder of the document, the following abbreviations will be used:

VS2010: Microsoft Visual Studio 2010 Professional Edition
NETMF SDK: Microsoft .NET Micro Framework SDK 4.1
NETMF PK: Microsoft .NET Micro Framework Porting Kit v4.1
HEW: High-performance Embedded Workshop

Step 1.4 Connect your device to your PC

Connect one end of the serial cable to your SH7264 – the serial port is located at the bottom of the LCD, near the buttons.

Connect the other end of the serial cable to your PC.

Connect the power supply to the SH7264 – the power adapter is located at the top of the LCD.
Plug the power supply in to the power source.

Step 1.5  Open VS2010

This is located under Start → Microsoft Visual Studio 2010 → Microsoft Visual Studio 2010

Step 1.6  Select File → New → Project...
This opens the **New Project** dialog.

**Step 1.7** Under **Installed Templates** select **Micro Framework**.

There are four project types:
We will revisit each one later.

Step 1.8  Select “Console Application”

Under **Name**: the default name will be MFConsoleApplication1
Under **Location**: the default location will be `c:\users\xxxx\documents\visual studio 2010\Projects`
Under **Solution name**: the default name will be MFConsoleApplication1

Step 1.9  **Click OK**
Step 1.10 If not already visible, display the Solution Explorer by selecting View → Solution Explorer.

Step 1.11 In Solution Explorer, right-click MFConsoleApplication1. Select Properties to view the Application properties.
Step 1.12 Select .NET Micro Framework.
Step 1.13  Change the Transport to Serial

The **Device** will be set to **COM1**.
Step 1.14  Build your solution by clicking **Build → Build Solution**

The **Output** window will display the following:

![Output window screenshot]

1. The highlighted button toggles the word wrap on contents of Output window. This is especially useful with very long paths!

Step 1.15  Run your solution on the hardware by clicking **Debug → Start Debugging**
You can also start debugging by pressing **F5** or using the green arrow toolbar button:
The **Output** window will display the following:

```
The thread '<No Name>' (0x2) has exited with code 0 (0x0).
Hello World!
Done.

The thread '<No Name>' (0x1) has exited with code 0 (0x0).
The program '[4] Micro Framework Application: Managed' has exited with code 0 (0x0).
Waiting for debug commands...
```

Exercise 1 Complete!

Rotate your dice to 1 to show your progress.
2 Using the debugger

Overview:
We will use the Visual Studio debugger and explore some of the features available for debugging.

Procedural Steps

Step 2.1 Using the project created in Exercise 1, in Solution Explorer double click on Program.cs.
When we created our Console Application project in the last exercise, the New Project Wizard creates the project and solution, with the main code in Program.cs:

```csharp
using System;
using Microsoft.SPOT;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            Debug.Print(
                Resources.GetString(Resources.StringResources.String1));
        }
    }
}
```

The namespace is called Microsoft.SPOT because the .NET Micro Framework was formerly known as Smart Personal Objects Technology (SPOT).

The Main method is the entry point of your program. It must be static and can either have a void or int return type.

Add the following namespace to the top of the file:

```csharp
using Microsoft.SPOT.Hardware;
```

Add the following code to the Main method:

```csharp
Version ver = SystemInfo.Version;
Debug.Print("Version Information");
Debug.Print("Major: " + ver.Major.ToString());
Debug.Print("Minor: " + ver.Minor.ToString());
Debug.Print("Big Endian: " + SystemInfo.IsBigEndian.ToString());
```
Step 2.2  Start debugging by pressing F5

Step 2.3  Set a breakpoint:

Step 2.4  Start debugging again by pressing F5

Once the debugger breaks at the line, you can review the information for ver

Step 2.5  Hover over the variable ver
Clicking “+” sign will show variable properties. Clicking the push-pin will retain the variable details.

**Question**

1. What line could be added to display build number?

Exercise 2 Complete!

Rotate your dice to 2 to show your progress.
3 Exploring the class libraries

Overview:
We will explore and some of the class libraries available in the TinyCLR.

