Table of Contents

Table of Contents

Table of Contents

1 German About the Book

Beginners Guide to
C#
and
.NET Micro Framework

August 3rd 2010
Rev 1.03

Copyright © 2010 GHI Electronics, LLC
www.GHIElectronics.com
www.TinyCLR.com
By: Gus Issa

E-Book

1.1 Change Log

Bitte haltet das Buch auf dem neuesten Stand.

1.1.1 Rev 1.03 1/4/2011

- Audio Playback in C-Sharp Level2 Kapitel

1.1.2 Rev 1.02 1/1/2011

- "wichtiger Hinweis" vom USB Client Kapitel entfernt. Die letzte firmware unterstützt debugging + virtual serial CDC port gleichzeitig

1.1.3 Rev 1.01

- Code zum Erkennen von SD Karten hinzugefügt

1.1.4 Rev 1.00

- Erste Version übersetzt durch RobotFreak

1.2 Wiki? Was ist ein Wiki?

Dieses Buch wird automatisch aus dem NETMF Wiki generiert. Dies erlaubt es jedem, Änderungen an dem Buch zu machen und anderen beim Übersetzen des Buches zu helfen.

Für weitere Details, siehe http://www.microframeworkprojects.com/index.php?title=Category:Beginner_Guide_ebook

1.3 Zielguppe

Dieses Buch ist für Anfänger bestimmt, die .NET Micro Framework (NETMF) erlernen möchten. Es sind keine Vorkenntnisse erforderlich. Das Buch deckt die Bereiche .NET Micro Framework, Visual C # und sogar C # ab! Wenn Sie Programmierer, programmieren als Hobby oder ein Ingenieur sind, werden sie eine ganze Menge Infos in diesem Buch finden. Dieses Buch setzt keine Vorkenntnisse voraus, es ist alles ausführlich erklärt.

Die Autoren haben ihre persönliche Freizeit genutzt (wenn das überhaupt möglich ist!), um dieses Buch zu machen. Wenn sie Tippfehler oder Grammatikfehler finden, dann melden Sie diese bitte im Forum, damit helfen sie dieses Buch weiter zu verbessern.

1.4 Das Buch übersetzen

Dieses Buch ist für die Community gedacht um NETMF einfacher für alle Benutzer zu machen. Wenn Sie denken, Sie können das Buch auch in andere Sprachen übersetzen, dann würden wir dies sehr gerne sehen.

Die englische Originalversion des Buches befindet sich unter http://www.microframeworkprojects.com/index.php?title=Category:Beginner_Guide_ebook Man kann einfach einzelne Kapitel mit copy/paste übernehmen und mit der Übersetzung beginnen. Bitte beachten Sie, dass alle Code Beispiele und Bilder beibehalten werden. Diese müssen nicht übersetzt werden.

Hier sind die einfachen Schritte:

1. Erstelle eine neue Wiki Seite mit dem selben Namen der Original Seite und setze den Namen der Sprache davor. Ein Beispiel: ändere den Link von http://www.microframeworkprojects.com/index.php?title=USB_Host zu http://www.microframeworkprojects.com/index.php?title=French_USB_Host
2. Öffne beide Seiten und copy/paste alles von der englischen zur französischen Seite.
3. **Wichtig:** Ändere die Kategorie am Anfang jeder Seite von Beginner_Guide_ebook nach French_Beginner_Guide_ebook
4. Beginne mit der Übersetzung des Textes
5. Füge weitere Kapitel hinzu bis zum Ende

Bitte fange damit an, auch wenn du nicht das ganze Buch schaffen wirst. Andere werden dir helfen, das Buch fertigzustellen.

2 Einführung

Hatten Sie jemals eine großartige Idee für ein Produkt gehabt, konnten diese aber nicht verwirklichen, weil die Technik nicht auf Ihrer Seite war? Oder dachten sie vielleicht: "Es muss doch einen einfacheren Weg geben!" Vielleicht sind sie ein Programmierer, der ein Sicherheitssystem entwerfen wollte, aber dann dachten mit PCs wird das zu teuer, um damit ein einfaches System zu bauen? Dann ist die Antwort Microsoft .NET Micro Framework!

Hier ist ein Szenario: Sie möchten einen Pocket-GPS-Datenlogger bauen, der Position, Beschleunigung und Temperatur auf einer Speicherkarte speichert und auf einem kleinen Display anzeigen soll. GPS-Geräte können Positionsdaten über eine serielle Schnittstelle senden. Man kann nun einfach ein Programm schreiben, dass auf einem PC läuft und die GPS-Daten lesen und in eine Datei schreiben kann. Aber der PC passt nicht in Ihre Tasche! Ein weiteres Problem ist, wie würden Sie mit einem PC Temperatur und Beschleunigung messen wollen? Wenn Sie dieses Projekt mit einem klassischen Mikrocontroller, wie z.B. einem AVR, oder PIC Mikrocontroller machen wollen, könnte all dies getan werden, aber dann brauchen Sie einen Compiler (wahrscheinlich nicht kostenlos), bis zu einer Woche Einarbeitungszeit für den Prozessor, eine Woche brauchen Sie zum Schreiben eines seriellen Treibers, einen Monat oder länger, um herauszufinden, wie man das FAT-Dateisystem benutzt und noch mehr Zeit für Speicherkarten etc. ... Grundsätzlich kann das in vielen Wochen Arbeit getan werden.

Eine weitere Option wäre die Verwendung einfacherer Methoden (BASIC STAMP, PICAXE, Arduino, etc). All diese Produkte vereinfachen ein Design aber jedes hat seine Grenzen. Nur ein paar von ihnen haben Debugging-Funktionen. Letztendlich sind diese Geräte in der Regel nicht für die Massenproduktion geeignet.

2.1 Vorteile

Wenn man das .NET Micro Framework benutzt ergeben sich viele Vorteile:

1. Es läuft unter der kostenlosen High End IDE Microsoft's Visual C# Express.
2. .NET Micro Framework ist Open Source und kostenlos.
3. Jedes Programm wird ohne grosse Änderungen auf allen Geräten laufen.
4. Volle Debugging Möglichkeiten. (Breakpoints, Progamm in Einzelschritten testen, Variablen anschauen...etc.)
5. Wurde in vielen kommerziellen Produkten getestet. Damit ist Qualität gewährleistet.
6. Beinhaltet viele Treiber.(SPI, UART , I2C...etc.)
7. Man benötigt keine Prozessor Datenblätter wegen des standardisierten Framework
8. Wenn sie bereits Erfahrung mit PC C# Programmen haben dann sind sie bereits ein Embedded Systementwickler mit NETMF!

3 Eigene Portierung gegenüber GHI's Angebot

Es gibt zwei Arten der Arbeit mit NETMF, eigene Portierung oder es einfach nur zu benutzen. Zum Beispiel ist das Schreiben eines Java-Spiels auf einem Handy viel einfacher als die Portierung der Java Virtual Machine (JVM) auf einem Handy. Der Handy Hersteller tat bereits die Arbeit der Portierung von Java auf das Handy und Spiele-Programmierer können sie mit weniger Aufwand nutzen. NETMF funktioniert auf dieselbe Weise, die Portierung ist nicht einfach, aber es ist sehr einfach eine portierte Umgebung zu nutzen. Bei Verwendung eines GHI NETMF Produktes, erhalten sie nicht nur ein Gerät mit NETMF sondern sie bekommen unbegrenzte Möglichkeiten, Support, Wartung, Robustheit und Time-to-Market. Schauen wir uns dieses im Detail an.

3.1 Features

GHI NETMF Produkte haben viele exklusive Features wie USB-Host, USB-Device, OneWire, CAN, PPP, WiFi, zu viele um sie hier alle aufzulisten. All dies ist ohne zusätzliche Kosten enthalten. GHI fügt ständig neue exklusive Features über Updates kostenlos hinzu!

3.2 Support

Unser erstklassiger Support ist kostenlos. Die selben Ingenieure, die diese Geräte entwickelt haben sind auch für die Überwachung der Foren, Emails und Telefone zuständig. Wir sind hier um Ihnen auf jedem Schritt des Weges zu helfen, damit Ihr Produkt so schnell wie möglich auf dem Markt erscheinen kann. Wir würden uns freuen wenn Sie unser Forum besuchen oder andere Kunden fragen wie zufrieden sie mit GHI sind.

3.3 Wartung

Alle paar Monate veröffentlicht Microsoft eine neue Version von NETMF. GHI arbeitet sehr eng mit Microsoft zusammen, um mögliche Probleme zu lösen und macht die ganze Arbeit, um alle GHI NETMF Geräte zu aktualisieren. Für GHI Kunden bedeutet dies ein fünfminütiges kostenloses Firmware-Update und GHI kümmert sich um den Rest.

3.4 Robustheit

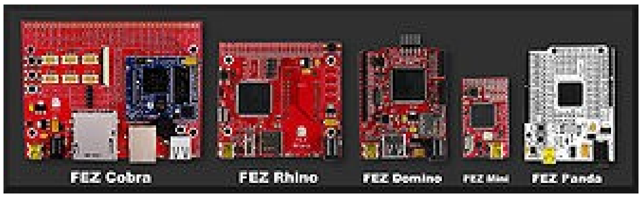
Es gibt Tausende von NETMF GHI Geräte die rund um die Welt in den meisten Märkten verwendet werden. Diese riesige Nutzung garantiert Qualität und Stabilität der GHI Electronics-Produkte. Sie können jedes der GHI-Electronics-Produkte mit Leichtigkeit nutzen.

3.5 Time-to-Market

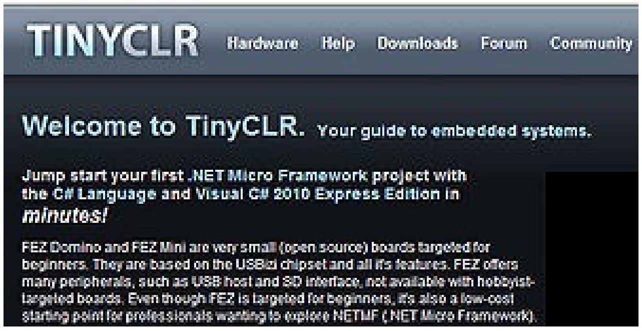
Mit GHI Electronics NETMF Produkten beschleunigen Sie die Entwicklung. Ihr Design ist fast fertig, sobald Sie eines der GHI NETMF Produkte hinzuzufügen. Wir haben Kunden, die vollständige Produkte in einer Woche geschafft haben! Sie können zum Beispiel mit dem FEZ Rhino Starter-Kit Programe innerhalb weniger Tage schreiben. Sie fügen Ihren Firmenlogo-Aufkleber hinzu und haben Ihr eigenes Produkt. Sie werden wahrscheinlich den größten Teil Ihrer Zeit mit dem Entwerfen / Bestellen der Logo-Aufkleber verbringen als mit dem Hardware-Design!

4 Auswählen eines Gerätes

GHI bietet eine große Auswahl an Geräten.



Für kommerzielle Module und Entwicklungs Syteme besuchen Sie bitte http://www.ghielectronics.com

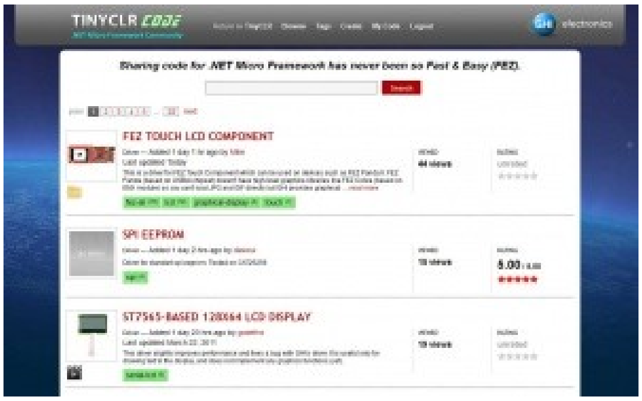


Welcome to TinyCLR. Your guide to embedded systems.

Jump start your first .NET Micro Framework project with the C# Language and Visual C# 2010 Express Edition in *minutes!*

FEZ Domino and FEZ Mini are very small (open source) boards targeted for beginners. They are based on the USBizi chipset and all it's features. FEZ offers many peripherals, such as USB host and SD interface, not available with hobbyist-targeted boards. Even though FEZ is targeted for beginners, it's also a low-cost starting point for professionals wanting to explore NETMF (.NET Micro Framework).

GHI's grosse NETMF Community Webseite http://www.TinyCLR.com enthält eine Menge OEM und Anfängerfreundliche Geräte.



Sharing code for .NET Micro Framework has never been so Fast & Easy (FEZ).

FEZ TOUCH LCD COMPONENT

SPI EEPROM

ST7565-BASED 128X64 LCD DISPLAY

A pro pos, nicht zu vergessen http://code.tinyclr.com mit hunderten von Treibern und Programm Beispielen.

5 German Getting Started

Wichtiger Hinweis: Wenn Sie Ihre Hardware gerade erst erhalten haben oder Sie nicht sicher sind, welche Firmware sich darauf befindet, MÜSSEN Sie die Firmware aktualisieren. Die Dokumentation (Handbuch oder Tutorial) des Gerätes zeigt Ihnen, wie Sie die Firmware aktualisieren. Dies ist ein erforderlicher Schritt. Also, stellen Sie sicher, das Sie die Firmware-/ Montage Anleitung in diesem Buch gelesen haben.

5.1 System Setup

Bevor wir irgendetwas ausprobieren, muß sichergestellt sein, das die notwendige Software auf dem PC installiert ist.

- Downloaden und installieren Sie zunächst Visual C# express 2010 (VS2008 oder älter wird nicht unterstützt) http://www.microsoft.com/express/vcsharp/. Es wird empfohlen, die englische Version zu installieren.
- Danach downloaden und installieren Sie das .NET Micro Framework 4.1 SDK (nicht das porting kit).

http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=cff5a7b7-c21c-4127-ac65-5516384da3a0

Falls der obige Link nicht funktioniert, suchen Sie nach ?.NET Micro Framework 4.1 SDK?

- Zuletzt, installieren sie das GHI NETMF SDK. Sie finden das SDK auf:

http://www.tinyclr.com/dl/

5.2 Der Emulator

NETMF beinhaltet einen Emulator, mit dem man ein NETMF Programm direkt auf dem PC ausführen kann. Für unser erstes Projekt werden wir den Emulator dazu benutzen, ein einfaches Programm laufen zu lassen.

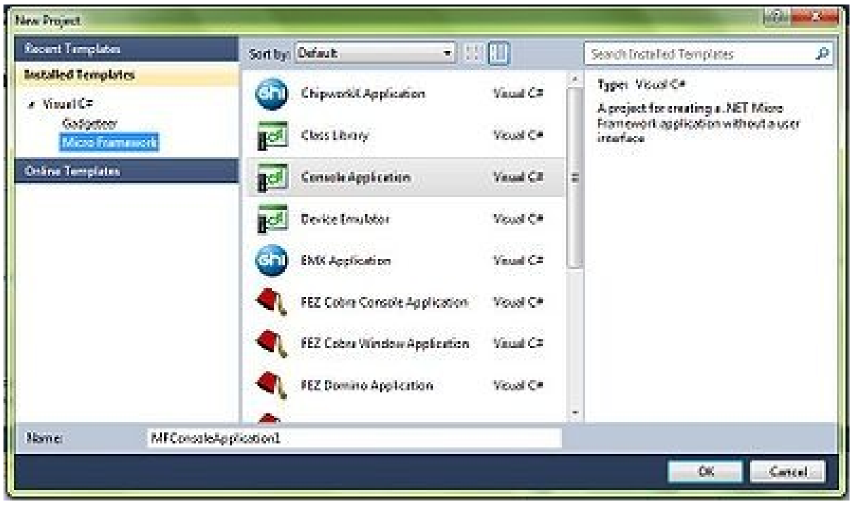
5.3 Experten Einstellung

Es gibt ein paar Einstellungen, auf die wir unter Visual C# Express 2010 zugreifen wollen, die per Default Einstellung versteckt sind. Um alle diese Features freizuschalten, muß man die "Expert Settings" einschalten.

```
Tools-> Settings->Expert Settings
```

5.3.1 Erzeugen eines Projekts

Öffnen Sie Visual C# Express und wählen aus dem Menü, select file -> New Project. Der Projekt Wizard sollte nun eine Option ?Micro Framework? im linken Menü zeigen. Klicken Sie darauf und wählen von den verfügbaren Templates ?Console Application? aus.


New Project
Recent Templates
Installed Templates
Visual C#
Gadgeteer
Micro Framework
Online Templates
Sort by: Default
Search Installed Templates
Chipworkx Application Visual C#
Class Library Visual C#
Console Application Visual C#
Device Emulator Visual C#
EMX Application Visual C#
FEZ Cobra Console Application Visual C#
FEZ Cobra Window Application Visual C#
FEZ Domino Application Visual C#
OK
Cancel

Type: Visual C#

A project for creating a .NET Micro Framework application without a user interface

Name: MFConsoleApplication1

Wenn Sie nun auf den ?OK? Button klicken, wird ein neues Projekt erzeugt, das sofort ausführbar ist. Das Projekt enthält nur eine C# Datei, mit dem Namen Program.cs, die nur wenige Code Zeilen enthält. Die Datei wird im ?Solution Explorer? Fenster angezeigt. Wenn dieses Fenster nicht angezeigt wird, können Sie es durch einen Klick auf den Menü Punkt ?View->Solution Explorer? öffnen.

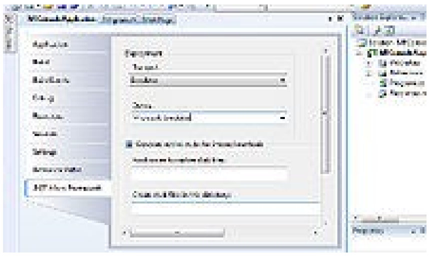
```
using System;
using Microsoft.SPOT;
namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            Debug.Print(
            Resources.GetString(Resources.StringResources.String1));
        }
    }
}
```

der Einfachheit halber ändern Sie den Code wie es das folgende Listing zeigt.

```
using System;
using Microsoft.SPOT;
namespace MFConsoleApplication1
{
  public class Program
  {
    public static void Main()
    {
      Debug.Print("Amazing!");
    }
  }
}
```

5.3.2 Ziel auswählen

Keine Sorge, wenn Sie den Code noch nicht verstehen. Dazu kommen wir später. Denn zunächst wollen wir das Programm im Emulator laufen lassen. Zunächst sollten wir sicherstellen, dass alles korrekt eingerichtet ist. Klicken Sie dazu im Menü auf "Project-> Properties". In dem neuen Fenster wählen Sie den Emulator aus. Auf der linken Seite wählen Sie ". NET Micro Framework". Nun sollte das Fenster so aussehen wie auf dem folgenden Bild. Ziel: Emulator

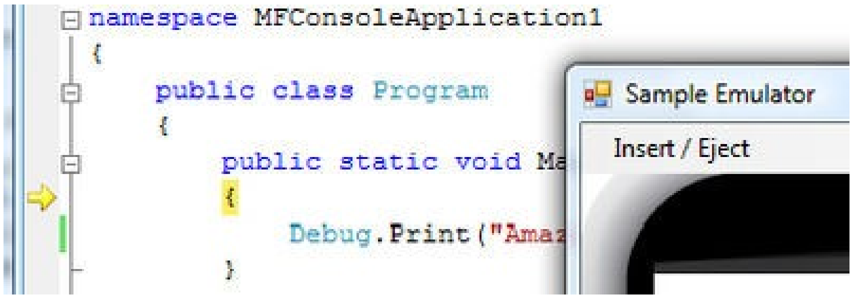


Device: Microsoft Emulator Stellen sie sicher, das das Ausgabe Fenster sichtbar ist, indem Sie auf ?View->Output? klicken.

