Freescale Semiconductor
Application Note

Document Number: AN3887
Rev. 0, 08/2009

Development with Microsoft® .NET Micro Framework 2.0

by *Multimedia Applications Division*
Freescale Semiconductor, Inc.
Austin, TX

This document describes the Microsoft® .NET Micro Framework architecture and provides detailed instructions (using a C# GUI example) for creating a sample application that can be used as the starting point for any complex application. The .NET Micro Framework allows users to take advantage of high end programming skills to develop powerful multimedia embedded solutions at very low licensing cost.

This document is a reference for .NET Micro Framework developers using Freescale platforms. The Tahoe-II Development Board from Device Solutions uses .NET Micro Framework with the Freescale i.MXS microprocessor. In addition, .Net Micro Framework can be ported to any i.MX processor. Consult the Microsoft .Net Micro Framework web site for more information.

Contents

1 Overview

This section provides an overview of the .NET Micro Framework version 2.0 architecture, tools, applications, and resources. Some of the information in this section is applicable to other versions of .Net Micro Framework. The information in this section is adapted from the Microsoft Developers Network® (MSDN) web site for use with Freescale devices.

1.1 Architecture

The .NET Micro Framework architecture is optimized for small devices such as, but not limited to:

- Sensor networks
- Robotics
- GPS navigation
- Wearable devices
- Medical instrumentation
- Industrial automation devices
- Other small devices that require an efficient, low-resource-consuming Microsoft .NET client

The .NET Micro Framework is available for an increasing variety of hardware platforms. Its architecture is extremely flexible and highly adaptable to new hardware platforms. Figure 1 illustrates the .NET Micro Framework hardware and software architecture.

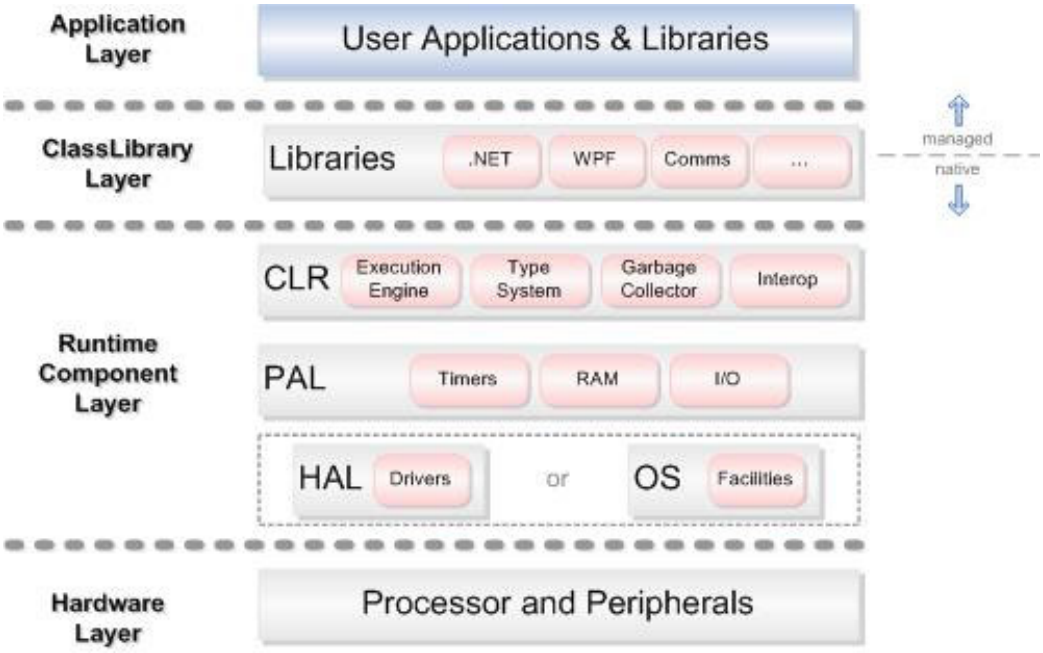

Figure 1. .NET Micro Framework Architecture

1.1.1 Hardware Layer

The hardware layer contains the user-selected microprocessor and other circuitry. The .NET Micro Framework runs on a growing number of i.MX hardware platforms. It is possible to port the .NET Micro Framework to other i.MX chipsets. It is also possible to run the .NET Micro Framework runtime on top of operating systems, such as Microsoft Windows®, but is almost exclusively used when ported directly to the hardware.

1.1.2 Runtime Component Layer (RCL)

The RCL has three components:

- .NET Micro Framework common language runtime (CLR)
- Hardware abstraction layer (HAL)
- Platform abstraction layer (PAL)

1.1.2.1 Common Language Runtime

The .NET Micro Framework CLR is a subset of the .NET Framework CLR, which is the run-time environment provided by the .NET Framework. The .NET Micro Framework CLR provides robust application support. It manages memory, thread execution, code execution, and other system services. The CLR provides all of these features and services from a very small memory footprint. It occupies only about 390 Kbytes of memory when all of the functionality is used. This memory-usage estimate is for the Microsoft implementation framework for existing i.MX platforms.