Procedural Steps

Step 3.1 Add Microsoft.SPOT.Net to the project solution:

In Solution Explorer right-click on References and select Add Reference…

Select Microsoft.SPOT.Net in the Add Reference dialog
Add the following namespace to the top of the file:

```csharp
using Microsoft.SPOT.Net.NetworkInformation;
```

And add the following code to `Main()`

```csharp
public static void Main()
{
    NetworkInterface[] arNIs = GetAllNetworkInterfaces();

    foreach (NetworkInterface net in arNIs)
    {
        Debug.Print("DHCP Enabled: "+ net.IsDhcpEnabled);
        Debug.Print("IP Address : "+ net.IPAddress);
        Debug.Print("Subnet      : "+ net.SubnetMask);
        Debug.Print("Gateway     : "+ net.GatewayAddress);
    }
}
```

Step 3.2 Start debugging by pressing F5

Exercise 3 Complete!

Rotate your dice to 3 to show your progress.
4 Event Handling

Overview:
We will revisit Hello World and introduce event handling in the .NET Micro Framework.

Procedural Steps

Step 4.1 Select File \(\rightarrow\) New \(\rightarrow\) Project…

This opens the New Project dialog.
Step 4.2 Under *Installed Templates* select *Micro Framework*.

Step 4.3 Select *"Window Application"*

Under **Name**: the default name will be MFWINDOWAPPLICATION1
Under **Location**: the default location will be c:\users\xxxx\documents\visual studio 2010\Projects
Under **Solution name**: the default name will be MFWindowApplication1

**Step 4.4**  Click OK

**Step 4.5**  In **Solution Explorer**, right-click MFWindowApplication1. Select **Properties** to view the Application properties.

**Step 4.6**  Select **.NET Micro Framework**.

**Step 4.7**  Change the **Transport** to **Serial**

**Step 4.8**  The **Device** will be set to **COM1**.

**Step 4.9**  Build your solution by clicking **Build → Build Solution**

**Step 4.10**  Start debugging by pressing **F5**

The display on the SH7264 will display "Hello World!"

Now let’s add some button feedback.

In **Solution Explorer**, double click on **GPIOButtonInputProvider.cs**

![Solution Explorer Screenshot](image)

The New Project Wizard created this file to manage mapping GPIO inputs to button events. Contained in this file are two classes:

- **GPIOButtonInputProvider**
- **ButtonPad**
The constructor contains the following code snippet:

```csharp
// Create the pins that are needed for the buttons. Default their
// values for the emulator.
Cpu.Pin pinLeft = Cpu.Pin.GPIO_Pin0;
Cpu.Pin pinRight = Cpu.Pin.GPIO_Pin1;
Cpu.Pin pinUp = Cpu.Pin.GPIO_Pin2;
Cpu.Pin pinSelect = Cpu.Pin.GPIO_Pin3;
Cpu.Pin pinDown = Cpu.Pin.GPIO_Pin4;

// Use the hardware provider to get the pins. If the left pin is
// not set, assume none of the pins are set, and set the left pin
// back to the default emulator value.
if ((pinLeft = hwProvider.GetButtonPins(Button.VK_LEFT)) ==
    Cpu.Pin.GPIO_NONE)
    pinLeft = Cpu.Pin.GPIO_Pin0;
else
{
    pinRight = hwProvider.GetButtonPins(Button.VK_RIGHT);
    pinUp = hwProvider.GetButtonPins(Button.VK_UP);
    pinSelect = hwProvider.GetButtonPins(Button.VK_SELECT);
    pinDown = hwProvider.GetButtonPins(Button.VK_DOWN);
}

// Allocate button pads and assign the (emulated) hardware pins as
// input from specific buttons.
ButtonPad[] buttons = new ButtonPad[]
{
    // Associate the buttons to the pins as discovered or set above
    new ButtonPad(this, Button.VK_LEFT , pinLeft),
    new ButtonPad(this, Button.VK_RIGHT , pinRight),
    new ButtonPad(this, Button.VK_UP   , pinUp),
    new ButtonPad(this, Button.VK_SELECT, pinSelect),
    new ButtonPad(this, Button.VK_DOWN , pinDown),
};
```