5.3.3 Ausführen

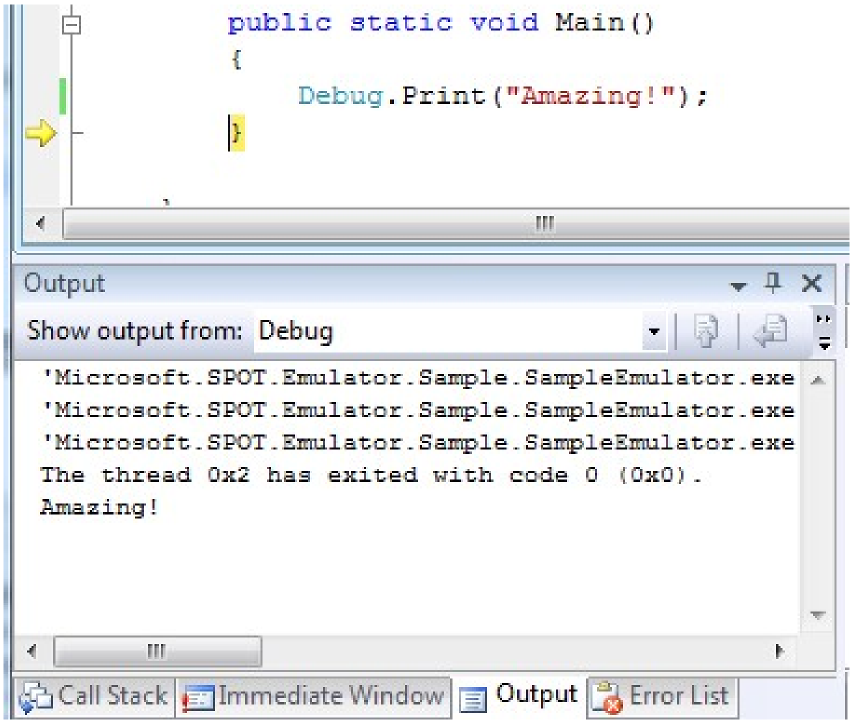
Jetzt sind wir bereit unsere erste Applikation zu starten. Drücken Sie die F5 Taste auf der Tastatur. Dies ist ein sehr nützlicher Tatatur Shortct, den wir sehr oft benutzen werden um Applikationen zu starten. Nachdem Sie F5 gedrückt haben, wird die Applikation übersetzt und in den Emulator geladen. Nach ein paar Sekunden wird die Applikation beendet, so dass nicht viel zu sehen ist.

Wir wollen den Code nun "debuggen". Debugging bedeutet, dass Sie in der Lage sind, den Code Schritt für Schritt auszuführen und sehen zu können was er tut. Dies ist einer der größten Vorteile von NETMF. Diesmal verwenden wir F11 anstatt F5, dadurch wird die Anwendung im "Einzelschritt" ausgeführt und nicht komplett durchlaufen. Damit wird die Anwendung auf dem Emulator ausgeführt und stoppt an der ersten Zeile des Codes. Dies wird durch den gelben Pfeil gekennzeichnet.



```
namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Ma
        {
            Debug.Print("Amaz
        }
```

C # Applikationen starten immer mit einer Methode namens Main und das ist dort, wo der Pfeil nach Starten des Debuggers steht. Drücken Sie erneut F11 und der Debugger durchläuft die nächste Codezeile, es ist die Zeile, die Sie zuvor geändert haben. Wahrscheinlich haben Sie es schon erraten, diese Zeile gibt "Amazing!", auf dem Debug-Fenster aus. Das Debug-Fenster ist das Ausgang Fenster von Visual C# Express. Stellen Sie sicher, dass das Output Fenster sichtbar ist, wie bereits erläutert, und drücken Sie F11 noch einmal. Sobald Sie auf diese Zeile kommen, sehen Sie das Wort "Amazing!" im Ausgabefenster.

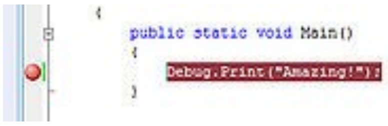

```
public static void Main()
{
    Debug.Print("Amazing!");
}
```

```
'Microsoft.SPOT.Emulator.Sample.SampleEmulator.exe
'Microsoft.SPOT.Emulator.Sample.SampleEmulator.exe
'Microsoft.SPOT.Emulator.Sample.SampleEmulator.exe
The thread 0x2 has exited with code 0 (0x0).
Amazing!
```

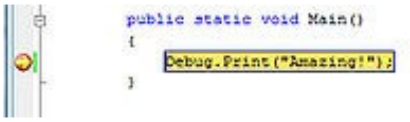
Wenn Sie jetzt wieder F11 drücken, wird das Programm beendet und auch der Emulator wird beendet.

5.3.4 Breakpoints

Breakpoints sind ein anderes sehr nützliches Feature zum debuggen von Programm Code. Während die Applikation läuft, checked der Debugger ob ein Breakpoint erreicht wurde. Wenn ja, wird die Programm Ausführung angehalten. Klicken Sie auf den Bereich links von der Zeile in der "Amazing!" ausgegeben wird. Es erscheint ein roter Punkt. Das ist der Breakpoint.


```
{
    public static void Main()
    {
        Debug.Print("Amazing!");
    }
```

Wenn Sie jetzt F5 drücken, wird das Programm gestartet und die Ausführung wird angehalten wenn der Breakpoint erreicht wird, wie im folgenden Bild zu sehen ist.


```
public static void Main()
{
    Debug.Print("Amazing!");
}
```

Nun können Sie weiter durch den Code mit F11 steppen oder das Programm mit F5 weiter ausführen.

5.4 Ausführen auf der Hardware

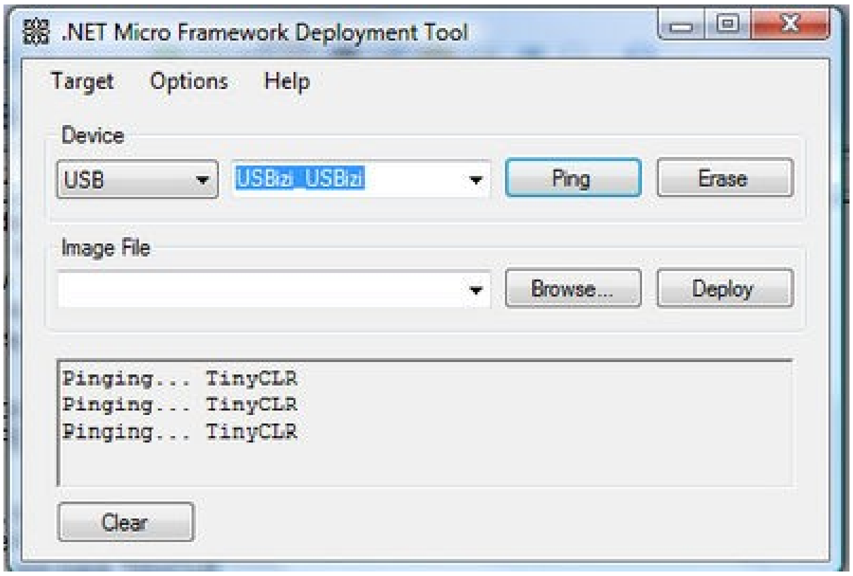
Ausführen von NETMF Anwendungen auf der Hardware ist sehr einfach. Die Anleitung funktioniert auf jeder anderen Hardware. Dieses Buch verwendet FEZ zu Demonstrationszwecken wird aber auf jeder anderen Hardware ähnlich funktionieren.

5.4.1 Ping mit MFDeploy

Bevor wir die Hardware verwenden, stellen Sie sicher, dass alles richtig angeschlossen ist. Das NETMF SDK enthält eine Software von Microsoft, genannt MFDeploy. Es gibt viele gute Verwendungsmöglichkeiten für MFDeploy aber im Moment brauchen wir nur das "Ping" zum Gerät. Grundsätzlich ist mit "Ping" gemeint, das MFDeploy "Hallo" zu einem Gerät sagt und dann prüft, ob das Gerät mit "Hallo" reagiert. Damit läßt sich einfach testen, ob das Gerät richtig angeschlossen und die Kommunikation mit ihm keine Probleme macht.

Starten Sie MFDeploy und schließen das FEZ Board über das mitgelieferte USB-Kabel an Ihrem PC an. Wenn Sie dies zum ersten Mal tun, wird Windows Sie nach einem Treiber fragen. Verwenden Sie den Treiber aus dem SDK-Ordner und warten Sie, bis Windows die Treiber Installation beendet hat.

Im Dropdown-Menü finden Sie die Option USB. Sie sollten USBizi in der Geräteliste finden. USBizi, weil FEZ auf dem USBizi Chipsatz basiert. Wählen Sie USBizi und klicken Sie auf die "Ping"-Taste. Sie sollten nun die Meldung 'TinyCLR' sehen.

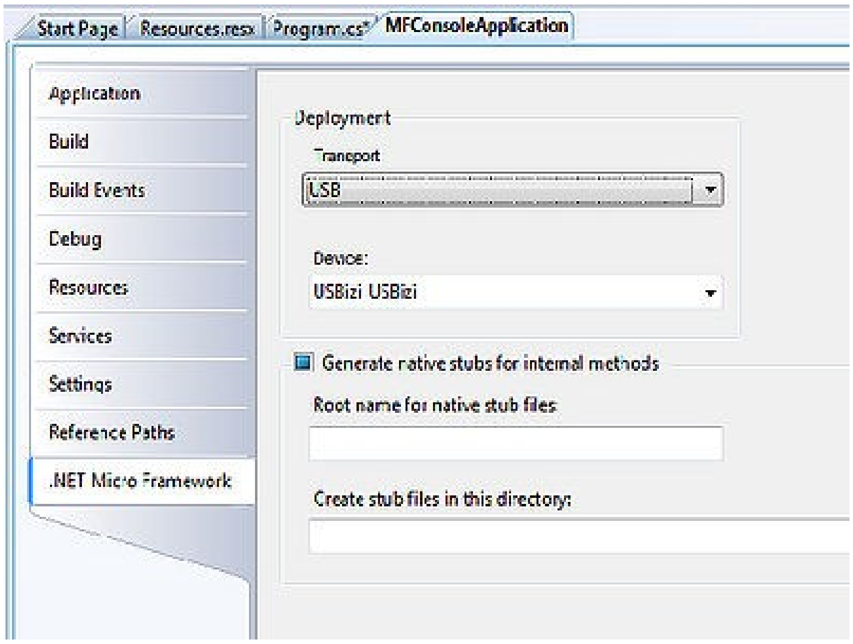


Device: USB — USBizi_USBizi

```
Pinging... TinyCLR
Pinging... TinyCLR
Pinging... TinyCLR
```

5.4.2 Bereitstellen auf die Hardware

Nachdem wir mit MFDeploy geprüft haben, ob die Hardware korrekt angeschlossen ist, gehen wir zurück zu Visual C# Express. Unter 'Project | Properties', wählen Sie USB als Schnittstelle und USBizi für das Gerät. Achten Sie darauf, dass Ihre Einstellungen dem folgenden Bild entsprechen.



Transport: USB

Device: USBizi USBizi

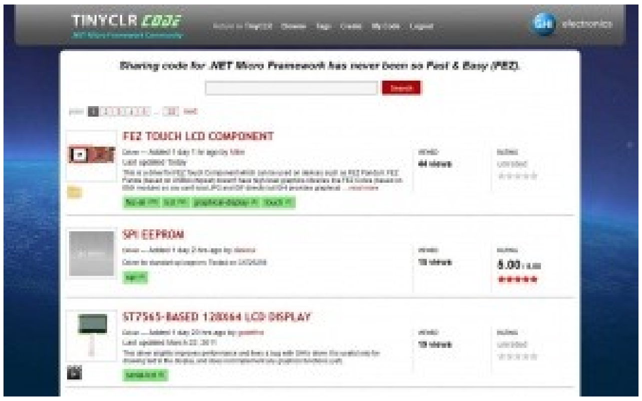
Nachdem Sie F5 gedrückt haben, wird das Programm auf das FEZ Board übertragen und läuft jetzt auf der realen Hardware. Der Wechsel vom Emulator auf die reale Hardware ist so einfach!

Probieren Sie die selben Schritte, die wir bereits mit dem Emulator getan haben, wie das Setzen von Breakpoints und mit F11 durch den Code steppen. Beachten Sie, dass "Debug.Print" nach wie vor die Debug-Meldungen aus der Hardware bis zum Ausgabe-Fenster von Visual C# Express sendet.

6 Komponenten Treiber

Für alle FEZ Komponenten (LEDs, Taster, Temperatur-Sensor, Relays, Servo-Treiber...etc.) und FEZ Shields (Ethernet, LCD, Motor-Treiber...etc.) gibt es Beispiel Treiber. Diese Treiber sind so geschrieben, das keine Vorkenntnisse über die Hardware vorhanden sein müssen. Um beispielsweise eine LED blinken zu lassen, schicken Sie einfach ein Kommando an den Treiber. Man muß nichts wissen über Prozessor Pins und wie man den Zustand eines Pins ändert... usw. Auf der anderen Seite, lehrt dieses Buch die Grundlagen. Verwenden Sie die Komponenten Treiber, als Ausgangspunkt und benutzen Sie dann dieses Buch um zu verstehen, was der Treiber tatsächlich tut.

Alle Komponenten Treiber und Hunderte mehr sind verfügbar auf http://code.tinyclr.com


TINYCLR CODE
electronics
Sharing code for .NET Micro Framework has never been so Fast & Easy (FEZ).
FEZ TOUCH LCD COMPONENT
SPI EEPROM
5.00
ST7565-BASED 128X64 LCD DISPLAY

7 German C-Sharp Level1

Dieses Buch ist nicht dazu gedacht, Ihnen C# beibringen, aber wir werden die meisten Grundlagen abdecken damit Sie loslegen können.

Damit das Lernen von C# nicht zu langweilig wird, unterteilen wir es in verschiedene Ebenen. So können wir viele lustige Dinge mit NETMF tun und dann wenn notwendig zurück zu C# gehen.

7.1 Was ist .NET?

Microsoft entwickelt das .NET Framework umd die Programmierung zu standardisieren. (Beachten Sie, dass wir nicht über das vollständige. NET Framework reden und auch nicht über das Micro Framework.) Es gibt eine Reihe von Bibliotheken, die Entwickler von vielen Programmiersprachen aus verwenden können. Das .NET Framework läuft nur auf PCs und nicht auf kleineren Geräten, da es ein sehr großes Framework ist. Auch hat das komplette Framework viele Dinge, die auf kleineren Geräten nicht möglich oder sehr nützlich wären. Das ist der Grund warum das .NET Compact Framework entwickelt wurde. Beim Compact Framework wurden nicht benötigte Bibliotheken entfernt um die Größe des Frameworks zu schrumpfen. Diese kleinere Version läuft auf Windows CE und Smartphones. Das Compact Framework ist kleiner als das .NET Framework, aber es ist immer noch zu groß für Mini-Geräte. Aufgrund seiner Größe und weil es ein Betriebssystem voraussetzt.

Das .NET Micro Framework ist die kleinste Version der Frameworks. Es wurden mehrere Bibliotheken entfernt und es ist Betriebssystem unabhängig. Wegen der Ähnlichkeit zwischen diesen drei Frameworks, kann man fast denselben Code jetzt auf PCs und kleinen Geräte laufen lassen, mit wenigen oder ohne Modifikationen.

Zum Beispiel funktioniert die serielle Schnittstelle auf einen PC, WinCE-Gerät oder FEZ (USBizi) auf dieselbe Weise, bei der Verwendung von .NET.

7.2 Was ist C#?

C und C++ sind die populärsten Programmiersprachen. C# ist eine aktualisierte und moderne Version von C und C++. Es enthält alles, was Sie von einer modernen Sprache erwarten können, wie Garbage Collector und Laufzeit-Validierung. Es ist auch eine objektorientierte Sprache, was Programme besser portierbar und das Debuggen vereinfacht.

Obwohl C# eine Menge von Vorschriften an die Programmierung stellt, um die Bug-Möglichkeiten zu verringern, bietet es immer noch die meisten der leistungsstarken Funktionen die C/C++ bietet.

7.3 ?Main? ist der Startpunkt

Wie wir zuvor gesehen haben, beginnen Programme immer mit einer Methode namens Main. Eine Methode ist ein kleines Stück Code, dass eine bestimmte Aufgabe hat. Methoden beginnen und enden mit öffnenden / schließenden geschweiften Klammern. Bei unserem ersten Programm hatten wir nur eine Zeile Code zwischen den geschweiften Klammern. Die Zeile war

```
Debug.Print("Amazing!");
```

Sie sehen, dass die Zeile mit einem Semikolon endet. Alle Zeilen enden auf die gleiche Weise. Diese Zeile ruft die Print-Methode auf, die im Debug-Objekt vorhanden ist. Die Methode wird mit der Zeichenfolge "Amazing!" aufgerufen.

Verwirrt? Versuchen wir das etwas aufzuklären. Angenommen, Sie sind ein Objekt. Sie haben auch mehrere Methoden, um das Objekt zu steuern. Eine Methode könnte "Sit" heissen und eine andere könnte "Run" heissen. Was nun, wenn wir "Amazing!" sagen wollen? Ich würde die "Speak" Methode aufrufen mit dem Satz (string) "Amazing!". So würde der Code aussehen

```
You.Say(?Amazing!?);
```

Nun, warum brauchen wir die Anführungszeichen vor und nach dem Wort Amazing? Das liegt daran, das C# nicht weiß, ob der Text den Sie schreiben wollen, ein Befehl ist oder ob es tatsächlich ein Text (Strings) ist. Sie können sehen, wie der Text in rot gefärbt ist, wenn Sie Anführungszeichen hinzuzufügen. Das erleichtert das Lesen des Codes für uns Menschen.

7.4 Kommentare

Was, wenn Sie Kommentare/Notizen/Warnungen im Code hinzufügen möchten? Diese Kommentare werden Ihnen und anderen helfen, den Code besser zu verstehen. C# ignoriert diese Kommentare. Es gibt 2 Arten von Kommentaren, Zeilen Kommentare und Block Kommentare. Kommentare (ignorierter Text) werden in grün dargestellt.

Für einen Zeilen Kommentar oder einen Kommentar zu einem Teil einer Zeile fügen Sie // vor dem Kommentartext. Die Farbe des Textes wird nun grün anzeigt, und der Kommentar Text wird von C# ignoriert.

```
using System;
using Microsoft.SPOT;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            // This is a comment
            Debug.Print("Amazing!");//this is a comment too!
        }

    }
}
```

Sie können auch einen ganzen Kommentar Block erstellen. Starten Sie den Kommentar mit /* und dann beenden Sie ihn mit */

```
using System;
using Microsoft.SPOT;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            /* This is a comment
             it it still a comment
             the block will end now */
            Debug.Print("Amazing!");
        }

    }
}
```

7.5 While-Schleife

Es ist Zeit für unser erstes Stichwort "while". Die while-Schleife beginnt und endet mit geschweiften Klammern zwischen denen der Code steht. Alles innerhalb der Klammern wird solange durchlaufen, solange ein Ausdruck wahr (true) ist. Zum Beispiel kann ich Sie bitten, dieses Buch zu lesen "while" du wach bist!