The CLR is fast and performs about 15,000 managed method calls per second at 27.6 MHz (these results are based on the average method-call complexity developed for the Smart Personal Objects Technology (SPOT) watch application). Managed applications for the CLR can be programmed and debugged using the Microsoft Visual Studio® 2005 development system.

NOTE

C# is currently the only programming language supported by the CLR.

Features of the .NET Micro Framework CLR include:

- Numeric types, class types, value types, arrays (one-dimensional only), delegates, events, references, and weak references
- Synchronization, threads, and timers
- Reflection
- Serialization
- Garbage collection
- Exception handling
- Time classes, including **DateTime** and **TimeSpan** (using INT64 arithmetic)
- Time-sliced thread management

Exceptions to and extensions of the CLR include:

- Execution constraints that limit call durations and prevent failures

- Strings represented internally as UTF-8 and exposed as Unicode
- No support for sparse or jagged multidimensional arrays
- A global, shared string table for well-known values (types, methods, and fields) that reduces RAM and ROM and reduces the number of cross-references
- No virtual tables for method resolution which saves RAM
- A **WeakDelegate** class to handle dangling references to delegates
- Support for extending the hardware chipset by programming its drivers directly in C#, using the Managed Drivers Framework. Hardware devices compatible with industry communication standards (such as GPIO, serial, SPI, or I^2C) can be added to the chipset and used by a managed C# application.

1.1.2.2 Hardware Abstraction Layer and Platform Abstraction Layer

The HAL and the PAL control the underlying system hardware. Both the HAL and the PAL are groups of C++ functions called by the CLR. The PAL functions are independent of the hardware and should not need to be ported.

NOTE

Users must develop their own version of the HAL when porting to an unsupported i.MX processor or porting to a new platform.

The bootstrap code is also associated with the HAL. The bootstrap code initializes the low-level hardware when the device is turned on and then starts the CLR to perform the higher-level initialization. The bootstrap code performs its tasks through calls to the HAL and assembly-language routines. Other than starting the CLR, the bootstrap code has no interaction with the code preceding it in the software architecture.

1.1.3 Class Library Layer

The class library included with the .NET Micro Framework is an object-oriented collection of reusable types that can be used to develop embedded applications. It includes C# libraries that provide support for the following:

- Encryption
- Debug, graphics, and shell DLLs
- The CLR API class library and the CLR corelib
- Access for managed C# applications to extended chipsets that support specific communications standards, such as GPIO, serial, SPI, or I^2C.

NOTE

The .NET Micro Framework libraries are implemented in namespaces that mirror the .NET Framework Base Class Libraries (BCL) whenever possible, otherwise they are placed in the .NET Micro Framework SPOT namespace.

1.1.4 Application Layer

The top layer of the .NET Micro Framework contains the managed applications that are created for devices. The types of applications developed depend entirely on the device hardware.

NOTE

C# is currently the only programming language supported for managed applications.

1.2 Targeted Applications

The .NET Micro Framework is targeted for very small devices. The range of functionality in these devices is typically constrained by some combination of cost, memory, processing capability, and power. The .NET Micro Framework has been used in some of the smallest devices, including watches, GPS navigation devices, and Windows® SideShow®-compatible displays.

Being targeted at the new, less expensive, and more power-efficient 32-bit processors, such as the Freescale i.MX Family, enables the .NET Micro Framework platform to run on significantly reduced resources. Additionally, it offers power management interfaces, thus maximizing the battery life of devices. All of the .NET Micro Framework functionality can be squeezed into a minimum of 256 Kbytes of Flash/ROM and executes in at least 512 Kbytes of RAM. Figure 2 illustrates the targeted Windows markets.