Step 4.11 Build your solution by clicking Build ➔ Build Solution

Step 4.12 Start debugging by pressing F5

Step 4.13 Tap the bottom left button – the Debug Window will report “40” every time you tap the button

The SH7264 has a 4 x 4 grid of buttons. To abstract the hardware dependent codes to the operating system, the .NET Micro Framework maps the hardware dependent codes to virtual-key codes. A virtual-key code is a hardware-independent number that identifies the key.
Inside the .NET Micro Framework, the SH7264 buttons are currently mapped to the following virtual keys:

<table>
<thead>
<tr>
<th>VK_MENU</th>
<th>VK_SELECT</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>VK_LEFT</td>
<td>VK_RIGHT</td>
<td></td>
</tr>
<tr>
<td>VK_UP</td>
<td>VK_HOME</td>
<td></td>
</tr>
<tr>
<td>VK_DOWN</td>
<td>VK_BACK</td>
<td></td>
</tr>
</tbody>
</table>

**Question**

2. How would you modify this code to add buttons for the additional virtual key codes of VK_MENU, VK_HOME, VK_BACK?

**Step 4.14  Modifying event handling**

The class `ButtonPad` connects GPIO interrupt handling to Microsoft.SPOT.Hardware.Button events.

Program.cs then adds an event handler for the mainWindow to respond to button events:

```csharp
// Connect the button handler to all of the buttons.
mainWindow.AddHandler(Buttons.ButtonUpEvent, new RoutedEventHandler(OnButtonUp), false);
```
The default definition for OnButtonUp prints the value of the virtual-key code to the debug window.

```csharp
private void OnButtonUp(object sender, RoutedEventArgs evt)
{
    ButtonEventArgs e = (ButtonEventArgs)evt;
    // Print the button code to the Visual Studio output window.
    Debug.Print(e.Button.ToString());
}
```

Step 4.15 Modify the code to handle OnButtonDown

Step 4.16 Inside OnButtonDown, modify the code to conditionally execute dependent on the button pressed:

For example:

```csharp
switch (e.Button)
{
    case Button.VK_UP:
        Debug.Print("Menu button pressed");
        break;
    case Button.VK_DOWN:
        Debug.Print("Select button pressed");
        break;
    default:
        Debug.Print("Unknown button pressed");
        break;
}
```

Step 4.17 Build your solution by clicking Build → Build Solution

Step 4.18 Start debugging by pressing F5

Step 4.19 Now as you press the buttons, the Debug Window will report which button was pressed

Exercise 4 Complete!

Rotate your dice to 4 to show your progress.
5 User Interface Controls

Overview:
We will revisit Hello World and introduce the user interface controls available in the .NET Micro Framework.

Procedural Steps

In Program.cs, the function CreateWindow creates a new Window and assigns a text control as its Child.

```
public Window CreateWindow()
{
    // Create a window object and set its size to the size of the display.
    mainWindow = new Window();
    mainWindow.Height = SystemMetrics.ScreenHeight;
    mainWindow.Width = SystemMetrics.ScreenWidth;

    // Create a single text control.
    Text text = new Text();
    text.Font = Resources.GetFont(Resources.FontResources.small);
    text.TextContent = Resources.GetString(Resources.StringResources.String1);
    text.HorizontalAlignment = Microsoft.SPOT.Presentation.HorizontalAlignment.Center;
    text.VerticalAlignment = Microsoft.SPOT.Presentation.VerticalAlignment.Center;

    // Add the text control to the window.
    mainWindow.Child = text;
}
```

Step 5.1 In this exercise, we will add some additional controls to the screen

mainWindow.Child can be assigned any object derived from UIElement. This includes container classes: Canvas, Panel. Etc.