So können wir ein Programm schreiben, das endlos "Amazing!" ausgibt. Diese Endlos-Schleife hat kein Ende, da der Ausdruck immer "true" ist.

```
using System;
using Microsoft.SPOT;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            while(true)
            {
                Debug.Print("Amazing!");
            }
        }

    }
}
```

Im obigen Code wird die Ausführung wie gewohnt mit der "Main"-Methode starten und dann mit der nächsten Zeile, der while-Schleife fortfahren. Die while-Schleife bestimmt die Laufzeit, in der der Code innerhalb der Klammern ausgeführt wird, solange der Ausdruck "wahr" ist. Im Moment gibt es noch keine Anweisungen, so dass der Ausdruck "true" ist, was bedeutet das diese Schleife immer ausgeführt wird.

Drücken Sie nicht F5, um das Programm auszuführen, sonst wird das Ausgabe-Fenster mit dem Wort "Amazing!" überflutet. Stattdessen drücken Sie F11 um den Code Schrittweise auszuführen um zu verstehen, wie die Schleife funktioniert. Beachten Sie, dass dieses Programm nie beendet wird, sie müssen es mit Shift + F5 abbrechen.

Hinweis: Sie können alle diese Debug-Short Cuts aus dem Menü unter Debug erreichen.

7.6 Variablen

Variablen sind für Ihre Anwendung reservierte Orte im Speicher. Die Größe des reservierten Speichers hängt von der Art der Variablen ab. Wir werden hier nicht jeden einzelnen Typ erkären aber jedes C# Buch wird Ihnen diese im Detail erklären. Wir werden mit der int-Variable beginnen. Diese Art von Variable wird verwendet, um ganze Zahlen zu speichern.

Einfach gesagt

```
int MyVar;
```

wird dem System mitteilen, dass es Speicher reservieren soll. Dieser Speicher wird der MyVar zugewiesen. Sie können einen beliebigen Namen wählen so lange der Name keine Leerzeichen enthält. Jetzt können Sie jede ganze Zahl in diesem Speicher ablegen. MyVar = 1234; Sie können ebenso mathematische Operationen verwenden um Zahlen zu berechnen. MyVar = 123 + 456; oder Sie können eine Zahl um eins erhöhen (inkrementieren)

```
MyVar++;
```

oder um eins dekrementieren (abziehen)

```
MyVar- -;
```

Mit diesem Wissen können wir ein Programm schreiben, das das Wort "Amazing!" dreimal ausgibt. Hier ist der Code

```
using System;
using Microsoft.SPOT;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            int MyVar;
            MyVar = 3;
            while(MyVar>0)
            {
                MyVar--;
                Debug.Print("Amazing!");
            }
        }

    }
}
```

Hinweis: hier ist die while-Schleife Aussage nicht mehr immer "wahr", sondern nur solange MyVar > 0 ist. Dies bedeutet, die Schleife wird solange durchlaufen solange der Wert der MyVar Variable größer als 0 ist. Im ersten Schleifen Durchlauf ist MyVar 3. Mit jedem Schleifen Durchlauf wird MyVar um eins dekrementiert. Das ergibt, dass die Schleife genau dreimal durchlaufen wird und das Ergebnis "Amazing!" dreimal ausgegeben wird.

Lasst uns etwas interessanteres tun. Ich möchte die Zahlen 1 bis 10 ausgeben lassen. OK, wir wissen, wie man eine Variable definiert, und wir wissen wie man eine Variable erhöht, aber wie kann man eine Zahl auf dem Debug-Ausgabe Fenster ausgeben? Einfach MyVar als Parameter zu Debug.Print angeben ergibt einen Fehler, und es wird nicht funktionieren. Dies liegt daran, dass Debug.Print nur Strings akzeptiert, keine Zahlen. Wie können wir eine Integer-Variable "In einen String" konvertieren? Es ist ganz einfach, rufen Sie MyVar.ToString() auf. Das war einfach!

```
using System;
using Microsoft.SPOT;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            int MyVar;
            MyVar = 0;
            while(MyVar<10)
            {
                MyVar++;
                Debug.Print(MyVar.ToString());
            }
        }

    }
}
```

Die letzte Sache die wir hinzufügen möchten ist das, was das Programm ausgeben soll

```
Count: 1
Count: 2
...
...
Count: 9
Count:10
```

Dies kann leicht durch hinzufügen weiterer Strings erfolgen. Strings werden mit dem +-Zeichen hinzugefügt, als ob Sie Zahlen addieren möchten. Versuchen Sie den folgenden Code

```
using System;
using Microsoft.SPOT;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            int MyVar;
            MyVar = 0;
            while(MyVar<10)
            {
                MyVar++;
                Debug.Print("Count: " + MyVar.ToString());
            }
        }

    }
}
```

7.7 Assemblies

Assemblies sind Dateien, die compilierten (assemblierten) Code enthalten. Dies ermöglicht Entwickler den Code zu verwenden, aber sie haben keinen Zugang zum Quellcode. Wir hatten bereits vorher Debug.Print verwendet. Wer hat die Debug-Klasse/Objekt geschrieben und wer die darin einhaltene Print-Methode? Diese Anrufe wurden durch das NETMF-Team bei Microsoft geschrieben. Sie kompilieren den Code und geben die Assembly an, die Sie benutzen wollen. Auf diese Weise werden die Benutzer nicht mit dem internen Code verwirrt, aber sie können ihn verwenden.

Am Anfang der bisher verwendeten Code Beispiel sehen wir, dass Microsoft.SPOT verwendet wird. Dies bedeuted in C#, dass Sie den "Namespace" Microsoft.SPOT verwenden. Okay, was ist dann ein Namensraum? Die Programme werden in Regionen "spaces" unterteilt. Dies ist sehr wichtig, wenn Programme sehr groß werden. Jedes Stück Code oder jede Bibliothek hat einen "Namen" für seinen "Space" zugeordnet. Programme mit dem gleichen "Namespace" können einander sehen, aber wenn der Namespace ein anderer ist, dann können wir C# optional anordnen den anderen Namensraum auch zu benutzen "use".

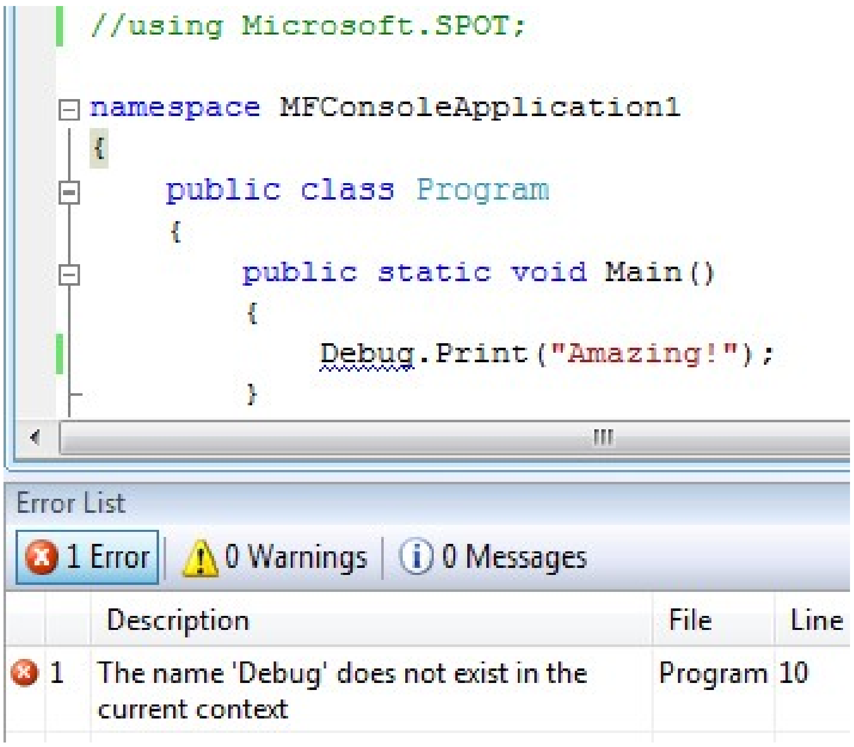
Der "Namensraum" für unser Programm ist der Namespace MFConsoleApplication1 Um andere Namesräume wie "Microsoft.SPOT" zu verwenden, müssen Sie diese hinzufügen:

```
using Microsoft.SPOT;
```

Was bedeutet SPOT überhaupt? Hier ist eine kurze Geschichte dazu! Vor ein paar Jahren, startete Microsoft ein Projekt namens SPOT. Sie erkannten, dass dieses Projekt eine gute Idee war und wollte es Entwicklern anbieten. Sie beschlossen, den Produktnamen in .NET Micro Framework zu ändern aber sie hielten den Namensraum wegen der Abwärtskompatibilität. Kurz gesagt SPOT ist NETMF!

Zurück zur Codierung, wenn Sie nun versuchen die Zeile using Microsoft.SPOT; zu entfernen oder aus zu kommentieren wir Ihr Programm nicht mehr funktionieren!

Hier ist die Fehlermeldung, nachdem Sie using Microsoft.SPOT; auskommentiert haben.



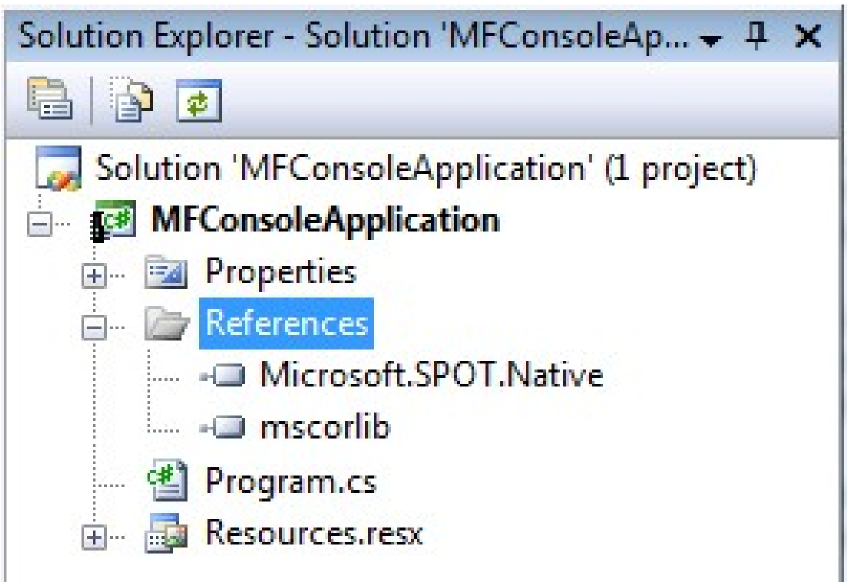
```
//using Microsoft.SPOT;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            Debug.Print("Amazing!");
        }
```

Error List

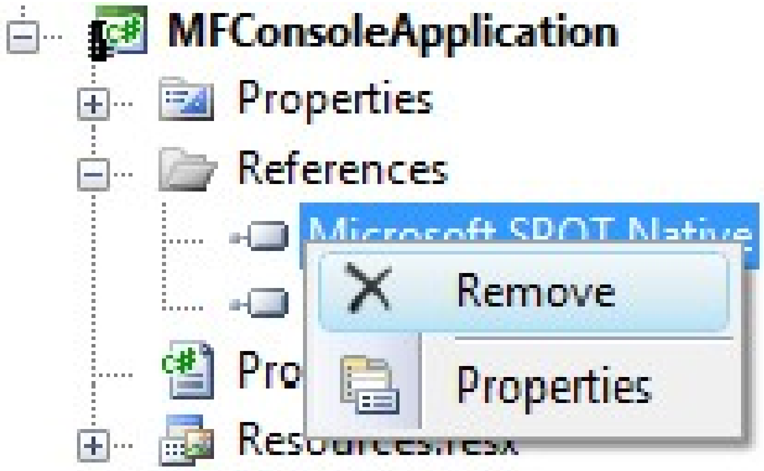
1 Error | 0 Warnings | 0 Messages

		Description	File	Line
	1	The name 'Debug' does not exist in the current context	Program	10

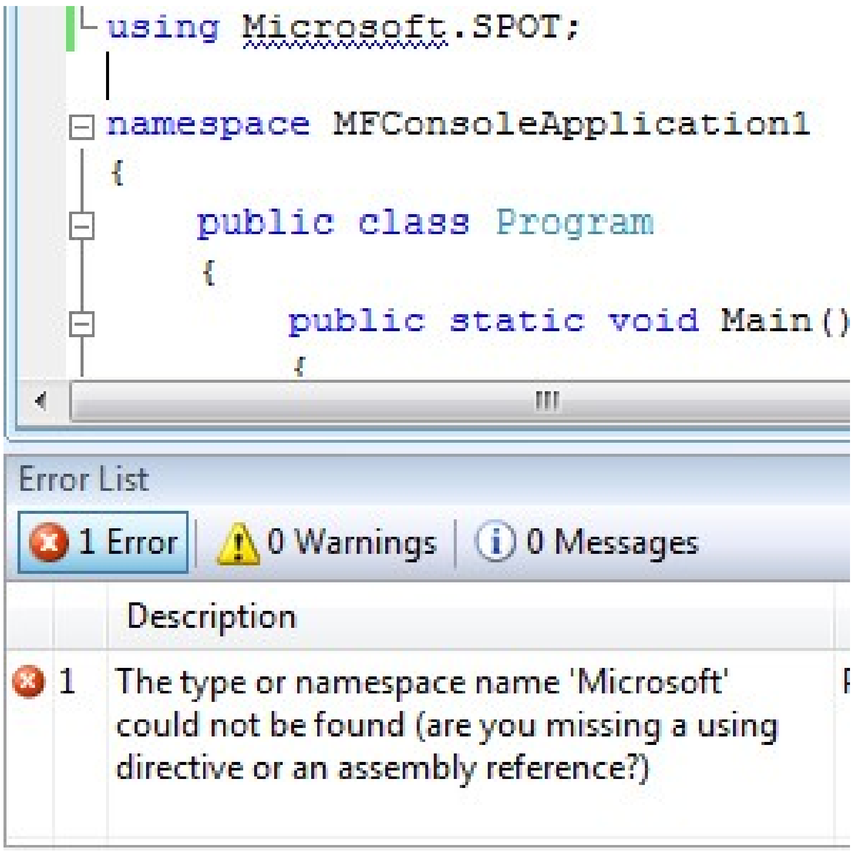
Wir benutzen Assemblies aber wo werden sie dem Projekt hinzugefügt? Nehmen sie den Kommentar wieder weg und stellen Sie sicher, dass alles noch funktioniert. Nun schauen Sie ins "Explorer"-Fenster. Klicken Sie auf das kleine Pluszeichen bei dem Wort "Referenzen", und Sie sollten zwei Assemblies sehen.



Nun machen Sie einen Rechts-Klick auf ?Microsoft.SPOT.Native?, dann klicken Sie auf ?Remove?



Unser Programm ist noch genau das gleiche wie vorher, aber nun fehlt eine sehr wichtige Assembly. Versuchen Sie es starten und Sie werden folgendes sehen



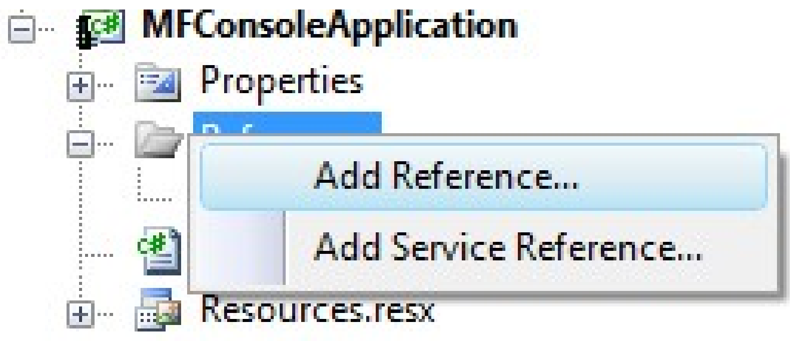
```
using Microsoft.SPOT;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
```

Error List

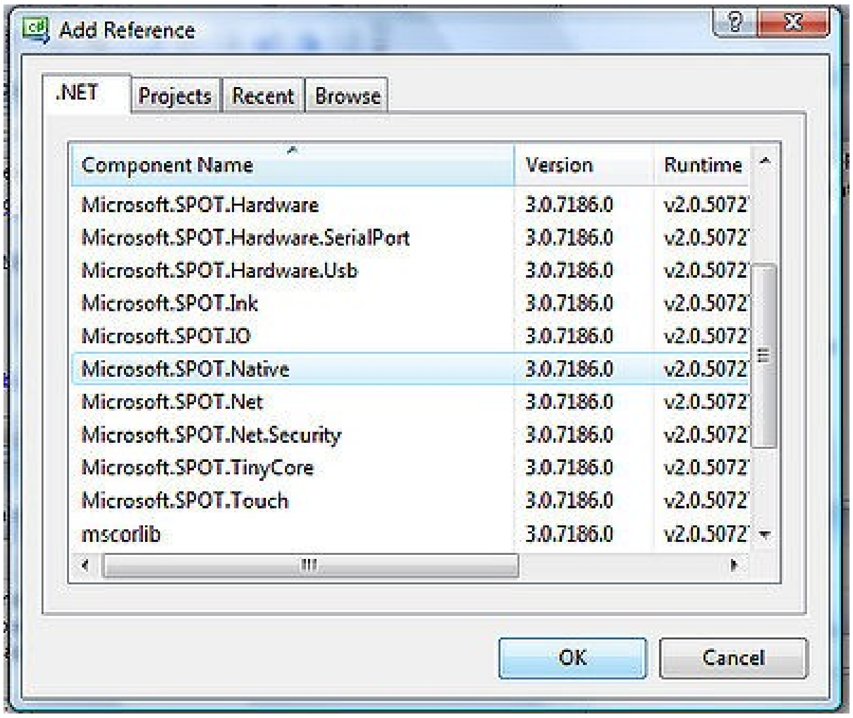
1 Error | 0 Warnings | 0 Messages

		Description
✖	1	The type or namespace name 'Microsoft' could not be found (are you missing a using directive or an assembly reference?)

Fügen wir die Assembly wieder dazu und stellen sicher, dass unser Programm noch läuft. Rechts-Klick auf "Referenzen" und wählen Sie "Verweis hinzufügen ..."



Im neuen Fenster wählen Sie ".NET" und dann wählen Sie "Microsoft.SPOT.Native" und klicken auf OK.



Component Name	Version	Runtime
Microsoft.SPOT.Hardware	3.0.7186.0	v2.0.5072
Microsoft.SPOT.Hardware.SerialPort	3.0.7186.0	v2.0.5072
Microsoft.SPOT.Hardware.Usb	3.0.7186.0	v2.0.5072
Microsoft.SPOT.Ink	3.0.7186.0	v2.0.5072
Microsoft.SPOT.IO	3.0.7186.0	v2.0.5072
Microsoft.SPOT.Native	3.0.7186.0	v2.0.5072
Microsoft.SPOT.Net	3.0.7186.0	v2.0.5072
Microsoft.SPOT.Net.Security	3.0.7186.0	v2.0.5072
Microsoft.SPOT.TinyCore	3.0.7186.0	v2.0.5072
Microsoft.SPOT.Touch	3.0.7186.0	v2.0.5072
mscorlib	3.0.7186.0	v2.0.5072

Versuchen Sie das Programm wieder zu starten. Wenn Sie irgendwelche Fehler haben, bitte gehen Sie zurück und lesen Sie, wie diese zu beheben sind.