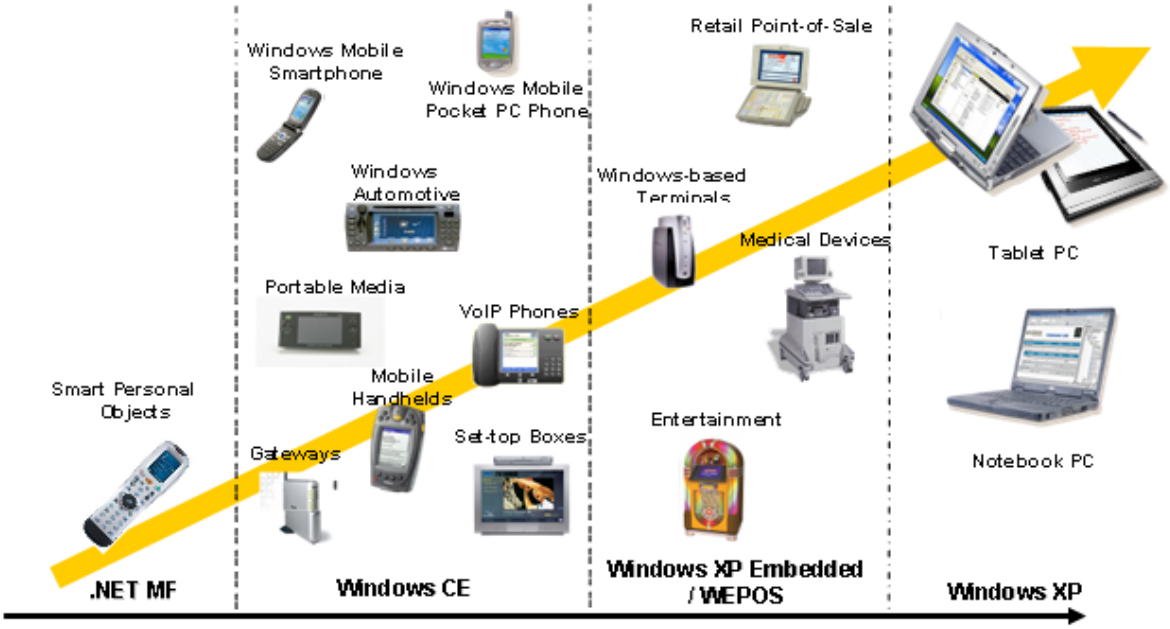

Figure 2. Targeted Markets for the Windows Operating Systems

To make the .NET Micro Framework available with the smallest possible memory footprint, it is designed to contain only those pieces of the .NET Framework that are most relevant to small devices. These include the a major subset of the BCL namespaces, among others: **System.Collections**, **System.Diagnostics**, **System.Globalization**, **System.IO**, **System.Reflection**, **System.Resources**, **System.Runtime**, and **System.Threading**.

1.3 Tools

The addition of the i.MX development tools to the .NET Micro Framework results in a powerful set of tools for developing and debugging applications. Two tools are especially useful: FlashLite Client and MFDeploy.

1.3.1 FlashLite Client

FlashLite Client captures debug output from the execution of the .NET Micro Framework on a device. While this tool also provides additional functionality, the capture function is the most frequently used. Debug output is produced by the .NET Micro Framework runtime, as well as by the execution of **Debug.Print(Strings)** statements located in the C# application. Figure 3 illustrates the FlashLite Client main screen.