Step 5.2 Create a new file for the project. In Solution Explorer, right click on MFWindowApplication1 and select Add → New Item...
In the **Add New Item** dialog, select **Visual C#** as the **Installed Templates** and select **Class**. This will create an empty class definition. In the **Name:** call this file **MyLayout.cs**
Step 5.3  Enter the following code in the newly created C# file.

```csharp
using Microsoft.SPOT.Presentation.Controls;

namespace MFWindowApplication
{
    public class MyLayout : Canvas
    {
        public MyLayout()
        {
            // Create a single text control.
            Text text = new Text();

            text.Font = Resources.GetFont(Resources.FontResources.small);
            text.TextContent = Resources.GetString(Resources.StringResources.String1);
            this.Children.Add(text);
        }
    }
}
```

Step 5.4  Replace the following code Program.cs → CreateWindow

```csharp
// Create a single text control.
Text text = new Text();

text.Font = Resources.GetFont(Resources.FontResources.small);
```
Step 5.5  Build your solution by clicking **Build → Build Solution**

Step 5.6  Start debugging by pressing **F5**

The display on the SH7264 will display "Hello World!" in the top-left of the screen.

Let's add some more controls to the screen!

Step 5.7  In Solution Explorer, double click on Resources.resx to edit the resources. The resource editor will open, listing the strings in the project.

Add a resource by clicking the drop-down next to **Add Resource**, then selecting **New Image → BMP Image**...
By default, the name of the new resource will be **Image1**. Click **Add**.

Visual Studio will create a 48x48, 8 bit BMP file.

**Step 5.8** Draw an up arrow using the image editing tools available in the **Image Editor** tool bar or on the **Image** menu
Step 5.9  Save the image

Step 5.10  Repeat the process to create a down arrow called Image2

Step 5.11  Modify the code in MyLayout.cs to the following:

```csharp
using Microsoft.SPOT.Presentation.Controls;

namespace MFWindowApplication
{
    public class MyLayout : StackPanel
    {
        public MyLayout()
        {
            // Create a single text control.
            Text text = new Text();

            text.Font = Resources.GetFont(Resources.FontResources.small);
            text.TextContent = Resources.GetString(Resources.StringResources.String1);
            this.Children.Add(text);

            Image img = new Image();
            img.Bitmap = Resources.GetBitmap(Resources.BitmapResources.Image1);
            this.Children.Add(img);
        }
    }
}
```
Exercise 5 Complete!

Rotate your dice to 5 to show your progress.

```csharp
Image img = new Image();
img.Bitmap = Resources.GetBitmap(Resources.BitmapResources.Image2);
this.Children.Add(img);
}
}
```

**Question**

3. How would you change this to a different layout?
Overview:
We will create a class library for use in a later project.

Procedural Steps

Step 6.1 Create a build shortcut

On the desktop, right click and select **New → Shortcut**

In the dialog, either **Browse** to or type the location of cmd.exe:

```
C:\windows\System32\cmd.exe
```
Click next. Type a name for this shortcut. E.g. Build SH7264

What would you like to name the shortcut?

Type a name for this shortcut:
Build SH7264

Click Finish.
On the newly created shortcut, right-click and select Properties.
Edit **Target** to: %SystemRoot%\system32\cmd.exe /K setenv_shc.cmd
c:\Progra~1\Renesis\HEW\Tools\Renesis\Sh\9_3_2

Edit **Start In** to: c:\MicroFrameworkPK_v4_1

![Build SH7264 Properties](image)

Press **OK**.

**Step 6.2** Double click your newly made shortcut
This creates the command line environment for your build system.

```bash
cd Solutions\SH7264_RSK
```

![Setting environment for using Microsoft Visual Studio 2010 x86 tools](image)

Run the build command:

```bash
Msbuild /t:build /p:flavor=debug;memory=flash;QVGALCD=false;EnableTCPIP=true
```

While TinyCLR is building, ready the device downloading the new TinyCLR!
Step 6.3  Disconnect power to the SH7264

Step 6.4  Make sure the E10A-USB emulator is not connected to the PC

Step 6.5  Connect the E10A-USB emulator to the SH7264 by plugging it into the E10A connector

Step 6.6  Plug the USB cable into the E10A-USB emulator

Step 6.7  Plug the USB cable into the desktop

Step 6.8  Power on the SH7264

Wait until the build completes before continuing

Step 6.9  Run the **High-performance Embedded Workshop (HEW)**. The program shortcut is available at Start → Renesas → High-performance Embedded Workshop.
In the Welcome dialog, select “Browse to another project workspace”.

Browse to c:\MicroFrameworkPK_v4_1\Solutions\SH7264_RSK\doc\SH7264RSK
Select **SH7264RSK.hws** – press **Select**.