7.7.1 Welche Assemblies werden benötigt?

In diesem Buch stelle ich zahlreiche Beispiele vor, aber ich sage Ihnen nicht, welche Assemblies benötigt werden. Das ist wirklich einfach anhand der Dokumentation herauszufinden, manchmal kann es aber auch schwierig sein. Warum fügt man einfach nicht alle hinzu? Als Anfänger sind Ihre Anwendungen noch sehr klein, so dass Sie sehr viel Speicher haben, auch wenn Sie alle Assemblies hinzufügen, auch wenn Sie sie nicht verwenden. Die Assemblies hier unten werden am häufigsten verwendet. Fügen sie alle zu Ihrem Projekt hinzu, für den Moment. Sobald Sie wissen wie alles funktioniert, können Sie beginnen jene zu entfernen, die Sie nicht brauchen.

```
GHIElectronics.NETMF.Hardware
GHIElectronics.NETMF.IO
GHIElectronics.NETMF.System
Microsoft.SPOT.Hardware
Microsoft.SPOT.Native
Microsoft.SPOT.Hardware.SerialPort
Microsoft.SPOT.IO
mscorlib
System
System.IO
```

Vergessen Sie nicht, eine der folgenden Assembly hinzu zu fügen je nach dem welches Gerät Sie verwenden. Diese Assemblies enthalten die Pin-Definitionen.

```
FEZMini_GHIElectronics.NETMF.FEZ
FEZDomino_GHIElectronics.NETMF.FEZ.
```

7.8 Threading

Dies ist ein sehr fortgeschrittenes Thema. Beachten Sie, dass hier nur sehr grundlegende Informationen abgedeckt werden.

Prozessoren/Programme führen nur einen Befehl auf einmal aus. Denken Sie daran, wie wir im Einzelschritt durch den Code gesteppt sind? Nur eine Anweisung wurde ausgeführt und dann ging es weiter zur nächsten Anweisung. Wie ist es dann möglich, dass Ihr PC mehrere Programme gleichzeitig ausführen kann? Eigentlich führt Ihr PC nie mehrere Programme gleichzeitig aus! Statt dessen läuft jedes Programm nur für eine kurze Zeit, dann hält er es an und führt das nächste Programm aus.

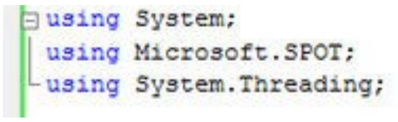
Generell ist Threading nicht für Anfänger empfohlen, aber es gibt Dinge, die mit Verwendung von Threads viel leichter realisiert werden können. Zum Beispiel, Sie wollen eine LED blinken lassen. Es wäre schön, die LED in einem separaten Thread blinken zu lassen und sich nicht im Hauptprogramm darum zu kümmern.

Außerdem erfordern Delays im Programm den Threading Namespace. Sie werden das besser in den kommenden Beispielen verstehen.

Apropos, LED steht für Light Emitting Diode. Sie sehen LEDs überall um Sie herum. Werfen Sie einen Blick auf Ihren TV, DVD oder andere elektronische Geräte, und Sie werden rote oder anders farbige Leuchten sehen. Diese sind LEDs.

FEZ enthält eine LED-Bibliothek um die Ansteuerung von LEDs noch weiter zu vereinfachen. Dieses Buch erklärt Ihnen, wie man Pins/Geräte direkt steuern kann.

Fügen Sie ?using System.Threading? zu Ihrem Programm dazu.



```
using System;
using Microsoft.SPOT;
using System.Threading;
```

Das ist alles was wir brauchen, um Threads zu verwenden! Es ist wichtig zu wissen, dass unser Programm selbst ein Thread ist. Am Anfang der System Ausführung schaut C# nach der "Main" Funktion und führt diese in einem Thread aus. Wir wollen nun eine Verzögerung in unserem Thread (unser Programm), hinzuzufügen, damit es einmal pro Sekunde das Wort "Amazing!" ausgibt. Um einen Thread zu verzögern schicken wir ihn schlafen mit "Sleep". Beachten Sie, dass dieser Schlaf nicht für das gesamte System gilt. Es wird nur der Thread schlafen geschickt.

```
Add  Thread.Sleep(1000);
```

Die ?Sleep? Methode erwartet die Zeit in Millisekunden als Parameter. Für 1 Sekunde werden 1000 Millisekunden benötigt.

```
using System;
using Microsoft.SPOT;
using System.Threading;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            while (true)
            {
                Debug.Print("Amazing!");
                Thread.Sleep(1000);
            }
        }

    }
}
```

Versuchen Sie, das Programm auszuführen und schauen Sie auf das Ausgabe-Fenster. Wenn Sie es auf dem Emulator probieren, wird es nicht genau 1 Sekunde sein. Probieren Sie es dagegen auf echter Hardware (FEZ), dann wird sehr genau 1 Sekunde sein.

Lassen Sie uns einen zweiten Thread erzeugen (unser erster wurde automatisch erstellt, schon vergessen?). Wir müssen dazu einen neuen Thread-Objekt-Handler (Referenz) erstellen und diesem einen sinnvollen Namem geben, wie MyThreadHandler. Und Sie brauchen eine neue lokale Methode; nennen Sie diese MyThread. Führen Sie dann den neuen Thread aus.

Wir sind nicht mehr im "Main"-Thread, deshalb werden wir diesen in einen endlosen Ruhezustand versetzen. Hier ist das Code Listing. Kümmern Sie sich nicht darum, wenn Sie nicht alles verstehen. Alles was Sie an dieser Stelle wissen müssen ist, wie man einen Thread zum schlafen "Sleep" bringt.

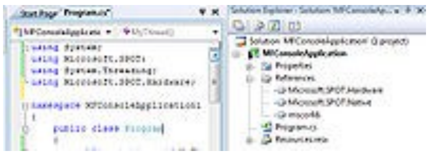
```
using System;
using Microsoft.SPOT;
using System.Threading;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void MyThread()
        {
            while (true)
            {
                Debug.Print("Amazing!");
                //sleep this thread for 1 second
                Thread.Sleep(1000);
            }
        }
        public static void Main()
        {
            // create a thread handler
            Thread MyThreadHandler;
            // create a new thread object
           //and assign to my handler
            MyThreadHandler = new Thread(MyThread);
            // start my new thread
            MyThreadHandler.Start();

            /////////////////////////////////
            // Do anything else you like to do here
            Thread.Sleep(Timeout.Infinite);
        }
        }
}
```

8 Digitale Eingänge & Ausgänge

Auf Prozessoren gibt es viele "digitale" Pins, die als Eingänge oder Ausgänge genutzt werden können. wenn wir "digital" Pin sagen, verstehen wir darunter Pins die entweder "Eins" oder "Null" sein können. Wichtiger Hinweis: Statische Entladung von allem, einschließlich des menschlichen Körpers, kann den Prozessor beschädigen. Sie kennen das, wenn Sie manchmal jemanden oder etwas berühren, fühlt es sich ein wenig nach elektrostatischer Entladung an? Diese kleine Entladung ist hoch genug, um elektronische Schaltungen zu töten. Profis nutzen Ausrüstungen und Vorkehrungen beim Umgang mit elektostatisch empfindlichen Geräten. Möglicherweise haben Sie nicht diese Geräte, deshalb sollten Sie wenn möglich jede unnötige Berührung der Schaltung vermeiden. Sie können auch ein Anti-Statik-Armband verwenden.

NETMF unterstützt digitale Ein- und Ausgangs Pins durch die ?Microsoft.SPOT.Hardware? Assembly und Namensraum.

Fahren Sie fort und fügen Sie die Assembly und den Namespace ein, wie wir es vorher gelernt haben.



```
using System;
using Microsoft.SPOT;
using System.Threading;
using Microsoft.SPOT.Hardware;

namespace MFConsoleApplication1
{
    public class Program
```

Wir sind nun bereit die digitalen Pins zu benutzen.

8.1 Digitale Ausgänge

Wir wissen, dass ein digitaler Ausgang Pin auf Null oder Eins gesetzt werden kann. Beachten Sie, dass mit Eins nicht 1 Volt gemeint ist, sondern es bedeutet, dass am Pin eine Spannung anliegt. Wenn der Prozessor mit 3,3V versogt wird, dann bedeutet der Zustand Eins an einem Pin, dass 3,3V am Ausgangs-Pin anliegt. Es sind nicht exakt 3,3V, aber sehr nahe. Wenn der Pin auf Null gesetzt wird, ist die Spannung ungefähr 0V.

Die digitalen Pins sind sehr schwach! Sie können nicht dazu verwendet werden, um Geräte, die viel Kraft erfordern zu treiben. Zum Beispiel kann ein Motor zwar mit 3,3V laufen, aber Sie können ihn nicht direkt an einem digitalen Pin des Prozessors anschliessen. Der Grund ist, dass der Prozessor Ausgang zwar 3,3V liefern kann aber nur sehr wenig Leistung. Sie können aber eine kleine LED anschliessen oder ein "Signal" 1 oder 0 an einem anderen Eingang Pin anlegen.

Alle FEZ boards haben eine LED die an einem digitalen Pin angeschlossen ist. Wir wollen diese LED zum blinken bringen. Digitale Ausgangs-Pins werden durch OutputPort Objekte angesteuert. Zuerst erstellen wir den Objekt Handler (Referenz), und dann erstellen wir ein neues Objekt OutputPort und weisen es unserem Handler zu. Beim Erstellen eines neuen Objekts OutputPort, müssen Sie den Anfangs Zustand des Pins, 1 oder 0 angeben. Die Eins und Null kann auch für HIGH oder LOW stehen und ebenso kann auch HIGH für TRUE und LOW für FALSE stehen. Wir setzen in diesem Beispiel den Pin auf TRUE (HIGH) um die on-Board LED einzuschalten.

Hier ist der Code unter Verwendung der Pin-Nummer 4, der FEZ Domino on-Board LED.

```
using System;
using Microsoft.SPOT;
using System.Threading;
using Microsoft.SPOT.Hardware;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            OutputPort LED;
            LED = new OutputPort((Cpu.Pin)4, true);

            Thread.Sleep(Timeout.Infinite);
        }

    }
}
```

Das FEZ SDK enthält die Assemblies "FEZMini_GHIElectronics.NETMF.FEZ" und "FEZDomino_GHIElectronics.NETMF.FEZ". Fügen Sie die entsprechenden Assembly zu Ihrem Programm hinzu und dann auch noch: "FEZ_GHIElectronics.NETMF.System". Jetzt ändern Sie den Code, indem Sie "Hilfe GHIElectronics.NETMF.FEZ"an die Spitze Ihres Programm Codes setzen. Hier ist der Code, diesmal mit der FEZ Pin Enumerationsklasse.

```
using System;
using Microsoft.SPOT;
using System.Threading;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.FEZ;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            OutputPort LED;
            LED = new OutputPort((Cpu.Pin)FEZ_Pin.Digital.LED, true);

            Thread.Sleep(Timeout.Infinite);
        }
    }
}
```

Sehen Sie, wieviel einfacher das ist? Wir müssen wirklich nicht genau wissen, wo die LED angeschlossen ist.

Führen Sie das Programm aus und beobachten Sie die LED. Sie sollte jetzt leuchten. Die Dinge werden immer spannender!

8.1.1 Eine LED blinken lassen

Um eine LED blinken zu lassen, müssen wir den Pin HIGH setzen und eine Verzögerung für einige Zeit einfügen, dann müssen wir den Pin LOW setzen und nochmals die Verzögerungs Zeit einfügen. Es ist wichtig daran zu denken, zweimal zu verzögern. Warum? Weil unsere Augen auch für Computer-Systeme zu langsam sind. Wenn die LED nur kurz aufleuchtet, und dann wieder sehr schnell ausgeschaltet wird, können deine Augen das nicht sehen, es ist eine zu kurze Zeit. Was brauchen wir, um eine LED blinken zu lassen? ... Wir haben gelernt, wie man eine while-Schleife benutzt, wir wissen, wie man eine Verzögerung hinzufügt und wir müssen wissen, wie man einen Pin auf HIGH oder LOW setzt. Dies geschieht durch den Aufruf der Write-Methode des OutputPort Objekt. Beachten Sie, dass Sie hier nicht "OutputPort.Write" verwenden können. Das ist falsch, denn auf was soll sich der Ausgangsports beziehen? Verwenden Sie stattdessen "LED.Write", um die Szene komplett zu machen. Hier ist der Code, um die on-Board-LED auf dem FEZ blinken zu lassen

```
using System;
using Microsoft.SPOT;
using System.Threading;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.FEZ;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            OutputPort LED;
            LED = new OutputPort((Cpu.Pin)FEZ_Pin.Digital.LED, true);
            while (true)
            {
                LED.Write(!LED.Read());

                Thread.Sleep(200);
            }
        }
    }
}
```

Dies ist ein weiterer, einfacherer Weg, um eine LED blinken zu lassen.

```
using System;
using Microsoft.SPOT;
using System.Threading;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.FEZ;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            OutputPort LED;
            LED = new OutputPort((Cpu.Pin)FEZ_Pin.Digital.LED, true);
            while (true)
            {
                LED.Write(true);
                Thread.Sleep(200);

                LED.Write(false);
                Thread.Sleep(200);
            }
        }
    }
}
```

Lassen Sie uns sehen, ob Sie die Delay Zeit ändern können, um die LED schneller oder langsamer blinken zu lassen. Versuchen Sie auch, einen anderen Wert für den Zustand zu verwenden, damit die LED lange Zeit an ist und dann für kurze Zeit aus ist.

Wichtiger Hinweis: Niemals zwei Ausgangs-Pins zusammen schliessen. Wenn sie miteinander verbunden sind und ein Pin wird auf HIGH gesetzt und der andere auf LOW, wird der Prozessor beschädigt. Verbinden die einen Ausgangs Pin nur mit einen Eingang Pin, einer Treiberschaltung oder an eine einfache Last wie eine LED.

8.2 Digitale Eingänge

Digitale Eingänge erkennen, ob der Status eines Pins HIGH oder LOW ist. Es gibt einige Einschränkungen für Eingangs-Pins. Zum Beispiel ist die minimale Spannung am Pin 0 Volt. Eine negative Spannung kann zu Schäden am Pin oder dem Prozessor führen. Ebenso darf die maximale Eingangsspannung an dem Pin kleiner sein als die Prozessor Versorgungsspannung. Alle GHI Electronics Boards benutzen Prozessoren, die mit 3,3V laufen, so dass die höchste Spannung am Pin ebenfalls 3,3V sein sollte. Dies gilt für ChipworkX aber nicht für Embedded Master und USBizi, diese Prozessoren sind 5V-tolerant. Dies bedeutet, dass, obwohl der Prozessor auf 3,3V läuft, bis zu 5V an seinen Eingängen anliegen dürfen. Die meisten digitalen Chips, die Sie anschliessen können haben 5V Versorgungsspannung. Dadurch ist es möglich diese digitalen Schaltungen mit unserem Prozessor zu verwenden.

Hinweis: FEZ basiert auf USBizi und ist damit 5V tolerant.

Wichtiger Hinweis: 5V-tolerant, bedeutet nicht, dass der Prozessor mit 5V versorgt werden kann. Immer nur mit 3.3V. Nur die Eingangs-Pins vertragen 5V.

Ein InputPort Objekt wird verwendet, um digitale Eingangs-Pins zu behandeln. Jeder Pin auf den von GHI verwendeten Prozessoren, kann Ein-oder Ausgang sein. Aber natürlich nicht beides gleichzeitig! Unbenutzte Eingangs-Pins werden floating genannt. Sie werden denken, dass unbenutzte Eingangs-Pins LOW sind, aber das ist nicht wahr. Wenn ein Eingangs-Pin nicht angeschlossen ist, ist er offen für alle Umgebungsgeräusche, diese können den Pin nach LOW oder HIGH ziehen. Um dieses Problem zu beseitigen verfügen moderne Prozessoren über interne Pull-Down-oder Pull-up-Widerstände, die in der Regel durch Software gesteuert werden. Aktivieren des Pull-up Widerstand wird den Pin auf HIGH ziehen. Beachten Sie, dass der Pull-up Widerstand keinen Pin auf HIGH setzt, sondern er zieht ihn auf HIGH. Wenn nichts angeschlossen ist, ist der Pin standardmäßig HIGH.

Es gibt viele Einsatzmöglichkeiten für Eingangs-Ports aber die häufigste ist, sie mit einer Schaltfläche oder einem Schalter zu verbinden. FEZ enthält bereits einen On-Board Taster, der mit dem Loader Pin verbunden ist. Der Loader Pin ist dazu da, beim Einschalten in den Bootloader Mode zu kommen, aber wir können diesen Pin auch zur Laufzeit verwenden. Der Taster ist mit "LDR" oder "Loader" beschriftet.

Der Taster verbindet Masse mit dem Eingangs-Pin. Wir werden auch den Pull-up Widerstand aktivieren. Dies bedeutet, dass der Pin HIGH ist (Pull-up), wenn die Taste nicht gedrückt ist und LOW (geerdet), wenn die Taste gedrückt wird. Wir lesen den Status der Taste und geben den Zustand auf der LED aus. Beachten Sie, dass der Pin HIGH ist, wenn der Knopf nicht gedrückt ist (nach HIGH gezogen wird) und er ist LOW, wenn die Taste gedrückt wird. Das heißt, die LED erlischt, wenn der Knopf gedrückt wird.

Der code:

```
using System;
using Microsoft.SPOT;
using System.Threading;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.FEZ;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            OutputPort LED;
            InputPort Button;

            LED = new OutputPort((Cpu.Pin)FEZ_Pin.Digital.LED, true);
            Button = new InputPort((Cpu.Pin)FEZ_Pin.Digital.LDR, false,
                                   Port.ResistorMode.PullUp);
            while (true)
            {
                LED.Write(Button.Read());
                Thread.Sleep(10);
            }
        }
    }
}
```

Beim Erstellen des InputPort Objekt übergeben wir "false" als Parameter. Das wird für das Glitch-Filter verwendet. Dies wird später erläutert. Auch ist es vielleicht verwirrend, wie wir den Zustand eines InputPort übergeben um einen OutputPort zu setzten. Wir werden dies im nächsten Abschnitt erläutern.

8.2.1 Interrupt Port

Wenn wir den Status eines Pin überprüfen wollen, muß man seinen Zustand regelmäßig überprüfen. Dieses verschwendet Prozessor-Zeit mit etwas unwichtigem. Sie werden den Pin vielleicht eine Million Mal abfragen, bevor er gedrückt wird! Interrupt-Ports ermöglichen es uns, eine Methode zu verwenden, die nur ausgeführt wird, wenn die Taste gedrückt wird (z.B: wenn der Pin LOW ist).

Wir können den Interrupt an vielen Zustandsänderungen des Pins auslösen, wenn der Pin LOW ist oder vielleicht wenn er HIGH ist. Die häufigste Verwendung ist bei "on change". Der Wechsel von LOW zu HIGH oder von HIGH zu LOW erzeugt eine Flanke. Die HIGH Flanke tritt auf, wenn sich das Signal von Low nach HIGH ändert. Die Low-Flanke tritt auf, wenn das Signal von HIGH nach LOW fällt.