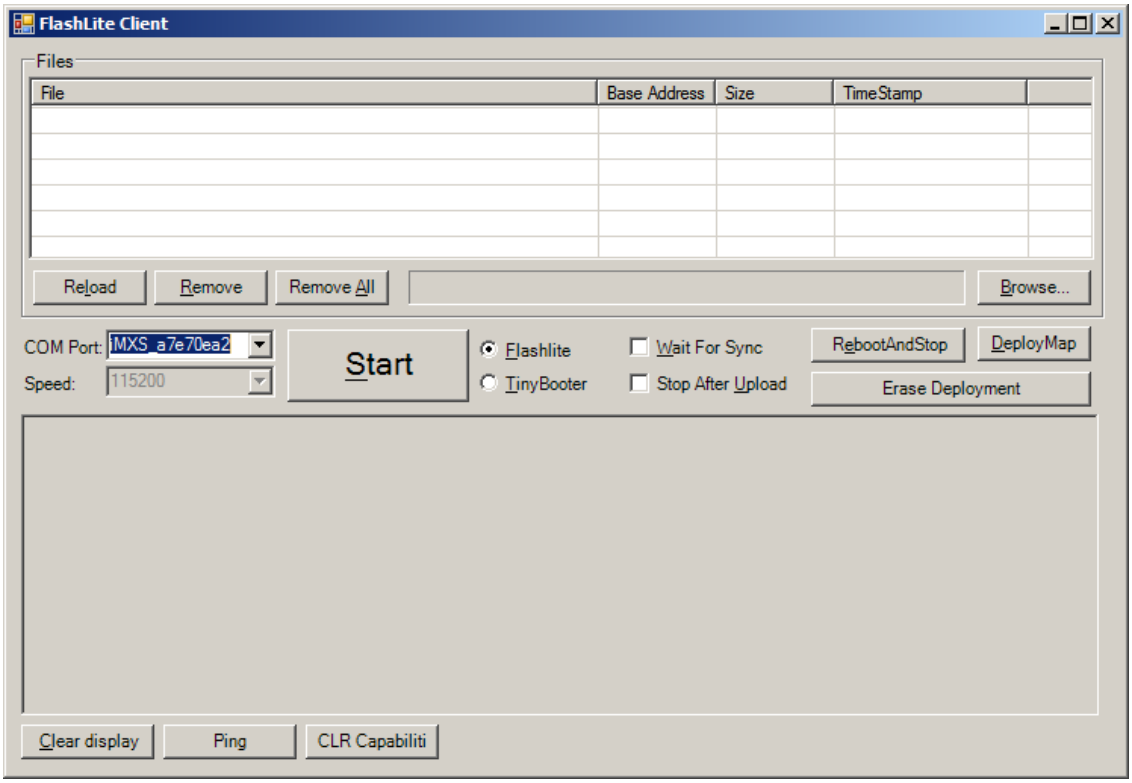

Figure 3. FlashLite Client Main Screen

1.3.2 MFDeploy

MFDeploy helps deploy, upgrade, or erase the firmware being developed for hardware. Two frequently performed tasks are described in Section 1.3.2.1, “Ping the Device,” and Section 1.3.2.2, “Clear the Deployment and Persistence Sectors.”

Figure 4 illustrates the MFDeploy screen.

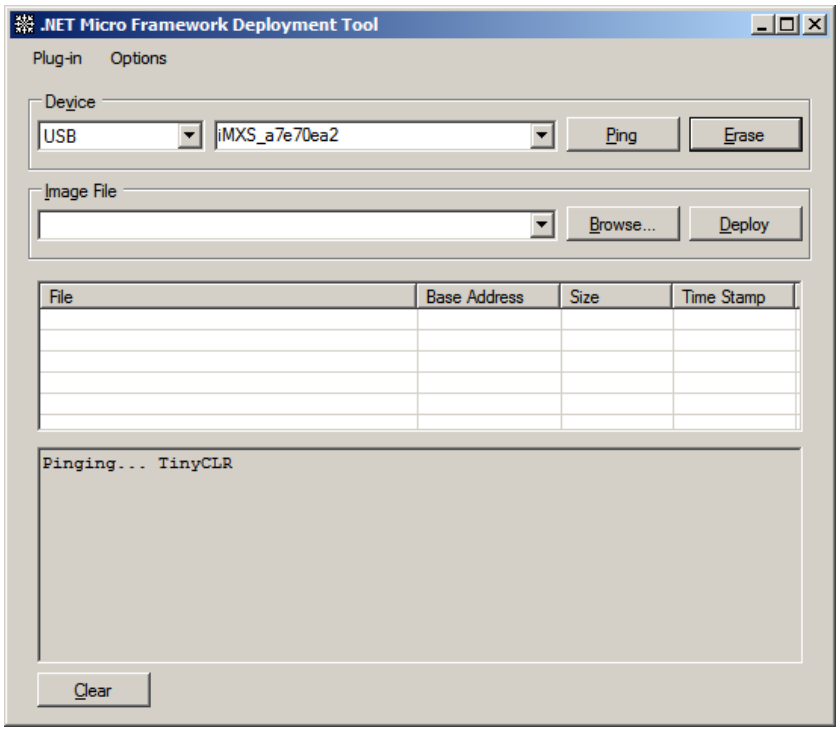

Figure 4. MFDeploy Main Screen

1.3.2.1 Ping the Device

Ping the device to confirm the correct communication between the PC and the i.MX development board through serial communication using these steps:

1. Run MFDeploy
2. Select the correct port
3. Click **Ping** to determine if the .NET Micro Framework is running and communicating through the designated port

When using a COM port, failure can indicate that another process on the computer is using that port.

1.3.2.2 Clear the Deployment and Persistence Sectors

Perform this task to deploy .NET Micro Framework application with errors using these steps:

1. Run MFDeploy
2. Select the correct port
3. Click **Erase** to clear all assemblies from the device

1.4 Help

To obtain help with .NET Micro Framework applications for Freescale devices:

- Search the Internet to review information available on this topic
- Use the .NET Framework Help program, because .NET Framework and .NET Micro Framework are very similar (.NET Framework contains additional advanced functionality)

The .NET Framework Help information searches can be configured to filter for .NET Micro Framework with C# using these steps:

1. Open Visual Studio or Visual C#
2. On the Help menu, select **Search**

   A dialog displays options for **Language**, **Technology**, and **Content Type**
3. Configure the options as follows:

   Language: C#

   Technology: .NET Micro Framework

   Content Type: All

2 Create a GUI Using C#

This section provides a sample application for the .NET Micro Framework using Visual Studio 2005 C# as the Integrated Development Environment (IDE). The sample information can be used as a reference for application development.

2.1 Hardware and Software Components

Table 1 identifies the software and hardware used in developing and testing the C# GUI example. If the software/firmware listed in Table 1 is not installed, download it from the recommended site and install it.

Table 1. Components

Component	Description	Location
i.MX Development Kit		
i.MX Development Board[1]	Hardware	i.MX Development Kit Supplier
.NET Micro Framework 2.0 flashed in the board	Firmware	CD of contents
PC		
Microsoft Visual Studio 2005	Software	Microsoft Web site
Microsoft Visual C#	Software	Microsoft Web site
Microsoft .NET Micro Framework 2.0	Software	Microsoft Web site
FlashLite Client	Software	CD of contents
MFDeploy	Software	Microsoft Web site
USB Driver for Board	Software	CD of contents

[1] For example the Tahoe-II Development Kit provided by Device Solutions.