Select the Emulator mode: **E10A-USB Emulator**

In **Driver Details**, select **Renesas E-Series USB Driver**
After a few seconds, **Interface** and **Channel** should auto-fill.

Click **Close**

If the E10A-USB Emulator is communicating, the following dialog will pop up:
Push Reset on the SH7264 board (button labeled RST) and click OK, or press Enter key.

Progress is shown

In the Projects window, right click on TinyCLR.abs and select Download
Wait for progress to complete

The flash has been programmed. Disconnect the USB emulator by selecting Debug → Disconnect
Introduction to the .NET Micro Framework
Step 6.10  Unplug the USB connection to your desktop.

Step 6.11  Power off the SH7264.

Step 6.12  Disconnect the E10A from the board.

Step 6.13  Power on the SH7264.

When TinyCLR has booted it will report on the LCD:

Waiting for debug commands...

Exercise 6 Complete!

Rotate your dice to 6 to show your progress.
7 TinyCLR – adding components to the TinyCLR

Overview:
We will add touch support to the TinyCLR to show the key files in the TinyCLR build system

Procedural Steps

Step 7.1 Modify SH7264_RSK.settings located in
C:\MicroFrameworkPK_v4_1\Solutions\SH7264_RSK; and add the highlighted lines

```xml
<Project ToolsVersion="4.0" DefaultTargets="Build"
xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <Author>Jennifer Cheng</Author>
    <Description>SH7264_RSK solution</Description>
    <Documentation/>
    <PlatformGuid>{e76ba7b4-d7c0-4c68-b027-6c88ca5aabaf}</PlatformGuid>
    <TARGETPLATFORM>SH7264_RSK</TARGETPLATFORM>
    <PLATFORM>SH7264_RSK</PLATFORM>
    <DEVICE_TYPE Condition=" '$(DEVICE_TYPE)' == ''">sh2afpu</DEVICE_TYPE>
    <IsSolutionWizardVisible>True</IsSolutionWizardVisible>
    <ENDIANNESS>Be</ENDIANNESS>
    <DEVICE_TYPE Condition=" '$(DEVICE_TYPE)' == ''">SH2A</DEVICE_TYPE>
    <EnableTCPIP Condition=" '$(EnableTCPIP)' == ''">false</EnableTCPIP>
    <FLASH4M Condition=" '$(FLASH4M)' == ''">false</FLASH4M>
    <QVGALCD Condition=" '$(QVGALCD)' == ''">true</QVGALCD>
    <SERIALFLASH Condition=" '$(SERIALFLASH)' == ''">false</SERIALFLASH>
    <TOUCHPANEL Condition=" '$(TOUCHPANEL)' == ''">false</TOUCHPANEL>
  </PropertyGroup>
  <ItemGroup>
    <IncludePaths Include="Solutions\SH7264_RSK" />
  </ItemGroup>
  <ItemGroup>
    <CC_CPP_MACRO_FLAGS Include="COPY_SECTION_P"
Condition=" '$(SERIALFLASH)' == false"
Include="SH7264_RSK_SERIAL_FLASH"
Condition=" '$(SERIALFLASH)' == true"
Include="SH7264_RSK_SERIAL_FLASH"
Condition=" '$(FLASH4M)' == false"
Include="SH7264_RSK_FLASH4M"
Condition=" '$(FLASH4M)' == true"
Include="SH7264_RSK_FLASH4M"
Condition=" '$(QVGALCD)' == false"
Include="SH7264_RSK_QVGA_LCD"
Condition=" '$(QVGALCD)' == true"
Include="SH7264_RSK_QVGA_LCD"
Condition=" '$(EnableTCPIP)' == false"
Include="SH7264_RSK_ENABLE_TCPIP"
Condition=" '$(EnableTCPIP)' == true"
Include="SH7264_RSK_ENABLE_TCPIP"
Condition=" '$(TOUCHPANEL)' == false"
Include="SH7264_RSK_TOUCH_PANEL"
Condition=" '$(TOUCHPANEL)' == true"
  </ItemGroup>
  <Import Project="$(SPOCLIENT)\devicecode\Targets\Native\SH7264\SH7264.settings" />
</Project>
```
Step 7.2 Modify TinyCLR.proj located in C:\MicroFrameworkPK_v4_1\Solutions\SH7264_RSK\TinyCLR as follows:

Locate the section that imports .featureproj files and add:

```xml
<Import Project="$(SPOCLIENT)\Framework\Features\TouchScreen.featureproj" />
<Import Project="$(SPOCLIENT)\Framework\Features\Ink.featureproj" />
<Import Project="$(SPOCLIENT)\Framework\Features\Gesture.featureproj" />
```

Replace the lines:

```xml
<ItemGroup>
  <RequiredProjects Include="$(SPOCLIENT)\DeviceCode\Pal\Gesture\stubs\dotNetMF.proj" />
  <DriverLibs Include="Gesture_pal_stubs.$(LIB_EXT)" />
</ItemGroup>
```

With:

```xml
<ItemGroup>
  <RequiredProjects Include="$(SPOCLIENT)\DeviceCode\Pal\Gesture\dotNetMF.proj" />
  <DriverLibs Include="Gesture_pal.$(LIB_EXT)" />
</ItemGroup>
```

Replace the lines:

```xml
<ItemGroup>
  <RequiredProjects Include="$(SPOCLIENT)\DeviceCode\Pal\Ink\stubs\dotNetMF.proj" />
  <DriverLibs Include="Ink_pal_stubs.$(LIB_EXT)" />
</ItemGroup>
```

With:

```xml
<ItemGroup>
  <RequiredProjects Include="$(SPOCLIENT)\DeviceCode\Pal\Ink\dotNetMF.proj" />
  <DriverLibs Include="Ink_pal.$(LIB_EXT)" />
</ItemGroup>
```
Include Touch project files:

```xml
<ItemGroup>
  <RequiredProjects
Include="$(SPOCLIENT)\CLR\Libraries\SPOT_Touch\dotnetmf.proj" />
  <PlatformIndependentLibs Include="SPOT_Touch.$(LIB_EXT)" />
</ItemGroup>
<ItemGroup>
  <RequiredProjects
Include="$(SPOCLIENT)\DeviceCode\Pal\TouchPanel\dotNetMF.proj" />
  <DriverLibs Include="TouchPanel_pal.$(LIB_EXT)" />
</ItemGroup>
<ItemGroup>
  <RequiredProjects
Include="$(SPOCLIENT)\DeviceCode\Targets\Native\sh7264\DeviceCode\dotNetMF.proj" />
  <DriverLibs Include="SH7264_TouchPanel.$(LIB_EXT)" />
</ItemGroup>
<ItemGroup>
  <RequiredProjects
Include="$(SPOCLIENT)\Solutions\SH7264_RSK\DeviceCode\TouchPanel\dotNetMF.proj" />
  <DriverLibs Include="SH7264RSK_TouchPanel_Config.$(LIB_EXT)" />
</ItemGroup>
```
Modify SH7264_functions_INTC.cpp located in C:\MicroFrameworkPK_v4_1\DeviceCode\Targets\Native\sh7264\DeviceCode\INTC

```c
// 64 Interrupt IRQ0
void INT_IRQ0(void)
{
    #ifdef SH7264_TOUCH_PANEL
        extern void SH7264_TOUCH_ISR(void);
        SH7264_TOUCH_ISR();
    #endif
    /* sleep() */
}
```

Run the build command:

```
Msbuild /t:build /p:flavor=debug;memory=flash;TOUCHPANEL=true;QVGALCD=false;EnableTCP=ture
```

Repeat the steps in Exercise 6 to reflash the new TinyCLR to the device.
When TinyCLR has booted it should report on the LCD:

```
Waiting for debug commands...
```

Exercise 7 Complete!