In the example below, I am using the low edge to detect a button press. ?IntButton_OnInterrupt? will automatically run when button is pressed.

```
using System;
using Microsoft.SPOT;
using System.Threading;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.FEZ;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            // the pin will generate interrupt on high and low edges
            InterruptPort IntButton =
                new InterruptPort((Cpu.Pin)FEZ_Pin.Interrupt.LDR, true,
                                   Port.ResistorMode.PullUp,
                                   Port.InterruptMode.InterruptEdgeLow);

            // add an interrupt handler to the pin
            IntButton.OnInterrupt +=
                new NativeEventHandler(IntButton_OnInterrupt);

            //do anything you like here
            Thread.Sleep(Timeout.Infinite);
        }

        static void IntButton_OnInterrupt(uint port, uint state,
                                          DateTime time)
        {
            Debug.Print("Button press");
        }
    }
}
```

Hinweis: Nicht alle Pins des Prozessors unterstützen Interrupts, aber die meisten. Zur leichteren Identifizierung der Interrupt-Pins, verwenden Sie die Enumeration für "Interrupt" anstelle von "Digital", wie im vorigen Code gezeigt wird.

8.2.2 Tristate Port

Was können wir tun, wenn ein Pin Eingang und Ausgang sein soll? Ein Pin kann nie gleichzeitig Eingang und Ausgang sein, aber wir können ihn als Ausgang setzten und ihn dann zum Eingang machen, um eine Antwort zurück zu lesen. Eine Möglichkeit ist den Pin zu entfernen "Dispose". Wir erzeugen einen Ausgang, verwenden ihn und dann entfernen wir ihn wieder. Dann können wir den Pin als Eingang anlegen und ihn einlesen.

NETMF unterstützt bessere Möglichkeiten dafür, durch Tristate-Port. Tristate bedeutet drei Zustände: Eingang, Ausgang LOW und Ausgang HIGH. Ein kleines Problem von Tristate-Pins ist, dass wenn Sie einen Pin als Ausgang gesetzt haben und dann erneut als Ausgang setzen, erhalten wir eine Schutzverletzung "Exception". Ein Ausweg um dieses Problem zu umgehen ist, indem Sie die Richtung des Pins vor dem Wechsel abfragen. Die Richtung des Pins findet man unter seiner Eigenschaft "Active", wobei man unter "false" Eingang und "true" Ausgang versteht. Ich persönlich empfehle die Verwendung von Tristate Ports nur wenn unbedingt nötig.

```
using System;
using Microsoft.SPOT;
using System.Threading;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.FEZ;

namespace MFConsoleApplication1
{
    public class Program
    {
        static void MakePinOutput(TristatePort port)
        {
            if (port.Active == false)
                port.Active = true;
        }
        static void MakePinInput(TristatePort port)
        {
            if (port.Active == true)
                port.Active = false;
        }
        public static void Main()
        {
               TristatePort TriPin =
                   new TristatePort((Cpu.Pin)FEZ_Pin.Interrupt.LDR, false,
                                    false, Port.ResistorMode.PullUp);
            MakePinOutput(TriPin);// make pin output
            TriPin.Write(true);
            MakePinInput(TriPin);// make pin input
            Debug.Print(TriPin.Read().ToString());
        }
    }
}
```

Hinweis: Aufgrund des internen Designs, wird TristatePort nur von Interrupt fähigen digitalen Pins unterstützt. Wichtiger Hinweis: Achten Sie darauf, dass der Pin nicht mit einem Taster verbunden ist, wenn der Pin als Ausgang HIGH gesetzt wird. Dies führt zur Beschädigung des Prozessors. Ich würde sagen, für Anfänger Anwendungen, brauchen Sie keine Tristate-Ports. Benutzen Sie Tristate-Ports erst, wenn Sie mit digitalen Schaltungen besser vertraut sind.

9 German C-Sharp Level2

9.1 Boolsche Variablen

Wir haben gelernt, wie Integer-Variablen Zahlen speichern. Im Gegensatz dazu können boolesche Variablen nur wahr "true" oder falsch "false" sein. Ein Licht kann nur ein- oder ausgeschaltet sein. Diesen Zustand in einer Integer Variable zu speichern, macht nicht viel Sinn, aber mit einer Booleschen Variable ist es wahr, für den Ein-Zustand und false für den Aus-Zustand. Wir haben bereits diese Variablen verwendet, um digitale Pins HIGH und LOW zu setzten, mit LED.Write (true);

Um den Zusatnd eines Tasters in einer Variable zu speichern.

```
bool button_state;
button_state = Button.Read();
```

Wir verwendeten auch while-Schleifen als Endlosschleife, wenn wir "true' als Anweisung verwenden.

```
while (true)
{
    //code here
}
```

Nehmen Sie den letzten Code, den wir verwendet haben und ändern Sie ihn dahingehend eine Boolesche Variable zu verwenden, so ist er leichter zu lesen. Statt der Übergabe des Tasten Zustands direkt an den LED-Zustand weiterzugeben, lesen wir den Status des Tasters in button_state und dann übergeben wir den button_state, um die LED anzusprechen.

```
using System;
using Microsoft.SPOT;
using System.Threading;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.FEZ;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            OutputPort LED;
            InputPort Button;
            bool button_state;
            LED = new OutputPort((Cpu.Pin)FEZ_Pin.Digital.LED, true);

            Button = new InputPort((Cpu.Pin)FEZ_Pin.Digital.LDR, false,
            Port.ResistorMode.PullUp);

            while (true)
            {
                button_state = Button.Read();
                LED.Write(button_state);
                Thread.Sleep(10);
            }
        }
    }
}
```

Können Sie die LED blinken lassen, solange die Taste gedrückt wird?

```
using System;
using Microsoft.SPOT;
using System.Threading;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.FEZ;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            OutputPort LED;
            InputPort Button;

            LED = new OutputPort((Cpu.Pin)FEZ_Pin.Digital.LED, true);
            Button = new InputPort((Cpu.Pin)FEZ_Pin.Digital.LDR, false,
                                    Port.ResistorMode.PullUp);
            while (true)
            {
                while (Button.Read() == false)//Button is false when pressed
                {
                    LED.Write(true);
                    Thread.Sleep(300);
                    LED.Write(false);
                    Thread.Sleep(300);
                }
            }
        }
    }
}
```

Wichtiger Hinweis: Die == werden verwendet, um auf Gleichheit in C# zu überprüfen. Dies unterscheidet sich von = mit dem Werte zugewiesen werden.

9.2 if-Anweisung

Ein wichtiger Teil der Programmierung ist die Überprüfung von Zuständen und entsprechende Maßnahmen zu ergreifen. Zum Beispiel, "wenn" die Temperatur über 80° steigt, schalten Sie den Lüfter ein. Um die if-Anweisung mit unserem einfachen Beispiel auszuprobieren, wollen wir die LED anschalten "wenn" die Taste gedrückt wird. Hinweis: Dies ist das Gegenteil von dem, was wir vorher hatten. Da bei unserer Anwendung der Taster LOW ist, wenn er gedrückt wird. Um dieses zu erreichen, müssen wir den Zustand der Taste für die LED umkehren. Wenn die Taste gedrückt wird (low), dann wollen wir wiederum die LED auf (hoch) setzen. Dies muß immer wieder überprüft werden, wir werden es einmal alle 10ms tun.

```
using System;
using Microsoft.SPOT;
using System.Threading;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.FEZ;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            OutputPort LED;
            InputPort Button;
            bool button_state;
            LED = new OutputPort((Cpu.Pin)FEZ_Pin.Digital.LED, true);
            Button = new InputPort((Cpu.Pin)FEZ_Pin.Digital.LDR, false,
                                   Port.ResistorMode.PullUp);
            while (true)
            {
                button_state = Button.Read();

                if (button_state == true)
                {
                    LED.Write(false);
                }

                if (button_state == false)
                {
                    LED.Write(true);
                }

                Thread.Sleep(10);
            }
        }
    }
}
```

9.3 if-else-Anweisung

Wir haben gelernt, wie die if-Anweisung funktioniert. Jetzt wollen wir die else-Anweisung verwenden. Einfach gesagt, "wenn" eine Aussage wahr ist, wird der Code innerhalb der if-Anweisung ausgeführt ansonsten "else" wird der Code in der else-Anweisung ausgeführt. Mit dieser neuen Erkenntnis können wir den obigen Code optimieren.

```
using System;
using Microsoft.SPOT;
using System.Threading;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.FEZ;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            OutputPort LED;
            InputPort Button;
            bool button_state;
            LED = new OutputPort((Cpu.Pin)FEZ_Pin.Digital.LED, true);
            Button = new InputPort((Cpu.Pin)FEZ_Pin.Digital.LDR, false,
                                   Port.ResistorMode.PullUp);
            while (true)
            {
                button_state = Button.Read();
                if (button_state == true)
                {
                    LED.Write(false);
                }
                else
                {
                    LED.Write(true);
                }

                Thread.Sleep(10);
            }
        }
    }
}
```

Ich werde Euch ein Geheimnis verraten! Wir benutzen die if-Anweisung und else-Anweisung in diesem Beispiel nur zu Demonstrationszwecken. Wir können den Code auch auf diese Weise schreiben.

```
using System;
using Microsoft.SPOT;
using System.Threading;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.FEZ;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            OutputPort LED;
            InputPort Button;

            LED = new OutputPort((Cpu.Pin)FEZ_Pin.Digital.LED, true);
            Button = new InputPort((Cpu.Pin)FEZ_Pin.Digital.LDR, false,
                                   Port.ResistorMode.PullUp);
            while (true)
            {
                LED.Write(Button.Read() == false);
                Thread.Sleep(10);
            }
        }
    }
}
```

Oder auf diesem Weg!

```
using System;
using Microsoft.SPOT;
using System.Threading;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.FEZ;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            OutputPort LED;
            InputPort Button;

            LED = new OutputPort((Cpu.Pin)FEZ_Pin.Digital.LED, true);
            Button = new InputPort((Cpu.Pin)FEZ_Pin.Digital.LDR, false,
                                   Port.ResistorMode.PullUp);
            while (true)
            {
                LED.Write(!Button.Read());
                Thread.Sleep(10);
            }
        }
    }
}
```

Normalerweise gibt es viele Wege, um den Code zu schreiben. Verwenden Sie, was für Sie am bequemsten ist und mit mehr Erfahrung, lernen Sie, wie Sie den Code optimieren können.

9.4 Methoden and Argumente

Methoden sind Aktionen die ein Objekt betreffen. Man könnte sie auch als Funktionen eines Objektes bezeichnen. Wir haben Methoden bereits kennegelernt und haben sie verwendet. Erinnern Sie sich an die Print-Methode des Debug-Objekts? Wir haben schon viele Male zuvor Debug.Print benutzt, dabei übergaben wir der Methode einen "string" zum anzeigen im Ausgabe Fenster. Der übergebene "String" wird Argument genannt.

Methoden können eine oder mehrere optionale Argumente übernehmen, aber sie können nur einer optionalen Wert zurückgeben.

Die Print-Methode im Debug-Objekt übernimmt nur ein String-Argument. Bei anderen Methoden kann kein Argument erforderlich sein oder mehr als ein Argument. Zum Beispiel kann eine Methode, die einen Kreis zeichnen soll, vier Argumente übernehmen, DrawCircle (posx, posy, diam, color). Ein Beispiel für Rückgabe-Werte könnte eine Methode sein, die die Temperatur zurückgibt.

Bisher haben wir drei Variablen-Typen kennengelernt, int, string und bool. Wir werden andere Variablen-Typen später erläutern, aber alles über das wir hier reden trifft auch auf andere Variablen-Typen zu.

Der optionale Rückgabe-Wert kann jeder Variablen-Typ sein. Wenn kein Wert zurückgegeben wird, ersetzen wir die Variable mit dem Typ "void". Das "return" Schlüsselwort wird verwendet, um Werte am Ende einer Methode zurückgeben.

Hier ist eine sehr einfache Methode, die die Summe von zwei ganzen Zahlen zurückgibt.

```
int Add(int var1, int var2)
{
    int var3;
    var3 = var1 + var2;
    return var3;
}
```

Die Methode beginnt mit dem Rückgabewert Typ "int" gefolgt vom Namen der Methode. Dann folgt die Liste der Argumente. Die Argumente werden immer durch Klammern gruppiert und durch Komma getrennt.

Innerhalb der Add-Methode wird eine lokale Integer-Variable var3 deklariert. Lokale Variablen werden innerhalb einer Methode erstellt und sterben, wenn die Methode beenden wird. Danach addieren wir unseren beiden Variablen und geben schließlich das Ergebnis zurück.

Was, wenn wir eine Zeichenfolge zurückgeben wollen, der die Summe der beiden Zahlen darstellt? Beachten Sie, dass eine Zeichenfolge mit der Nummer 123 nicht das gleiche ist wie eine ganze Zahl mit 123. Ein Integer ist eine Zahl, aber ein String ist ein Array von Zeichen, die Text oder Zahlen repräsentieren können. Für uns Menschen ist das dasselbe, aber in der Computer-Welt ist dies ganz anders.

Hier ist der Programm Code der einen String zurückgibt.

```
string Add(int var1, int var2)
{
    int var3;
    var3 = var1 + var2;

    string MyString;
    MyString = var3.ToString();

    return MyString;
}
```

Sie können sehen, wie der zurückgegebene Typ in eine Zeichenfolge geändert wurde. Wir konnten nicht var3 zurückgeben, weil es eine Integer-Variable ist, deshalb mussten wir sie in einen String konvertieren. Um dies zu erreichen, erzeugen wir eine neue Objekt Variable mit dem Namen MyString. Dann konvertieren wir var3 in eine Zeichenfolge mit "ToString" und fügen die neue Zeichenfolge in MyString ein.

Die Frage ist nun, wie wir eine "ToString"-Methode auf eine Variable vom Typ Integer anwenden? In Wirklichkeit ist alles in C# ein Objekt, selbst Variablen-Typen. Dieses Buch wird nicht auf diese Details eingehen, es dient Ihnen nur zum Einstieg.

Dies wird t alles in mehreren Schritten durchgeführt, um Ihnen zu zeigen, wie es gemacht wird, aber wir können alles kompakter schreiben und die Ergebnisse werden das gleiche sein.

```
string Add(int var1, int var2)
{
    return (var1+var2).ToString();
}
```

Ich empfehle Ihnen nicht den Code schreiben, der extrem kompakt ist, wie im Beispiel oben, bis Sie sehr vertraut mit der Programmiersprache sind. Selbst dann sollte es Grenzen geben, wie kompakt Sie den Code schreiben. Sie sollten immer noch in der Lage sein, den Code nach einiger Zeit zu überarbeiten, und auch jemand anderes sollte den Code lesen und verstehen können.

9.5 Klassen

Alle Objekte über die wir bereits gesprochen haben, sind C# "Klassen". In modernen objektorientierten Programmiersprachen, ist alles ein Objekt und Methoden gehören immer zu einem Objekt. Dies ermöglicht es Methoden mit dem gleichen Namen zu haben, aber sie können zu ganz anderen Objekte gehören. Ein "Mensch" kann "gehen" und auch eine "Katze" kann "gehen", aber gehen sie auf die selbe Weise? Beim Aufruf der "gehen"-Methode in C# dann ist es nicht klar, ob die Katze oder der Mensch geht, sondern erst human.walk oder cat.walk macht es klarer.

Erstellen von Klassen würde den Rahmen dieses Buches sprengen. Hier ist eine sehr einfache Klasse, um Ihnen den Einstieg zu erleichtern.

```
class MyClass
{
    int Add(int a, int b)
    {
        return a + b;
    }
}
```

9.6 Public vs. Private

Methoden können privat zu einer Klasse gehören oder öffentlich zugänglich sein. Dies ist dazu gut, um die Objekte aus robuster zu machen vor dem Missbrauch durch Programmierer. Wenn Sie ein Objekt (Klasse) erstellen und dieses Objekt über Methoden verfügt, die nicht jedermann benutzen soll, dann fügen Sie das Stichwort "privat" vor den Rückgabetyp der Methode, andernfalls verwenden Sie das "public" Schlüsselwort.

Hier ist ein kurzes Beispiel

```
class MyClass
{
    public int Add(int a, int b)
    {
        // the object can use private methods
        // inside the class only
        DoSomething();
        return a + b;
    }
    private void DoSomething()
    {
    }
}
```

9.7 Static vs. non-static

Some objects in life have multiple instances but others only exist once. The objects with multiple instances are non-static. For example, an object representing a human doesn't mean much. You will need an ?instance? of this object to represent one human. So this will be something like

```
human Mike;
```

We now have a ?reference? called Mike of type human. It is important to note that this reference is at this point not referencing to any object (no instance assigned) just yet, so it is referencing NULL.

To create the ?new? object instance and reference it from Mike

```
Mike = new human();
```

We now can use any of the human methods on our ?instance? Mike

```
Mike.Run(distance);
Mike.Eat();
bool hungry = Mike.IsHungry();
```

We have used those non-static methods already when we controlled input and output pins.

When creating a new non-static object, the ?new? keyword is used with the ?constructor? of the object. The constructor is a special type of method that returns no vale and it is only used when creating (construction) new objects.

Static methods are easier to handle because there is only one object that is used directly without creating instances. The easiest example is our Debug object. There is only one debug object in the NETMF system so using its methods, like Print, is used directly.

```
Debug.Print(?string?);
```

I may not have used the exact definitions of static and instances but I wanted to describe it in the simplest possible way.

9.8 Constants

Some variables may have fixed values that never change. What if you accidentally change the value of this variable in your code? To protect it from changing, add the ?const? keyword before the variable declaration.

```
const int hours_in_one_day = 24;
```

9.9 Enumeration

An Enumeration is very similar to constant. Let's say we have a device that accepts four commands, those are MOVE, STOP, LEFT, RIGHT. This device is not a human and so these commands are actually numbers. We can create constants (variables) for those four commands like the following

```
const int MOVE = 1;
const int STOP = 2;
const int RIGHT = 3;
const int LEFT = 4;
//now we can send  a command...
SendCommand(MOVE);
SendCommand(STOP);
```

The names are all upper case because this is how programmers usually name constants. Any other programmer seeing an upper case variable would know that this is a constant.

The code above is okay and will work but it will be nicer if we can group those commands

```
enum Command
{
    MOVE = 1,
    STOP = 2,
    RIGHT = 3,
    LEFT = 4,
}
//now we can send  a command...
SendCommand(Command.LEFT);
SendCommand(Command.STOP);
```

With this new approach, there is no need to remember what commands exist and are the command numbers. Once the word ?Command? is typed in, Visual Studio will give you a list of available commands.

C# is also smart enough to increment the numbers for enumerations so the code can be like this listing and will work exactly the same way

```
enum Command
{
    MOVE = 1,
    STOP ,
    RIGHT,
    LEFT ,
}
//now we can send a command...
SendCommand(Command.LEFT);
SendCommand(Command.STOP);
```

10 Assembly/Firmware Matching

NETMF devices usually include extended features. Those extended features require an extra assembly/library to be added to the C# projects so a user can make use of the new features. For example, NETMF doesn't support Analog pins but FEZ and other hardware from GHI does support analog pins. To use the analog pins, you need to add an assembly/library provided by GHI then you have new classes/objects that can be used to access those new features. Important Note: The firmware will fail to run if the version of the assembly/library that used in the project does not match the version of the firmware.