2.2 IDE Layout and Description

The IDE used for this example is the standard Microsoft Visual Studio IDE. The IDE (Figure 5), provides a set of tools using four window panes, to help write and modify the program code as well as detect and correct program errors. Table 2 describes the IDE window panes.

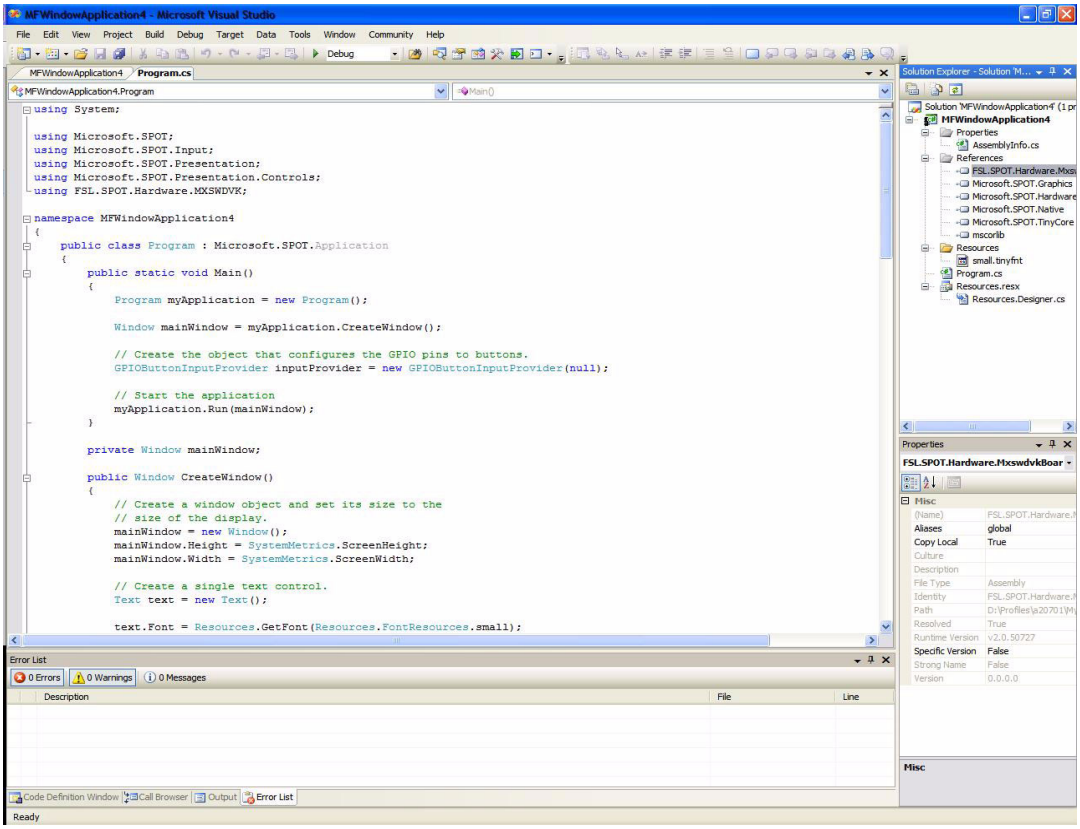

Figure 5. Visual Studio 2005 IDE

Table 2. IDE Tools

Pane	Location in Figure 5	Description
Source Code Editor	Top Left	Displays handwritten C# source code development activities. Provides helpful development features, such as syntax highlighting, class/method auto-completion (IntelliSense), and squiggly underline error indication. Multiple source files can be opened simultaneously and can be accessed from tabs above the editor.
Solution Explorer	Top Right	Lists the files in the current project. Double-clicking a source file (*.cs, *.xml …) in the tree displays the code in the Source Code Editor.
Properties	Bottom Right	Displays additional information about the selected file.
Errors/Warnings List	Bottom Left	Displays errors and warnings as the compiler runs. Visual Studio runs auto-compilation in the background as new code is typed without requiring an explicit compile.

2.3 Common IDE Tasks

Table 3 describes the commands for frequently used tasks.

Table 3. Common Tasks

Task	Command
Build Solution	Click **Build** > **Build Solution**, or press F6
Build and Deploy Solution	Click **Debug** > **Start Without Debugging**, or press CTRL+F5
Build, Deploy, and Debug Solution	Click **Debug** > **Start Debugging**, or press F5
Add New File to Solution	Right-click the project file in the Solution Explorer and click **Add** > **New Item**
Add New Resource	Double-click the Resources.resx file in the Solution Explorer to open the Resource Editor in place of the Source Code Editor. Click **Add Resource** and select an option.