Rotate your dice to 1 to show your progress.
8 Adding Touch To Applications

Step 8.1 Now that we have touch – let’s revisit our window application and add touch

Step 8.2 In Program.cs of our MFWindowApplication1 add

```csharp
Touch.Initialize(myApplication);
```

Main() will look like this:

```csharp
public static void Main()
{
    Program myApplication = new Program();
    Touch.Initialize(myApplication);
    Window mainWindow = myApplication.CreateWindow();
    // Create the object that configures the GPIO pins to buttons.
    GPIOButtonInputProvider inputProvider = new GPIOButtonInputProvider(null);
    // Start the application
    myApplication.Run(mainWindow);
}
```

The UIElement::TouchUp event occurs when a finger is raised off a UI element. We will modify the code so that the two images respond to touch events.

Step 8.3 Create a new class that inherits from Image

```csharp
public class MyImage : Image
{
}
```

Step 8.4 Override the UIElement::TouchUp event

```csharp
public class MyImage : Image
{
    // Handle the touch up event
    protected override void OnTouchUp(TouchEventArgs e)
    {
    }
}
```
Step 8.5 Add a Name property so that we can easily identify which MyImage is touched

```csharp
public class MyImage : Image
{
    public string Name;

    // Handle the touch up event
    protected override void OnTouchUp(TouchEventArgs e)
    {
        base.OnTouchUp(e);
        Debug.Print(Name + " pressed");
    }
}
```

Step 8.6 Now modify MyLayout::MyLayout()

```csharp
public class MyLayout : StackPanel
{
    public MyLayout()
    {
        // Create a single text control.
        Text text = new Text();
        text.Font = Resources.GetFont(Resources.FontResources.small);
        text.TextContent = Resources.GetString(Resources.StringResources.String1);
        this.Children.Add(text);

        MyImage img = new MyImage();
        img.Name = "Up";
        img(Bitmap = Resources.GetBitmap(Resources.BitmapResources.Image1);
        this.Children.Add(img);

        MyImage img2 = new MyImage();
        img2.Name = "Down";
        img2(Bitmap = Resources.GetBitmap(Resources.BitmapResources.Image2);
        this.Children.Add(img2);
    }
}
```
Step 8.7  Build your solution by clicking **Build ➔ Build Solution**

Step 8.8  Start debugging by pressing **F5**

Step 8.9  Press the Up and Down buttons on the screen and watch the debug output

Exercise 8 Complete!

Rotate your dice to 2 to show your progress.
9 Creating your own class libraries

Overview:
We will create a class library for use in a later project.

Procedural Steps

Step 9.1 Select File → New → Project…

This opens the New Project dialog.
Step 9.2  Under **Installed Templates** select **Micro Framework**.

Step 9.3  Select “Class Library”

Under **Name:** type **PrizeWinner**
Under **Location:** the default location will be `c:\users\xxxx\documents\visual studio 2010\Projects`
Under **Solution name:** type **PrizeWinner**
Step 9.4  Create a member function

```csharp
public string GetPrizeCode()
```

This member function will return a unique string to your device.

**Question**

4. What type of information can we use on the device to uniquely identify it?

Implement a unique string for your device. One solution is on the next page!
We will implement a unique string by using the MAC address of the Ethernet chip.

```csharp
using System;
using Microsoft.SPOT;
using Microsoft.SPOT.Net.NetworkInformation;

namespace PrizeWinner
{
    public class Prize
    {
        string PrizeCode;

        public Prize()
        {
            // Read Ethernet MAC address
            NetworkInterface ni;
            ni = NetworkInterface.GetAllNetworkInterfaces()[0];
            PrizeCode = "";
            for (int i = 0; i < ni.PhysicalAddress.Length; i++)
            {
                PrizeCode += ni.PhysicalAddress[i].ToString();
            }
        }

        public string GetPrizeCode()
        {
            return PrizeCode;
        }

        public void SetPrizeCode(string code)
        {
            PrizeCode = code;
        }
    }
}
```
Step 9.5  Build your solution by clicking **Build → Build Solution**

We cannot run or debug this class library until we use it in a later exercise.

Exercise 9 Complete!

Rotate your dice to 3 to show your progress.
10 DPWS: Hello World Client

Overview:

The .NET Micro Framework v4.1 SDK comes with some very useful samples. These samples are located in C:\Users\XXXX\Documents\Microsoft .NET Micro Framework 4.1\Samples, including:

- HelloWorldClient_MF
- HelloWorldClient_WCF
- HelloWorldServer_MF
- HelloWorldServer_WCF

These samples are designed to be paired – with HelloWorldClient_xx working with HelloWorldServer_xx.