This is very common issue that users run into when updating the firmware where the application just stop working and debugging seem to fail. Here is what happens:

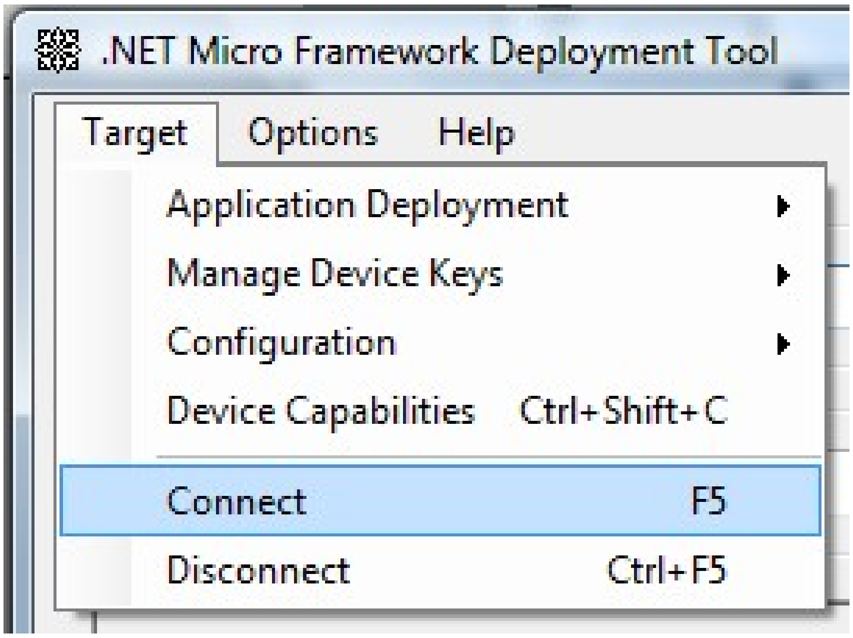
Scenario #1: A developer had received a new device. This device happens to have firmware version 1.0 on it. Then the developer went to the website and downloaded the latest SDK. The SDK had firmware version 1.1 in it. When trying to upload a project, VS2010 will fail to attach to the device with no indication why! The developer will now think the new device is not functioning, actually, the device is just fine but, in this example, the firmware is version 1.0 and the assembly is version 1.1 so the system will refuse to run. To fix the issue, update the firmware on the device to match the firmware in the SDK.

Scenario #2: A developer has a perfectly working system that, for example, uses firmware version 2.1. Then a new SDK comes out with firmware version 2.2, so the developer installs the new SDK on the PC then uploads the new firmware to the device (FEZ). When rebooting the device, it stops working because the new loaded firmware is version 2.2 but the user application still uses assembly version 2.1. To fix this issue, open the project that has the user application and remove any device-specific assemblies. After they are removed, go back and add them back. With this move the new files will be fetched from the new SDK.

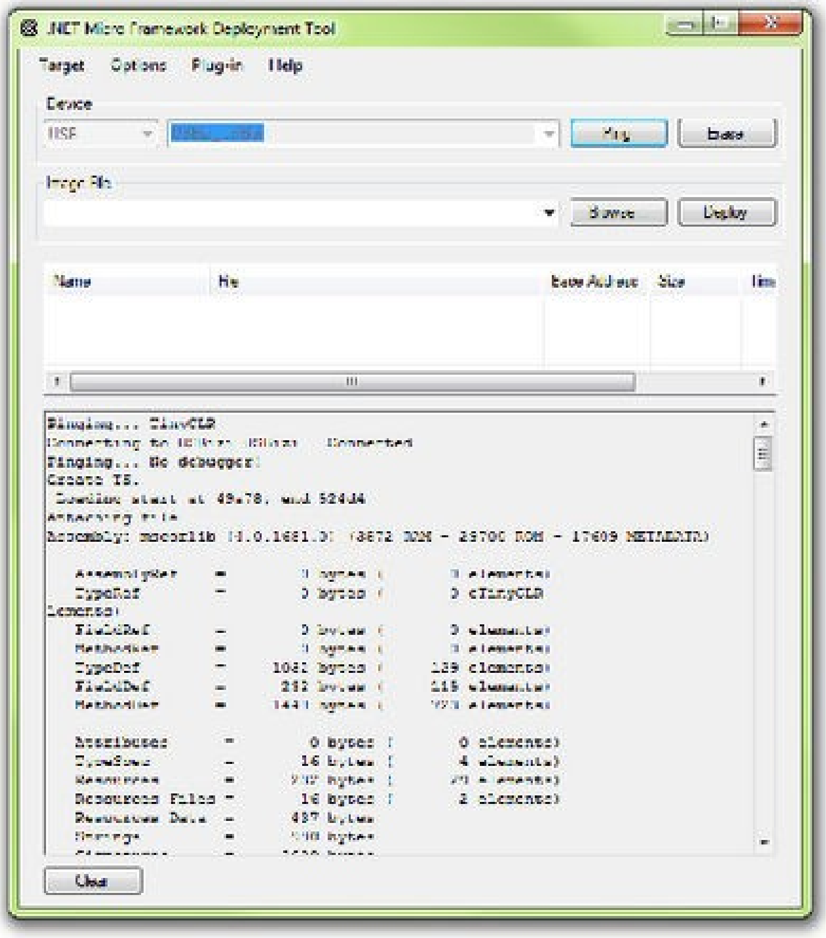
10.1 Boot-up Messages

We can easily see why the system is not running using MFDeploy. NETMF outputs many useful messages on power up. Should the system become unresponsive, fails to run or for any other debug purposes, we can use MFDeploy to display these boot up messages. Also, all ?Debug.Print? messages that we usually see on the output window are visible on MFDeploy.

To display the boot up messages, click on ?Target->Connect? from the menu then reset the device. Right after you reset the device in one second, click on ?ping?. MFDeploy will freeze for a second then display a long list of messages.



Note that on FEZ Domino the reset button is available on the board. For FEZ Mini, you need to connect a switch to the reset pin or if you have FEZ Mini in the starter kit or robot kit then you can use the reset button on the board.



```
Pinging... TinyCLR
Connecting to [illegible]    Connected
Pinging... No debugger!
Create TS.
 Loading start at 49a78, end 524d4
Attaching file
Assembly: mscorlib (4.0.1681.0) (3672 RAM - 29700 ROM - 17609 METADATA)

   AssemblyRef     =        0 bytes (       0 elements)
   TypeRef         =        0 bytes (       0 eTinyCLR
lements)
   FieldRef        =        0 bytes (       0 elements)
   MethodRef       =        0 bytes (       0 elements)
   TypeDef         =     1032 bytes (     129 elements)
   FieldDef        =      232 bytes (     115 elements)
   MethodDef       =     1443 bytes (     723 elements)

   Attributes       =        0 bytes (       0 elements)
   TypeSpec         =       16 bytes (       4 elements)
   Resources        =      232 bytes (      29 elements)
   Resources Files  =       16 bytes (       2 elements)
   Resources Data   =      437 bytes
   Strings          =      [illegible] bytes
```

11 Pulse Width Modulation

PWM is a way of controlling how much power is provided to a device. Dimming a LED or slowing down a motor is best done using a PWM signal. If we apply power to a LED it comes completely on and if we shut off the power then it is completely off. Now, what if we turn the LED on for 1ms and then off for 1ms? In reality it is blinking on and off very fast but our eyes would never see it blinking that fast causing our eyes to see the LED to be dimmer than before. We calculate this using (on/(off+on))x100= 50%. So, the LED is only getting half the energy.

So if we turn the LED on for 1ms and off for 9ms we will end up with (1/(1+9))x100=10% energy, so the LED is very dimmed.

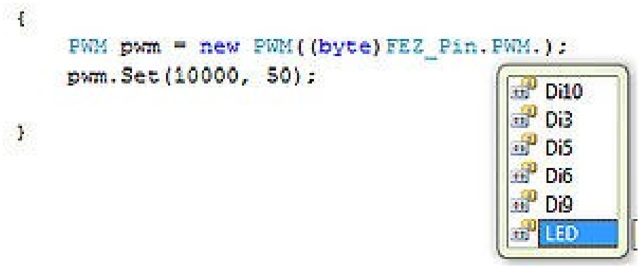
PWM is very simple to generate but if we are going to toggle a pin few hundred or thousand times a second then we are wasting a lot of processor power. There are devices that generate PWM signals more efficiently. Also, many processors include a built-in circuitry needed to generate PWM in hardware. This means, we can setup the PWM hardware to generate a PWM signal and then it is done automatically without the need for processor interaction.

FEZ included few pins that can serve as PWM pins. The FEZ library includes everything needed to activate PWM on any one of those pins.

Here is an example that creates a PWM object that sets the PWM to 10 Khz (10,000 clocks/sec) and the duty cycle to 50 (50% energy)

```
PWM pwm = new PWM((PWM.Pin) FEZ_Pin.PWM.LED);
pwm.Set(10000, 50);
```

FEZ includes an enumeration of what pins have PWM hardware. Using the PWM enumeration, we can easily find the PWM pins on your device. Visual studio will give you a list of them while you are typing your code, just like this image.



```
{
    PWM pwm = new PWM((byte)FEZ_Pin.PWM.);
    pwm.Set(10000, 50);

}
```

From the list above, you can see that the LED is connected to a PWM signal. Instead of blinking LED, what about we fade it in and out? This will make it look so much cooler! Here is the code.

```
using System;
using Microsoft.SPOT;
using System.Threading;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.Hardware;
using GHIElectronics.NETMF.FEZ;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            sbyte dirr = 1;
            byte duty = 40;
            PWM pwm = new PWM((PWM.Pin)FEZ_Pin.PWM.LED);
            while (true)
            {
                pwm.Set(10000, duty);
                duty = (byte)(duty + dirr);
                if (duty > 90 || duty < 10)
                {
                    dirr *= -1;
                }

                Thread.Sleep(10);
            }

        }
    }
}
```

The code will loop while incrementing the energy of the LED to 90% and then goes back and decrement it to 10% and so on.

Important: PWM will be perfect to control how fast your robot will move or how fast a fan is spinning. A pin with PWM signal can power an LED just fine. This is not the case if we are controlling something with high power demands, like a motor or a light bulb. In this case, we would need to use the PWM signal to control a circuitry and that circuitry will supply the needed power to the device. For example, H-bridge circuits are most commonly used to control the speed and direction of motors.

Important note: All hardware PWM signals share the same clock. Changing the frequency on one will change it on other channels, but duty cycle is independent.

11.1 Simulating PWM

PWM pins are controlled internally inside the processor using a special PWM hardware. This means, the hardware internally will toggle the pin. The processor only needs to set some registers and then no processor interaction is needed at all to generate the PWM signal.

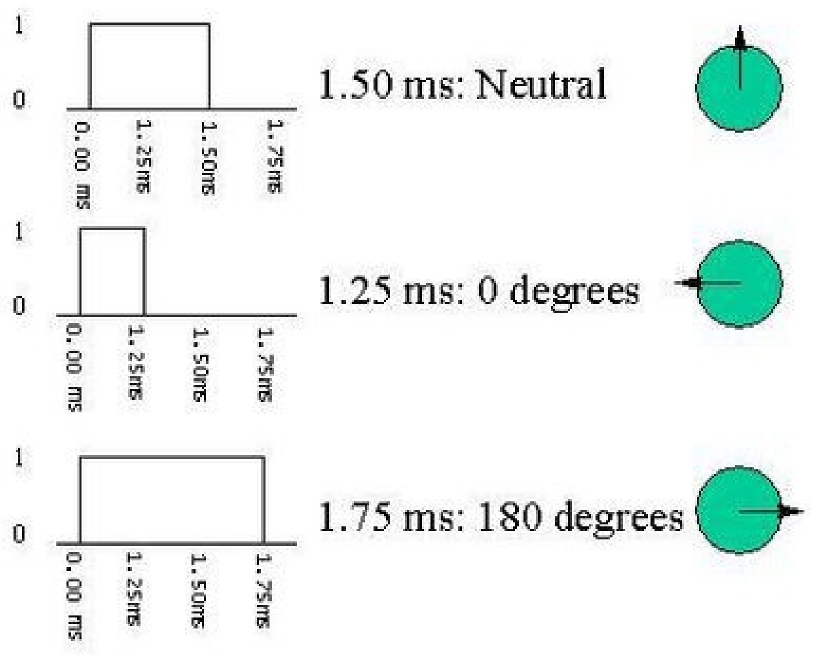
GHI's NETMF devices provide an OutputCompare class. Through this class a user can generate about any signal, including PWM. Note that this is done in software so it is better to use the PWM class is possible. Why? When you use the PWM class then the signal for PWM is generated using hardware and the processor doesn't need to do anything. On the other hand, when you use OutputCompare to generate PWM then the processor need to keep track of time and toggle the PWM pin at specific time constants.

The servo motor example provided on www.TinyCLR.com uses OutputCompare. The advantage of this is that you can use any pin to control a servo. It is better to use PWM to control the servo but then you will only be able to use the PWM specific pins.

11.2 Servo Motors and Pulse Control

We have learned how to control the amount of energy using PWM. There are other good uses for PWM, such as controlling servos. The PWM class provides two methods Set and SetPulse methods. Earlier, we utilized PWM. Set which very well suited for setting energy levels. For controlling servos, using SetPulse is more suitable.

So how do servos work? This is best explained in this image I found on www.pc-control.co.uk



You can control the position of a servo by providing it with a pulse (see image above). If the pulse width is about 1.25ms then the servo is at 0 degrees. Increasing the pulse to 1.25ms will move the servo to 90 degrees (neutral). A wider pulse of 1.75ms will move the servo to 180 degrees. Okay, those is easy to generate using PWM but are still missing one piece of info. The image tells us how wide (in time) is the ?high pulse? but what about the ?low time?? Servos expect a pulse every 20ms to 30ms but this is not very critical to servos. So, we now have everything we need. Before I proceed, I want to explain what the ?period? is. A period of a frequency (signal) is the high time plus the low time. Now we are ready for some code.

The method PWM.SetPulse needs the high time and the period of the signal. Since our servo low time is not critical, can be 20ms to 30ms then I will just use a period of 20ms. This will result in a low time that will vary slightly but again, this is not important to servos. What is important is the high pulse which must be between 1.25ms to 1.75ms.

One last thing and we are done! SetPulse accepts value in nanoseconds but we have value in milliseconds. One millisecond is the equivalent of 1,000,000 (a million) nanoseconds.

Finally, here is the example.

```
using System;
using System.Threading;
using Microsoft.SPOT;
using GHIElectronics.NETMF.Hardware;
using GHIElectronics.NETMF.FEZ;
class Program
{
    public static void Main()
    {
        PWM servo = new PWM((PWM.Pin)FEZ_Pin.PWM.Di5);
        while (true)
        {
            // 0 degrees. 20ms period and 1.25ms high pulse
            servo.SetPulse(20 * 1000 * 1000, 1250 * 1000);
            Thread.Sleep(1000);//wait for a second

            // 90 degrees. 20ms period and 1.50ms high pulse
            servo.SetPulse(20 * 1000 * 1000, 1500 * 1000);
            Thread.Sleep(1000);//wait for a second

            // 180 degrees. 20ms period and 1.75ms high pulse
            servo.SetPulse(20 * 1000 * 1000, 1750 * 1000);
            Thread.Sleep(1000);//wait for a second
        }
        Thread.Sleep(-1);
    }
}
```

12 Piezo

Piezoelectric devices are a low cost way of adding tones and beeps to your design. They generate sounds one supplied with a frequency. PWM hardware is perfect for generating frequencies and so they can be perfect to activate a Piezo. We will always use duty cycle of 50% but we will change the frequency to get different tones.

When connecting a Piezo, make sure that it its polarity is correct, plus goes to PWM signal and minus goes to ground. See the FEZ Piezo speaker component on www.TinyCLR.com This is a project that decodes a MIDI file from a SD card then plays the main tone on a piezo.

http://www.microframeworkprojects.com/index.php?title=PWM_MIDI_Player

13 Glitch filter

When we used the button earlier (section 9.2), we set it up so when it is pressed, the pin value is low; otherwise, it is high. When you flip a witch or press a button, the button or switch can bounce when it is pressed. In other words, the button generates few on and off before it is settled. This will make it look like if the button was pressed few times. Those few on/off bounces come for a very short time. To eliminate them in hardware, we can add a capacitor between the pin and ground. To handle this in software, we check the time between button presses, if it is a short time then this is a ?glitch? and so it needs to be ignored. A user can press the button on and off maybe every 200 ms. So, if the time between presses is 10ms then we know the user couldn't have possible pressed it so fast, and for that, this is a glitch and we need to ?glitch filter? it.

Luckily, NETMF includes a glitch filter built internally so we never need to worry about this. When we enable an input pin, we have the option of enabling the glitch filter. The second argument when creating an ?InputPort? is a boolean indicating if the glitch filter to be enabled or not. For using switches, you probably would want this to always be true

```
Button = new InputPort((Cpu.Pin)FEZ_Pin.Digital.LDR, true, Port.ResistorMode.PullUp);
```

The glitch filter time can be changed as follows

```
TimeSpan ts = new TimeSpan(0, 0, 0, 0, 200);//200ms
Microsoft.SPOT.Hardware.Cpu.GlitchFilterTime = ts;
```

Important note: Glitch filter only works on interrupt-capable pins. If you try to use InputPort with glitch filter is set to true and you see an exception then most likely you are using a pin that is not interrupt capable.

14 Analog input & output

Analog pins are usually multiplexed with digital pins. Some of the processor pins can be digital and analog as well but not both at the same time.

14.1 Analog Inputs

Digital input pins can only read high and low (one or zero) but analog pins can read the voltage level. There are limitations on voltages that can be applied to analog inputs. For example, FEZ analog input can read voltages anywhere between zero and 3.3V. When the pins are digital they are tolerant of 5V but when the same pin is set to analog only up to 3.3V can be used. This limitation of analog inputs is not a big issue usually because most analog signals are conditioned to work with the analog inputs. A voltage divider or an op-amp circuit can be used to get a fraction of the actual signal, scale the signal. For example, if we want to measure the battery voltage, which is 6V, then we need to divide the voltage in half, using a voltage divider resistor(s), so the analog pin will only see half the voltage, which is 3V max. In software, we know we have the voltage divided in half so any voltage we see will need to be multiplied by 2 to give us the actual voltage we are trying to measure.

Luckily, GHI implementation of analog inputs already handles signal scaling. When constructing a new analog input object, optionally, you can set the scaling.

The internal reference is 0V to 3.3V so everything you measure needs to take this in mind. The easiest is to set the scale to 0 to 3300. You can think of this as millivolts. If we read 1000 then the input voltage is 1 volt.

```
AnalogIn BatteryVoltage = new AnalogIn((AnalogIn.Pin)FEZ_Pin.AnalogIn.An0);
BatteryVoltage.SetLinearScale(0, 3300);
int voltage = BatteryVoltage.Read();
```

Remember to use the analog pin enumeration to determine what pins can be used as analog. To print the analog value to the debug output in volts, we would need to convert the value to volts and then to a string.

```
AnalogIn BatteryVoltage = new AnalogIn((AnalogIn.Pin) FEZ_Pin.AnalogIn.An0);
BatteryVoltage.SetLinearScale(0, 3300);
int voltage = BatteryVoltage.Read();
Debug.Print("Voltage = " + (voltage / 1000).ToString() + "." + (voltage % 1000).ToString());
```

We divide by 1000 to get the voltage and then we use the modules to get the fraction.

14.2 Analog Outputs

Analog outputs are like digital outputs where they have limits on how much power they can provide. Analog outputs are even weaker than digital outputs. They are only capable of providing a very little signal to drive the power circuit, maybe drive a power amplifier.

Digital outputs can only be high or low but analog input can be set to any voltage between 0 and 3.3V. GHI implementation for analog outputs allows for automated scaling. You can give it a min, max and the actual value. An easy test would be to set the min to zero and max to 330V (3.3Vx100) then the value to 100 (1Vx100). This will give us 1V on the pin. We will connect this pin to another analog input to measure the value to verify it is 1V. It may not be exactly 1V but will be very close.