3 Develop an Application

This section provides the information needed to develop a .NET Micro Framework application, including descriptions of the file contents.

3.1 Create the Solution, Project, or Application

To create a .NET Micro Framework C# solution, project, or application, use these steps:

1. Open Visual Studio or Visual C#
2. Click **File** > **New** > **Project**

   The New Project window is displayed (Figure 6).

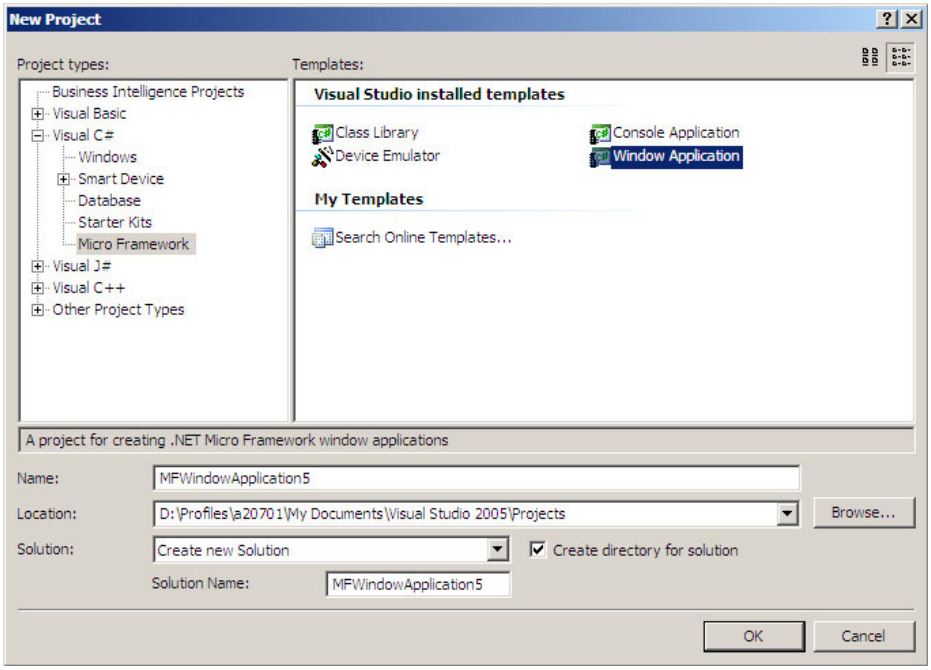

Figure 6. New Project Window

3. In the Project types window, select **Visual C# > Micro Framework**
4. In the Templates window, select **Window Application**
5. For **Name**, enter a descriptive name for the project
6. Next to **Location**, select from the drop-down list or click **Browse** to select the location where the project is saved
7. For **Solution**, select **Create New Solution**
8. Select **Create directory for solution** to create a new directory in which to place this solution
9. Click **OK**

   The Solution Explorer pane lists the files that are added to the project.

The most important file in which the program is located is `Program.cs`. This file contains the definition of the **Program** class that must be of type **Microsoft.SPOT.Application**. The **Program** class contains the **Main()** method, which is called when the application starts. Additional files provide code that is related to window creation and configuration for the button press interrupt handler.

3.2 Include the Hardware-Specific Files

The hardware-specific files contain platform-specific information that is used in the managed code to configure the serial ports, keyboard, GPIO, specific modules, and so on.

3.2.1 HardwareProvider Class

The **HardwareProvider** class provides access to information about pin assignments for serial communications devices such as I^2C, UART, and SPI. The **HardwareProvider** class is implemented in one of the hardware-specific files provided by the board supplier. The file may be edited as needed. Defining this class correctly for an application ensures that the standard serial classes of .NET Micro Framework operates with the serial ports.

3.3 Add UI Elements to the Application

The .NET Micro Framework includes a limited set of user interface (UI) elements, with limited functionality. These elements (Canvas, Text, Image, ListBox, Panel and StackPanel among others) can be used in any .NET Micro Framework application.

To add UI elements to an application, use these steps:

1. Ensure that the **Microsoft.SPOT.Presentation.Controls** namespace is referenced at the beginning of the `Program.cs` file, with the following instruction line:

   ```
   using Microsoft.SPOT.Presentation.Controls
   ```

   If the line is not included, add it.

   The **Microsoft.SPOT.Presentation.Controls** namespace provides classes, delegates, and enumerations that are used to create elements (known as controls) that enable a user to interact with an application.

NOTE

If an error is displayed after the instruction line is added and compiled, include a reference to the `Microsoft.SPOT.TinyCore.dll`, as shown in the Solution Explorer pane of the IDE, Figure 5.