The XXX_MF samples are versions that run on the .NET Micro Framework. The XXX_WCF samples are versions that run on the desktop.

In this exercise, we will use the device as a server and the desktop as a client.

Procedural Steps

Step 10.1 Run two instances of Visual Studio 2010 on your desktop

Step 10.2 In one instance of Visual Studio 2010, select File → Open → Project/Solution… and open the project located at C:\Users\XXXX\Documents\Microsoft .NET Micro Framework 4.1\Samples\HelloWorldServer_MF

Step 10.3 In Solution Explorer, right-click HelloWorldServer_MF. Select Properties to view the Application properties.

Step 10.4 Select .NET Micro Framework.

Step 10.5 Change the Transport to Serial

Step 10.6 The Device will be set to COM1.

Step 10.7 Build your solution by clicking Build → Build Solution

Step 10.8 Start debugging by pressing F5

Step 10.9 In the other instance of Visual Studio 2010, select File → Open → Project/Solution… and open the project located at C:\Users\XXXX\Documents\Microsoft .NET Micro Framework 4.1\Samples\HelloWorldServer_WCF
Step 10.10 Build your solution by clicking **Build → Build Solution**

Step 10.11 Start debugging by pressing **F5**

Step 10.12 Set breakpoints in either solution to look further into the behavior.

Step 10.13 Try it the other way around: Use HelloWorldClient_MF and HelloWorldServer_WCF.

Exercise 10 Complete!

Rotate your dice to 4 to show your progress.
11 Simple Service

Overview:

Another sample is SimpleService. This sample uses Discovery to announce its services on the network.

Procedural Steps

Step 11.1  Close one of the instances of Visual Studio 2010 on your desktop

Step 11.2  Select File → Open → Project/Solution... and open the project located at C:\Users\XXXX\Documents\Microsoft .NET Micro Framework 4.1\Samples\SimpleService

Step 11.3  In Solution Explorer, right-click SimpleService. Select Properties to view the Application properties.

Step 11.4  Select .NET Micro Framework.

Step 11.5  Change the Transport to Serial

Step 11.6  The Device will be set to COM1.

Step 11.7  Build your solution by clicking Build → Build Solution

Step 11.8  Start debugging by pressing F5

Step 11.9  This sample uses Discovery to announce the device information to the network. On your desktop, go to Start → Computer
Step 11.10 Click on Network

Step 11.11 The “Simple Service” device will appear in the list of devices on the Network.

Step 11.12 Right click on the Simple Service to view its Properties
The displayed information is set in MFSimpleService.cs in Start()

```csharp
// Set device information
Device.ThisModel.Manufacturer = "Microsoft Corporation";
Device.ThisModel.ManufacturerUrl = "http://www.microsoft.com/";
Device.ThisModel.ModelName = "SimpleService Test Device";
Device.ThisModel.ModelNumber = "1.0";
Device.ThisModel.ModelUrl = "http://www.microsoft.com/";
Device.ThisModel.PresentationUrl = "http://www.microsoft.com/";

Device.ThisDevice.FriendlyName = "Jules Device";
Device.ThisDevice.FirmwareVersion = "alpha";
Device.ThisDevice.SerialNumber = "12345678";
```
Step 11.13 Use the class library from Exercise 9 to customize your device information.

Step 11.14 Build your solution by clicking **Build → Build Solution**

**Step 11.15** Start debugging by pressing **F5**

Exercise 11 Complete!

Rotate your dice to 5 to show your progress.
12 Putting It All Together

Overview:

Now that we have learnt about networking, services and the UI provided in the .NET Micro Framework, this section of the lab is about combining it altogether. There are no procedural steps for this one!

- Starting with the Simple Service, add UI that displays one text control, an up image and a down image
- In the text control, we want to display a number between 1 and 100
- If the up button is pressed, the number displayed increments by 1
- If the down button is pressed, the number displayed decrements by 1
- This value is sent via IntegerEvent to any SimpleServiceClient applications running.

Exercise 12 Complete!

Rotate your dice to 6 to show your progress.