The analog output is multiplexed with analog input 3. When using the analog output, the analog input can' be used on this pin but we can use any other analog input on other pins.

```
using System;
using Microsoft.SPOT;
using System.Threading;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.Hardware;
using GHIElectronics.NETMF.FEZ;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            AnalogOut VoltLevel =
               new AnalogOut((AnalogOut.Pin)FEZ_Pin.AnalogOut.An3);
            VoltLevel.SetLinearScale(0, 3300);
            VoltLevel.Set(1000);

            AnalogIn PinVoltage =
               new AnalogIn((AnalogIn.Pin)FEZ_Pin.AnalogIn.An0);
            PinVoltage.SetLinearScale(0, 3300);

            while (true)
            {
                int voltage = PinVoltage.Read();
                Debug.Print("Voltage = " + (voltage / 1000).ToString() +
                              "." + (voltage % 1000).ToString());

                Thread.Sleep(200);
            }
        }
    }
}
```

Connect a wire between An0 and AOUT (An3) and then run this program. Note if no wire is connected then the analog pin is floating and can be of any value. Try touching AIN0 and see how the numbers change then connect to AOUT to see if you get back 1V. If you have a volt meter, connect it to AOUT to verify we have 1V. Note that it is not going to be exactly 1V but very close.

14.2.1 Audio Playback

There are many way to generate the sound using FEZ devices, for example we can use multi-channel PWM to generate audio, or add an extra device like Piezo by adjusting frequency and duration to produce the audio tones.

```
...
FEZ_Components.Piezo myPiezo =
                 new FEZ_Components.Piezo(FEZ_Pin.PWM.Di9);
                ...
                myPiezo.Play(tones[toneVal], duration);
...
```

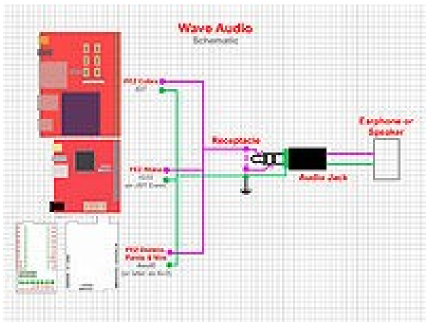
But could we use an analog output to generate the audio tone or sound.

Fortunately, GHI devices can play audio on the analog output.

How much does it cost you to add audio playback to FEZ? Technically, It costs you a piece of wire!

But of course, you probably would need a speaker, an earphone for MP3 or iPod should work fine or you could use the computer speaker as well. And a receptacle that matches the speaker jack.

Here is the Schematic:



That is all you need to play wav audio files. Now, this is not meant to play music but it is ideal if your product/project will need to voice out some commands or alerts. Maybe dial a number or playback a message ... You can probably think of a million ways to use this.

Here's the project page: http://www.microframeworkprojects.com/index.php?title=FEZ_Audio_OUT

15 Garbage Collector

When programming in older languages like C or C++, programmers had to keep track of objects and release them when necessary. If an object is created and not released then this object is using resources from the system that will never be freed. The most common symptom is memory leaks. A program that is leaking memory will contentiously use more memory till the system run out of memory and probably crash. Those bugs are usually very difficult to find in code.

Modern languages have garbage collector that keeps track of used objects. When the system runs low on memory resources, the garbage collector jumps in and search through all objects and frees the one with no ?references?. Do you remember how we created objects before using the ?new? keyword and then we assigned the object to a ?reference?? An object can have multiple references and the garbage collector will not remove the object till it has zero references.

```
// new object
OutputPort Ref1 = new OutputPort(FEZ_Pin.Digital.LED, true);
// second reference for same object
OutputPort Ref2 = Ref1;
// lose the first reference
Ref1 = null;
// Our object is still referenced
// it will not be removed yet
// now remove the second reference
Ref2 = null;
// from this point on, the object is ready to be
// removed by the garbage collector
```

Note that the object is not removed immediately. When needed, the Garbage collector will run and remove the object. This can be an issue in some rare cases because the garbage collector needs some time to search and remove objects. It will only be few milliseconds but what if your application can't afford that? If so, the garbage collector can be forced to run at a desired anytime.

```
//force the garbage collector
Debug.GC(true);
```

Losing Resources The garbage collector ease object allocation but it can also cause problems if we are not careful. A good example will be on using digital output pins. Lets say we need a pin to be high. We create an OuputPort object and set the pin high. Later on we lose the ?reference? for that object for some reason. The pin will still be high when the reference is lost so all is good so far. After few minutes, the garbage collector kicks in and it finds this unreferenced object, so it will be removed. Freeing an OutputPort will case the pin to change its state to input. Now, the pin is not high anymore!

```
// Turn the LED on
OutputPort LED = new OutputPort((Cpu.Pin)FEZ_Pin.Digital.LED, true);
LED = null;
// we have lost the reference but the LED is still lit
//force the garbage collector
Debug.GC(true);
// The LED is now off!
```

An important thing to note is that if we make a reference for an object inside a method and the method returns then we have already lost the reference. Here is an example

```
using System;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;

namespace Test
{
    public class Program
    {
        static void TurnLEDOn()
        {
            // Turn the LED on
            OutputPort LED =
               new OutputPort((Cpu.Pin)FEZ_Pin.Digital.LED, true);
        }
        public static void Main()
        {
            TurnLEDOn();
            // we think that everythign is okay but it is not
            // run the GC
            Debug.GC(true);
            // is LED still on?
        }

    }
}
```

To solve this, we need a reference that is always available. Here is the correct code

```
using System;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;

namespace Test
{
    public class Program
    {
        static OutputPort LED;
        static void TurnLEDOn()
        {
            // Turn the LED on
            LED = new OutputPort((Cpu.Pin)FEZ_Pin.Digital.LED, true);
        }
        public static void Main()
        {
            TurnLEDOn();
            // run the GC
            Debug.GC(true);
            // is the LED on?
        }

    }
}
```

Another good example is using timers. NETMF provide a way to create timers that handle some work after a determined time. If the reference for the timer is lost and the garbage collector had run, the timer is now lost and it will not run as expected. Timers are explained in later chapter.

Important note: If you have a program that is working fine but then right after you see the GC running in the ?Output Window? the program stops working or raises an exception then this is because the GC had removed an object that you need. Again, that is because you didn't keep references to the object you want to keep alive.

15.1 Dispose

The garbage collector will free objects at some point but if we need to free one particular object immediately? Most objects have a Dispose method. If an object needs to be freed at anytime, we can ?dispose? it.

Disposing object is very important in NETMF. When we create a new InputPort object, the assigned pin is reserved. What if we want to use the same pin as an output? Or even use the same pin as an analog input? We will first need to free the pin and then create the new object.

```
OutputPort OutPin = new OutputPort((Cpu.Pin)FEZ_Pin.Digital.Di5, true);
OutPin.Dispose();

InputPort InPort = new InputPort((Cpu.Pin)FEZ_Pin.Digital.Di5, true,
                        Port.ResistorMode.PullUp);
```

15.2 GC Output Messages

When the garbage collector runs, it outputs a lot of useful information to the output window. These messages give you an idea of what is using resources in the system. Although not recommended, you may want to disable those messages to free up the output window for your own usage. This is easily achievable using this line of code.

```
Debug.EnableGCMessages(false);
```

16 German C-Sharp Level3

This section will cover all C# materials we wanted to include in this book. A good and free eBook to continue learning about C# is available at http://www.programmersheaven.com/2/CSharpBook

16.1 Byte

We learned how int are useful to store numbers. They can store very large numbers but every int consumes four bytes of memory. You can think of a byte as a single memory cell. A byte can hold any value from 0 to 255. It doesn't sound like much but this is enough for a lot of things. In C# bytes are declared using ?byte? just like how we use ?int?.

```
byte b = 10;
byte bb = 1000;// this will not work!
```

The maximum number that a byte can hold is 256 [0..255], What?s going to happen if we increment it to over 255? Incrementing 255 by one would overlap the value back to zero.

You will probably want to use int for most of your variables but we will learn later where bytes are very important when we start using arrays.

16.2 Char

To represent a language like English, we need 26 values for lower case and 26 for upper case then 10 for numbers and maybe another 10 for symbols. Adding all these up will give us a number that is well less than 255. So a byte will work for us. If we create a table of letters, numbers and symbols, we can represent everything with a numerical value. Actually, this table already exists and it is called ASCII table.

So far a byte is sufficient to store all ?characters? we have in English. Modern computer systems have expanded to include other languages, some used very complex non-Latin characters. The new characters are called Unicode characters. Those new Unicode characters can be more than 255 and so a byte is not sufficient and an integer (four bytes) is too much. We need a type that uses 2-bytes of memory. 2-bytes are good to store numbers from 0 to over 64,000. This 2-bytes type is called ?short?, which we are not using in this book.

Systems can represent characters using 1-byte or using 2-bytes. Programmers decided to create a new type called char where char can be 1-byte or 2-bytes, depending on the system. Since NETMF is made for smaller systems, its char is actually a byte! This is not the case on a PC where a char is a 2-byte variable! Do not worry about all this mess, do not use char if you do not have to and if you use it, remember that it is 1-byte on NETMF.

16.3 Array

If we are reading an analog input 100 times and we want to pass the values to a method, it is not practical to pass 100 variables in 100 arguments. Instead, we create an ?array? of our variable type. You can create an array of any object. We will mainly be using byte arrays. When you start interfacing to devices or accessing files, you will always be using byte arrays.

Arrays are declared similar to objects.

```
byte[] MyArray;
```

The code above creates a ?reference? of an object of type ?byte array?. This is only a reference but it doesn't have any object yet, it is null. If you forgot what is a reference then go back and read more in C# Level 2 chapter.

To create the object we use ?new? keyword and then we need to tell it the size of our array. This size is the count of how many elements on the type we will have in a n array. Our type is a byte and so the number is how many bytes we are allocating in memory.

```
byte[] MyArray;
MyArray = new byte[10];
```

We have created a byte array with 10 elements in it. The array object is referenced from ?MyArray?.

We now can store/read any of the 10 values in the array by indicating which ?index? we want to access.

```
byte[] MyArray;
MyArray = new byte[10];
MyArray[0] = 123;// first index
MyArray[9] = 99;// last index
MyArray[10] = 1;// This is BAD...ERROR!!
```

A very important note here is that indexes start from zero. So, if we have an array of size 10 then we have indexes from 0 to 9. Accessing index 10 will NOT work and will raise an exception. We can assign values to elements in an array at the time of initialization. This example will store the numbers 1 to 10 in indexes 0 to 9.

```
byte[] MyArray = new byte[10] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
```

To copy an array, use the Array class as follows

```
byte[] MyArray1 = new byte[10] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
byte[] MyArray2 = new byte[10];
Array.Copy(MyArray1, MyArray2, 5);//copy 5 elements only
```

One important and useful property of an array is the Length property. We can use it to determine the length of an array.

```
byte[] MyArray1 = new byte[10] { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
byte[] MyArray2 = new byte[10];
Array.Copy(MyArray1, MyArray2, MyArray1.Length);//copy the whole array
```

16.4 String

We have already used strings in many places. We will review what we have learned and add more details.

Programs usually need to construct messages. Those messages can be a human readable text. Because this is useful and commonly used feature in programs, C# supports strings natively. C# knows if the text in a program is a string if it is enclosed by double-quotes.

This is a string example.

```
string MyString = "Some string";
string ExampleString = "string ExampleString";
```

Whatever inside the double quotes is colored in red and considered to be a string. Note how in the second line I purposely used the same text in the string to match what I used to assign the string. C# doesn't compile anything in quote (red text) but only take it as it is as a string. You may still have confusion on what is the difference between a integer variable that have 5 in it and a string that have 5 in it. Here is an example code

```
string MyString = "5" + "5";
int MyInteger = 5 + 5;
```

What do you think the actual value of the variables now? For integer, it is 10 as 5+5=10. But for string this is not true. Strings do not know anything about what is in it, text or numbers make no difference. When adding two strings together, a new string is constructed to combine both. And so ?5?+?5?=?55? and not 10 like integers.

Almost all objects have a ToString method that converts the object information to a printable text. This demonstration shows how ToString works

```
int MyInteger = 5 + 5;
string MyString = "The value of MyInteger is: " + MyInteger.ToString();
Debug.Print(MyString);
```

Running the above code will print

```
The value of MyInteger is: 10
```

Strings can be converted to byte arrays if desired. This is important if we want to use a method that only accepts bytes and we want to pass our string to it. If we do that, every character in the string will be converted to its equivalent byte value and stored in the array

```
using System.Text;
//.....
//.....
byte[] buffer = Encoding.UTF8.GetBytes("Example String");
```

16.5 For-Loop

Using the while-loop is enough to serve all our loop needs but for-loop can be easier to use in some cases. The simplest example is to write a program that counts from 1 to 10. Similarly, we can blink an LED 10 times as well. The for-loop takes three arguments on a variable. It needs the initial value, how to end the loop and what to do in every loop

```
int i;
for (i = 0; i < 10; i++)
{
    //do something
}
```

We first need to declare a variable to use. Then in the for-loop, we need to give it three arguments (initial, rule, action). In the very first loop, we asked it to set variable ?i? to zero. Then the loop will keep running as long as the variable ?i? is less then 10. Finally, the for-loop will increment variable i in every loop. Let us make a full program and test it.

```
using System.Threading;
using Microsoft.SPOT;
using System;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            int i;
            for (i = 0; i < 10; i++)
            {
                Debug.Print("i= " + i.ToString());
            }
        }

    }
}
```

If we run the program above, we will see that it is printing i from 0 to 9 but not 10. But, we wanted it to run from 1 to 10 and not 0 to 9! To start from 1 and not 0, we need to set i to 1 in the initial loop. Also, to run to 10, we need to tell the for-loop to turn all the way to 10 and not less than 10 so we will change the less than (?<?) with less than or equal (?<=?)

```
using System.Threading;
using Microsoft.SPOT;
using System;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            int i;
            for (i = 1; i <= 10; i++)
            {
                Debug.Print("i= " + i.ToString());
            }
        }

    }
}
```

Can we make the 'for' loop count only even numbers (increment by two)?

```
using System.Threading;
using Microsoft.SPOT;
using System;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            int i;
            for (i = 2; i <= 10; i = 1 + 2)
            {
                Debug.Print("i= " + i.ToString());
            }
        }
    }
}
```

The best way to understand for-loops is by stepping in code and seeing how C# will execute it.

16.6 Switch Statement

You will probably not use switch statement for beginner application but you will find it very useful when making large programs, especially when handling state-machines. The switch-statement will compare a variable to a list of constants (only constants) and make an appropriate jump accordingly. In this example, we will read the current ?DayOfWeek? value and then from its value we will print the day as a string. We can do all this using if-statement but you can see how much easier switch-statement is, in this case.

```
using System.Threading;
using Microsoft.SPOT;
using System;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            DateTime currentTime = DateTime.Now;
            int day = (int)currentTime.DayOfWeek;
            switch (day)
            {
                case 0:
                    Debug.Print("Sunday");
                    break;
                case 1:
                    Debug.Print("Monday");
                    break;
                case 2:
                    Debug.Print("Tuesday");
                    break;
                case 3:
                    Debug.Print("Wednsday");
                    break;
                case 4:
                    Debug.Print("Thursday");
                    break;
                case 5:
                    Debug.Print("Friday");
                    break;
                case 6:
                    Debug.Print("Saturday");
                    break;
                default:
                    Debug.Print("We should never see this");
                    break;
            }
        }
    }
}
```

One important note about switch-statement is that it compares a variable to a list of constants. After every ?case? we must have a constant and not a variable. We can also change the code to switch on the enumeration of days as the following.

```
using System.Threading;
using Microsoft.SPOT;
using System;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            DateTime currentTime = DateTime.Now;
            switch (currentTime.DayOfWeek)
            {
                case DayOfWeek.Sunday:
                    Debug.Print("Sunday");
                    break;
                case DayOfWeek.Monday:
                    Debug.Print("Monday");
                    break;
                case DayOfWeek.Tuesday:
                    Debug.Print("Tuesday");
                    break;
                case DayOfWeek.Wednesday:
                    Debug.Print("Wednsday");
                    break;
                case DayOfWeek.Thursday:
                    Debug.Print("Thursday");
                    break;
                case DayOfWeek.Friday:
                    Debug.Print("Friday");
                    break;
                case DayOfWeek.Saturday:
                    Debug.Print("Saturday");
                    break;
                default:
                    Debug.Print("We should never see this");
                    break;
            }
        }
    }
}
```

Try to step in the code to see how switch is handled in details.

17 German Serial Interfaces

There are many serial interfaces available for data transfer between processors. Each interface has its advantages and disadvantages. I will try to cover them in enough details so you can use them with NETMF.

Even though there are many serial interfaces, when we say we talk about serial, we mean UART or RS232. Other interfaces, like CAN and SPI, still transmit its data serially but they are not serial ports!!

If further details are needed, consult the web, especially http://www.wikipedia.org/

17.1 UART

UART is one of the oldest and most common interfaces. Data is sent out on a UART TXD pin in a sequence at a predefined speed. When the transmitter sends out zeros and ones, the receiver is checking for incoming data on RXD at the same speed that the transmitter is sending. The data is sent one byte at the time. This is one direction for data. To transfer the data in the opposite direction, a similar circuit is constructed at the opposite side. Transmit and receive are completely separate circuits and they can work together or independently. Each side can send data at any time and can also receive data at anytime.

The predefined speed of sending/receiving data is called baud rate. Baud rate is how many bits are transmitted in one second. Usually, one of the standard baud rates are used, like 9600, 119200, 115200 or others.

Through UART, two processors can connect directly by connecting TXD on one side to RXD on the other side and vice versa. Since this is just a digital IO on the processor, the voltage levels on UART TXD/RXD pins will be 0V (low) and 3.3V or 5V (high).

In industrial systems or when long wires are used, 3.3V or even 5V doesn't provide enough room for error. There are standards that define how we can take UART standard signal and convert it to higher voltages or differential signal to allow data transfer with better reliability. The most common one is RS232. Almost all computers are equipped with RS232 port. With RS232, data is simply UART but the voltage levels are converted from TTL (0V to 3.3V) to RS232 levels (-12V to +12V). One important fact about RS232 is that the voltage level is inverted from how we would logically think of it. When the signal is logically low, the voltage is at +12V and when the signal is logically high, the voltage is at -12V.There are many ready chips that convert TTL levels to RS232 levels, like MAX232 and MAX3232.

When we need to interface a processor using UART to a computer's serial port, we need to convert the UART TTL level to RS232 using some circuitry. For beginners or a quick hack, there are companies who provide ready circuits that convert RS232 to UART.

This is just an example: http://www.nkcelectronics.com/rs232-to-ttl-converter-board-33v232335.html

In the PC world, USB is more popular than serial ports. Newer computers, especially laptops, do not have serial ports but every computer and laptop does have a USB interface. For that reason, many companies have created a USB to UART chipset. USB<->RS232 cable is a very common device available at any computer store. An interesting product from FTDI is a USB cable with UART interface, note that it is TTL UART not RS232 on the other side of USB, which means it can be connected directly to your processor's TTL UART. The product number for this cable is ?TTL-232R-3V3?.

To summarize all above, you can interface two processors by connecting UART pins directly. To interface a processor to a PC, you need to convert the UART signals to RS232 or USB using one of the ready circuits like MAX232 for RS232 and FT232 for USB.