2. Write code into the **CreateWindow()** method, which is created by default. Example 1 displays the default code.

Example 1. Default Content of the CreateWindow() Method

```
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;

       // Connect the button handler to all of the buttons.
       mainWindow.AddHandler(Buttons.ButtonUpEvent,new ButtonEventHandler(OnButtonUp),false);
       // Set the window visibility to visible.
       mainWindow.Visibility = Visibility.Visible;

       // Attach the button focus to the window.
       Buttons.Focus(mainWindow);

       return mainWindow;
}
```

3. To modify this code, create the **StackPanel** element and assign to it the **text** element.

   Add the following lines before the `mainWindow.Child = text;` line:

```
StackPanel panel = new StackPanel(Orientation.Vertical);
panel.Children.Add(text);
```

4. Assign the **panel** element as the **Child** element of the **mainWindow**, instead of the **text** element. Example 2 displays the updated **CreateWindow()** method.

Example 2. Updated Content of the CreateWindow() Method

```
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;

        // Create the StackPanel control.
        StackPanel panel = new StackPanel(Orientation.Vertical);

        // Add the text to the StackPanel.
        panel.Children.Add(text);

        // Add the StackPanel control to the window.
        mainWindow.Child = panel;

        // Connect the button handler to all of the buttons.
        mainWindow.AddHandler(Buttons.ButtonUpEvent,new ButtonEventHandler(OnButtonUp),false);

        // Set the window visibility to visible.
        mainWindow.Visibility = Visibility.Visible;

        // Attach the button focus to the window.
        Buttons.Focus(mainWindow);
        return mainWindow;
}
```

3.3.1 Add Keyboard Events to the Application

This section describes how to add keyboard interaction to the application. When a key is pressed, the name of the key is printed through a text element in the StackPanel. To add keyboard management, first ensure that the hardware-specific namespace is referenced at the beginning of the `Program.cs` file. In this example, the instruction line is:

```
FSL.SPOT.Hardware.BOARD;
```

NOTE

If an error is displayed after this line is added and compiled, see Section 3.2, "Include the Hardware-Specific Files."

Write code into the `Program.cs` file, which is created by default:

1. Delete the `GPIOButtonInputProvider.cs` file from the Solution Explorer pane of the IDE. This example uses the **GPIOButtonInputProvider** class from the hardware-specific files.
2. Add code to the **OnButtonUp()** method, which is created by default. The **OnButtonUp()** method adds keyboard events to the application. The goal is to identify the pressed button, then assign custom text to a variable that is assigned to the StackPanel at the end of the function. The updated method is shown in Example 3.

Example 3. Updated Content of the OnButtonUp() Method

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

     // Define the temporalText variable with the name of the button pressed.
     Text temporalText = new Text("Button");
     temporalText.Font = Resources.GetFont(Resources.FontResources.small);
     temporalText.HorizontalAlignment=Microsoft.SPOT.Presentation.HorizontalAlignment.Center;
     temporalText.VerticalAlignment=Microsoft.SPOT.Presentation.VerticalAlignment.Center;
     switch(e.Button){
            case Button.Select:
                  temporalText.TextContent += " Center";
                  break;
            case Button.Down:
                  temporalText.TextContent += " Down";
                  break;
            case Button.Up:
                  temporalText.TextContent += " Up";
                  break;
            case Button.Left:
                  temporalText.TextContent += " Left";
                  break;
            case Button.Right:
                  temporalText.TextContent += " Right";
                  break;
            default:
                  temporalText.TextContent += " Not identified";
                  break;
     }
     // Add the temporalText to the StackPanel defined for the mainWindow.Child
     ((StackPanel)mainWindow.Child).Children.Add(temporalText);
}
```

Now, each time the **OnButtonUp()** method executes due to a key press, the program updates the content of the **StackPanel** in the window with the name of the key that is pressed.

3.3.2 Deploy and Run the Application

The last step is to deploy and run the application. First, the board must be connected to the PC using a USB cable. Next, configure the .NET Micro Framework application to work with the board using these steps:

1. Click **Project > [Application Name] Properties...**
   The Source Code Editor is displayed (Figure 7).

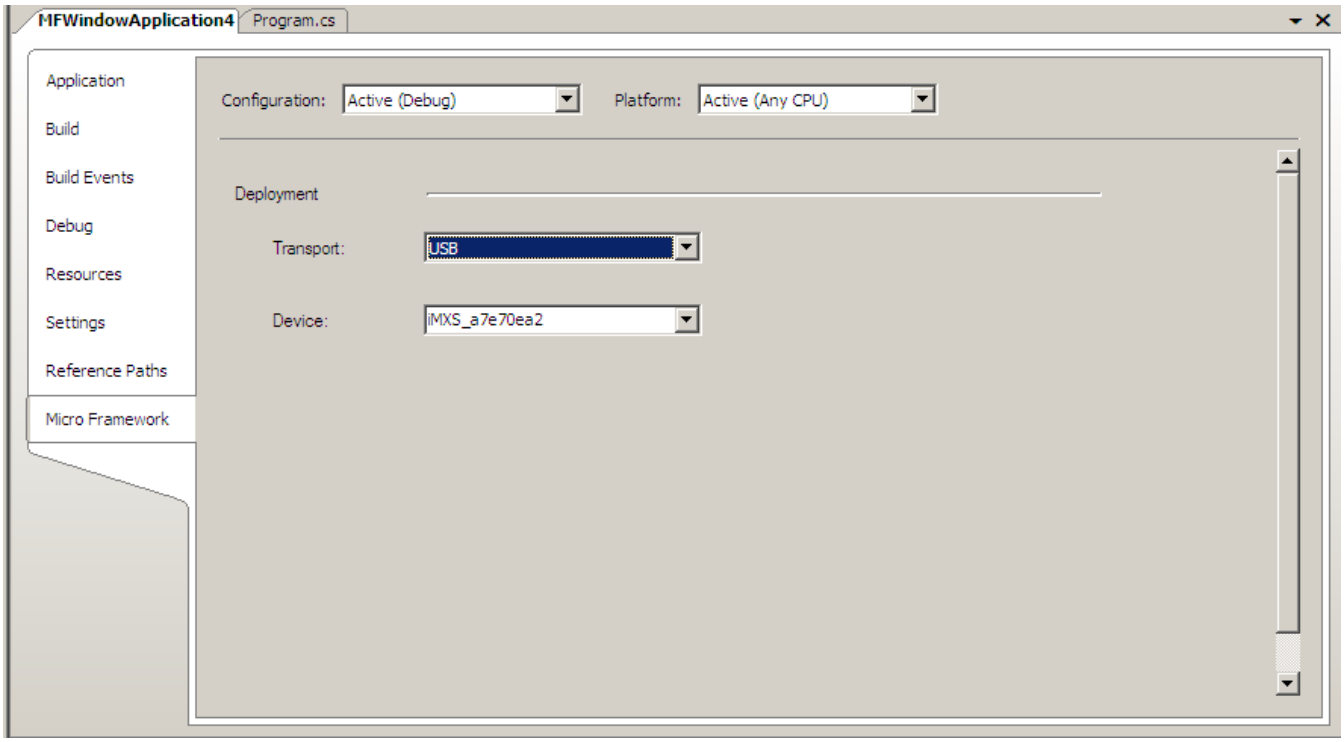

Figure 7. Micro Framework Deployment Configuration

2. Select the Micro Framework tab in the navigation panel
3. For **Configuration**, select the type of configuration to use
4. For **Transport**, select **USB**
5. For **Device**, accept the displayed Device number, which is the board identification

The application is ready to deploy.

4 Deployment Errors

This section describes the most frequently encountered issues related to developing and deploying .NET Micro Framework applications.

4.1 USB Port is Busy with Other Application

In some cases, the development boards for .NET Micro Framework have only one USB device port. To support debug logging and application deployment, the USB port is multiplexed between FlashLite Client (Debug Logging) and Visual Studio 2005 (Application Deployment).

When FlashLite Client is in **Start** mode, the USB port is not available to Visual Studio. If deployment problems in Visual Studio are encountered, ensure that the FlashLite Client is in **Stop** mode.

4.2 USB Driver is Incorrectly Installed

This error message indicates that the USB driver for the board is not correctly installed and the driver must be reinstalled. The hardware must be correctly installed and ready to use.

To reinstall the USB driver, use these steps:

1. Click **Start**, point to **Run**, and enter **devmgmt.msc**, to run the Device Manager
2. Find and uninstall the element **Micro Framework MXS Reference Design** or similar term
3. Run the scan, by clicking **Menu Action** > **Scan for hardware changes**
4. In the **Found New Hardware Wizard**, select the USB board driver recommended by the i.MX development board manufacturer. Typically, the board driver is located on the CD that is included with the development kit.

4.3 USB Communication is Broken

The following failures indicate that the USB communication is not working:

- The device seems connected, but Visual Studio cannot deploy new applications
- FlashLite Client does not display debug information and it cannot ping the board

To restart the USB driver, use these steps:

1. Ensure that FlashLite Client is in **Stop** mode
2. Unplug the mini-USB connector from the board and plug it back in
3. Restart the board

4.4 Application is Deployed with Errors

In Visual Studio 2005, the last deployment is interrupted or is completed with errors and any subsequent try at re-deployment fails.

To clear the deployment and persistence sectors, use these steps:

1. Ensure that FlashLite Client is in **Stop** mode
2. Follow the steps in Section 1.3.2.2, “Clear the Deployment and Persistence Sectors”
3. Try to deploy the application again

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
1-800-521-6274 or
+1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku
Tokyo 153-0064
Japan
0120 191014 or
+81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor
Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800 441-2447 or
+1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor
@hibbertgroup.com

Document Number: AN3887
Rev. 0
08/2009