NETMF supports serial ports (UART) in the same way the full .NET framework on the PC is supported. To use the Serial port, add ?Microsoft.SPOT.Hardware.SerialPort? assembly and add ?using System.IO.Ports? at the beginning of your code.

Note that serial ports on PC's and on NETMF are named COM ports and they start from COM1. There is no COM0 on computer systems. This can cause a little confusion when we want to map the COM port to the UART port on the processor because the processor usually start with UART0 not UART1. So, COM1 is UART0 and COM2 is UART1...etc.

PCs have terminal programs that open the serial ports to send/receive data. Any data received will be displayed on the terminal software and any data typed in the terminal will be sent out on the serial port. Teraterm is one of common and free software that we recommended. Following is a program that sends a counter value every 10 times a second. The data is sent at 115200 so make sure the terminal is setup the same way. This program sends the data on COM1 of your NETMF device. This COM number has nothing to do with COM number on your PC. For example, you may have a USB serial port on your PC that maps to COM8 and so you need to open COM8 on your PC, not COM1 but the NETMF program will still use COM1 because it uses UART0 (COM1).

```
using System.Threading;
using System.IO.Ports;
using System.Text;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            SerialPort UART = new SerialPort("COM1", 115200);
            int counter=0;
            UART.Open();
            while (true)
            {
                // create a string
                string counter_string = "Count: " + counter.ToString() +
                                        "\r\n";
                // convert the string to bytes
                byte[] buffer = Encoding.UTF8.GetBytes(counter_string);
                // send the bytes on the serial port
                UART.Write(buffer, 0, buffer.Length);
                // increment the counter;
                counter++;
                //wait...
                Thread.Sleep(100);
            }
        }
    }
}
```

Note how we ended the string with ?\r\n?. The ?\r? is code to tel the terminal to ?return? back to the beginning of the line and ?\n? is to add ?new? line. When data is received on UART, it is automatically queued a side so you wouldn't lose any data. Note that there is limit on how much data the system can buffer so if you are debugging or not reading the data then close the serial port; otherwise, the system will become very slow and debugging/execution will be very unreliable and slow. Ideally, events will be used so we are automatically receiving data. We will cover events later.

This example will wait till byte is received then it prints back some string telling you what you have typed in (transmitted).

```
using System.Threading;
using System.IO.Ports;
using System.Text;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            SerialPort UART = new SerialPort("COM1", 115200);
            int read_count = 0;
            byte[] rx_byte = new byte[1];

            UART.Open();
            while (true)
            {

                // read one byte
                read_count = UART.Read(rx_byte, 0, 1);
                if (read_count > 0)// do we have data?
                {
                    // create a string
                    string counter_string =
                            "You typed: " + rx_byte[0].ToString() + "\r\n";
                    // convert the string to bytes
                    byte[] buffer = Encoding.UTF8.GetBytes(counter_string);
                    // send the bytes on the serial port
                    UART.Write(buffer, 0, buffer.Length);
                    //wait...
                    Thread.Sleep(10);
                }
            }
        }
    }
}
```

This last example is a loop-back. Connect a wire from TX to RX on your board and it will send data and make sure it is receiving it correctly.

```
using System.Threading;
using System.IO.Ports;
using System.Text;
using Microsoft.SPOT;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            SerialPort UART = new SerialPort("COM1", 115200);
            int read_count = 0;
            byte[] tx_data;
            byte[] rx_data = new byte[10];
            tx_data = Encoding.UTF8.GetBytes("FEZ");
            UART.ReadTimeout = 0;
            UART.Open();

            while (true)
            {
                // flush all data
                UART.Flush();
                // send some data
                UART.Write(tx_data, 0, tx_data.Length);
                // wait to make sure data is transmitted
                Thread.Sleep(100);
                // read the data
                read_count = UART.Read(rx_data, 0, rx_data.Length);
                if (read_count != 3)
                {
                    // we sent 3 so we should have 3 back
                    Debug.Print("Wrong size: " + read_count.ToString());
                }
                else
                {
                    // the count is correct so check the values
                    // I am doing this the easy way so the code is more clear
                    if (tx_data[0] == rx_data[0])
                    {
                        if (tx_data[1] == rx_data[1])
                        {
                            if (tx_data[2] == rx_data[2])
                            {
                                Debug.Print("Perfect data!");
                            }
                        }
                    }
                }
                Thread.Sleep(100);
            }
        }
    }
}
```

17.2 SPI

SPI uses three or four wires for transferring data. In UART, both sides need to have a predetermined baud rate. This is different on SPI where one of the nodes will send a clock to the other along with data. This clock is used to determine how fast the receiver needs to read the data. If you know electronics, this is a shift register. The clock is always transmitted from the master device. The other device is a slave and it doesn't send a clock but receives a clock from the master.

So, the master will transmit a clock on a SCK (serial clock) pin and will simultaneously transmit the data on MOSI (Master Out Slave In) pin. The slave will read the clock on SCK pin and simultaneously read the data from MOSI pin. So far, this is a one way communication. While data is transmitted in one direction using MOSI another set of data is sent back on MISO (Master In Slave Out) pin. This is all done simultaneously, clock send and receive. It is not possible to only send or only receive with SPI. You will always send a byte and receive a byte back in response. Other data sizes are possible but bytes are most common. NETMF supports 8-bit (byte) and 16-bit (short) data transfers. Because of this master/slave scheme, we can add multiple slaves on the same bus where the master selects which slave it will swap the data with. Note I am using the word swap because you can never send or receive but you can send and receive (swap) data. The master selects the slave using SSEL (Slave Select) pin. This pin can be called CS (Chip Select) as well. In theory, the master can have unlimited slaves but it can only select one of them at any given time. The master will only need 3 wires (SCK, MISO, MOSI) to connect to all slaves on the bus but then it needs a separate SSEL pin for each one of the slaves. Some SPI devices (slaves) can have more than one select pin, like VS1053 MP3 decoder chip that uses one pin for data and one pin for commands but both share the 3 data transfer pins (SCK, MOSI, MISO).

SPI needs more wires than other similar buses but it can transfer data very fast. A 50Mhz clock is possible on SPI, that is 50 million bits in one second. NETMF devices are always SPI masters. Before creating a SPI object, we would need a SPI configuration object. The configuration object is used to set the states of the SPI pins and some timing parameters. In most cases, you need the clock to be idle low (false) with clocking on rising edge (true) and with zero for select setup and hold time. The only thing you would need to set is the clock frequency. Some devices may accept high frequencies but others do not. Setting the clock to 1000Khz (1Mhz) should be okay for most getting started projects.

This example is sending/receiving (swapping, if you will) 10 bytes of data on SPI channel 1. Note that NETMF start numbering SPI channels (module) from 1 but on processors, the channels start from 0. So, using SPI1 in code is actually using SPI0 on the processor.

```
using System.Threading;
using System.Text;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            SPI.Configuration MyConfig =
                 new SPI.Configuration((Cpu.Pin)FEZmini.Pin.Di2x,
                 false,0,0,false,true,1000,SPI.SPI_module.SPI1);
            SPI MySPI = new SPI(MyConfig);

            byte[] tx_data = new byte[10];
            byte[] rx_data = new byte[10];
            MySPI.WriteRead(tx_data, rx_data);

            Thread.Sleep(100);
        }
    }
}
```

17.3 I2C

I2C was developed by Phillips to allow multiple chipsets to communicate on a 2-wire bus in home consumer devices, mainly TV sets. Similar to SPI, I2C have a master and one or more slaves on the same data bus. Instead of selecting the slaves using a digital pin like SPI, I2C uses software addressing. Before data is transferred, the master sends out a 7-bit address of the slave device it want communicate with. It also sends a bit indicating that if the master wants to send or receive data. The slaves that see its address on the bus will acknowledge its presence. At this point, the master and send/receive data. The master will start data transfers with ?start ? condition before it sends any address or data and then end it with ?stop? condition.

I2C NETMF driver is based on transactions. If we want to read from a register on a sensor, we would need to send it the register number and then we need to read the register. Those are two transactions, write then read.

This example will send communicate with I2C device address 0x38. It will write two (register number) and read back the register value

```
using System;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;

namespace Test
{
    public class Program
    {
        public static void Main()
        {
            //create I2C object
            I2CDevice.Configuration con =
               new I2CDevice.Configuration(0x38, 400);
            I2CDevice MyI2C = new I2CDevice(con);

            //create transactions (we need 2 in this example)
            I2CDevice.I2CTransaction[] xActions =
               new I2CDevice.I2CTransaction[2];

            // create write buffer (we need one byte)
            byte[] RegisterNum = new byte[1] { 2 };
            xActions[0] = I2CDevice.CreateWriteTransaction(RegisterNum);
            // create read buffer to read the register
            byte[] RegisterValue = new byte[1];
            xActions[1] = I2CDevice.CreateReadTransaction(RegisterValue);

            // Now we access the I2C bus and timeout in one second
            // if no response
            MyI2C.Execute(xActions, 1000);

            Debug.Print("Register value: " + RegisterValue[0].ToString());
        }
    }
}
```

17.4 One Wire

GHI exclusively supports one wire devices on NETMF. Dallas semiconductor's one wire devices, like temperature sensors or EEPROMs, use only a single wire for data transfers. Multiple devices can be connected and controlled on a single wire. The one wire class, provide many methods to read and write bytes from one wire devices. It also includes the one wire CRC calculation method as well.

This example will read the temperature from DS18B20 one wire digital thermometer. Note that this is a GHI exclusive feature and so it requires adding the GHI assembly to the build.

```
using System;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.Hardware;
using GHIElectronics.NETMF.FEZ;

namespace Test
{
    public class Program
    {
        public static void Main()
        {
            // Change this your correct pin!
            Cpu.Pin myPin = (Cpu.Pin)FEZ_Pin.Digital.Di4;
            OneWire ow = new OneWire(myPin);
            ushort temperature;

            // read every second
            while (true)
            {
                if (ow.Reset())
                {
                    ow.WriteByte(0xCC);     // Skip ROM
                    ow.WriteByte(0x44);     // Start temperature conversion

                    while (ow.ReadByte() == 0) ;   // wait while busy

                    ow.Reset();
                    ow.WriteByte(0xCC);     // skip ROM
                    ow.WriteByte(0xBE);     // Read Scratchpad

                    temperature = ow.ReadByte();                 // LSB
                    temperature |= (ushort)(ow.ReadByte() << 8); // MSB

                    Debug.Print("Temperature: " + temperature / 16);
                    Thread.Sleep(1000);
                }
                else
                {
                    Debug.Print("Device is not detected.");
                }

                Thread.Sleep(1000);
            }
        }
    }
}
```

17.5 CAN

Controller Area Network is a very common interface in industrial control and automotive. CAN is very robust and works very well in noisy environments at high speeds. All error checking and recovery methods are done automatically on the hardware. TD (Transmit Data) and RD (Receive Date) are the only two pins needed. These pins carry out the digital signal that need to be converted to analog before it is on the actual wires using the physical layer. Physical layers are sometimes called transceivers.

There are many kinds of physical layers but the most commonly used is high-speed-dual-wire that uses twisted pair for noise immunity. This transceiver can run at up to 1Mbps and can transfer data on very long wires if low bit-rate is used. Data can be transferred between nodes on the bus where any node can transfer at any time and all other nodes are required to successfully receive the data. There is no master/slave in CAN. Also, all nodes must have a predefined bit timing criteria. This is much more complicated that calculating a simple baud rate for UART. For this reason, many CAN bit rate calculators are available.

The CAN peripheral of Embedded Master is same as the popular SJA1000. A quick Internet search for SJA1000 should result in more than one free calculator.

In short, this is a simple way to calculate the bit timing:

1. Divide the system clock (72Mhz) to get an appropriate clock for the CAN peripheral (this is NOT the baud rate) This is called the BRP.
2. Figure out how many clocks you need to make up one bit. Usually this is 24 or 16. This is called TQ.
3. Assign values to T1 and T2 where T1+T2+1=TQ

Let us assume we need 250Kbps.

1. From 72Mhz system clock, I want the CAN clock to be 6Mhz so I need to divide by 12 (BRP=12).
2. 6Mhz/250kbps = 24 TQ (we usually want 16 or 24).
3. T1 = 15, T2 = 8 will give us 15 + 8 + 1 = 24 and this is what we need.

I got the T1 and T2 values from: http://www.kvaser.com/can/protocol/index.htm I picked the first value and subtracted 1 from T1 because the calculator included the sync bit

Here is the code with detailed comments

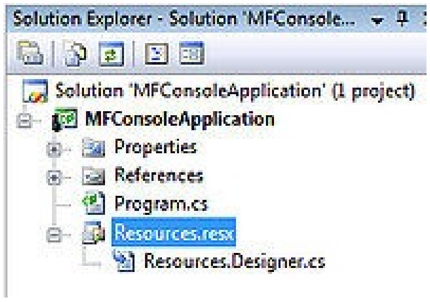
```
using System.Threading;
using Microsoft.SPOT;
using System;
using GHIElectronics.Hardware;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            // These numbers were calculated using the
            // calculator on this link:
            // http://www.kvaser.com/can/protocol/index.htm
            // We used the very first value from the calculator output
            ///////////////////////////////////////////////////////////
            // Bitrate 250Kbps
            // CLK = 72 Mhz, with BRP = 12 -> 6Mhz CAN clock
            // 6Mhz/250Kbps = 24 TQ
            // T1 = 16 minus 1 for sync = 15
            // T2 = 8
            // 15 + 1 + 8 = 24 TQs which is what we need
            //////////////////////////////////////////////////////////
            const int BRP = 12;
            const int T1 = 15;
            const int T2 = 8;
            // For 500Kbps you can use BRP=6 and for 1Mbps you can use BRP=3
            // and for 125Kbps use BRP=24...and so on
            // Keep T1 and T2 the same to keep the sampling pint
            // the same (between 70% and 80%)

            // Initialize CAN channel, set bit rate
            CAN canChannel = new CAN(CAN.CANChannel.Channel_1,
                 ((T2 - 1) << 20) | ((T1 - 1) << 16) | ((BRP - 1) << 0));
            // make new CAN message
            CAN.CANMessage message = new CAN.CANMessage();
            // make a message of 8 bytes
            for (int i = 0; i < 8; i++)
                message.data[i] = (byte)i;
            message.DLC = 8; // 8 bytes
            message.ArbID = 0xAB; // ID
            message.isEID = false; // not extended ID
            message.isRTR = false; // not remote
            // send the message
            canChannel.PostMessage(message);
            // wait for a message and get it.
            while (canChannel.GetRxQueueCount() == 0) ;
            // get the message using the same message object
            canChannel.GetMessage(message);
            // Now "message" contains the data, ID,
            // flags of the received message.
        }
    }
}
```

18 German Loading Resources

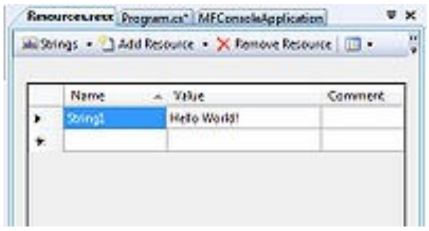
Eine Resource ist eine Datei die in das Programm integriert wird. Wenn unser Programm von einer Datei abhängig ist (Bilder, Icons, Audio, usw.) können wir diese Dateien zu unseren Programm Resources hinzufügen. Das Programm könnte die Resource Dateien auch direkt vom Dateisystem auslesen nur wäre Sie dann von diesem abhängig. Resourcen können auch strings sein, z.B. ein copyright Text. Wenn wir, z.B. den copyright Text ändern wollen, müssen wir nur den Text selber ändern aber nichts am Programm Code.

Beachten Sie immer wieviel Speicher für das Programm reserviert wird. Das hinzufügen von grossen Dateien kann in Fehlern enden, wobei Ihnen VS2010 nicht mitteilen wird dass der Fehler auf Grund zu grosser Dateien auftritt.

Im ?Solution Explorer? Fenster finden Sie ?Resource.resx? und wenn Sie erweitern sehen Sie die Datei ?Resources.Designer.cs?.

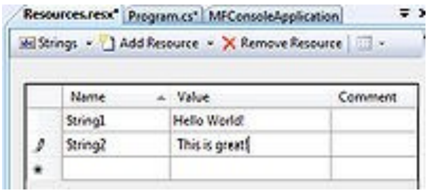

Solution Explorer - Solution 'MFConsole...
Solution 'MFConsoleApplication' (1 project)
MFConsoleApplication
Properties
References
Program.cs
Resources.resx
Resources.Designer.cs

Visual Studio generiert automatisch die ?resources designer? Datei. Versuchen Sie niemals diese Datei selbst zu ändern. Doppelklicken Sie stattdessen auf die Datei ?Resources.resx? um sie in einem geeigneten Tool zu bearbeiten.



Name	Value	Comment
String1	Hello World!	

Im ?Resource Editor? Fenster können Sie in der ersten drop-down Liste auf der linken Seite die Resource die Sie bearbeiten möchten auswählen. Wie Sie sehen können ist bereits eine ?string resource? namens ?String_1? angelegt die den Wert ?value? ?Hello World!? hat. Klicken Sie direkt unter ?String_1? und fügen Sie eine neue String Resource hinzu wie im nachfolgenden Bild gezeigt.



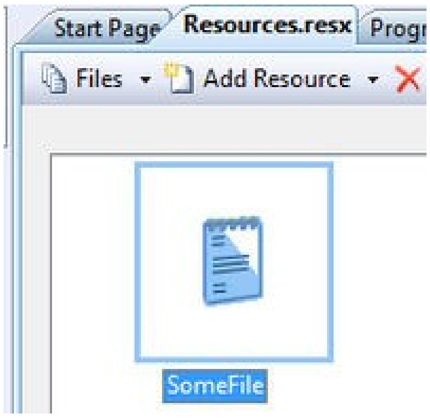
Name	Value	Comment
String1	Hello World!	
String2	This is great!	

Nun haben wir 2 String Resources. Lassen Sie uns diese nun verwenden

```
using System.Threading;
using Microsoft.SPOT;
using System;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            Debug.Print(
               Resources.GetString(Resources.StringResources.String1));
            Debug.Print(
               Resources.GetString(Resources.StringResources.String2));
        }
    }
}
```

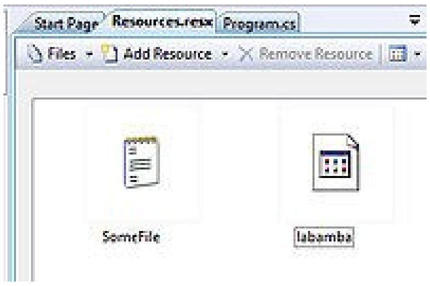
Versuchen Sie den Text (value) der String Resourcen zu ändern und beobachten Sie, wie sich die Ausgabe ändert.

Lassen Sie uns eine Datei hinzufügen. Erzeugen Sie auf Ihrem Desktop eine Textdatei und schreiben Sie ein paar Zeilen Text hinein. Wählen Sie ?files? vom linken drop-down Menü und ?Add Resource...?



Obwohl es sich um eine Textdatei handelt, fügt Visual Studio sie ähnlich den Strings in das Projekt ein. Um den Inhalt der Datei zu nutzen greifen wir darauf wie bei einem String zu.

```
using System.Threading;
using Microsoft.SPOT;
using System;

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



Die Datei ist 33Kb gross. Das ist in der Computerwelt nicht unbedingt gross aber in integrierten Systemen schon. Das unten angeführte Beispiel wird auf Systemen mit viel RAM, wie ChipworkX und Embedded Master, laufen. Mit USBizi und FEZ kann es funktionieren, muss aber nicht. Wir können mit USBizi und FEZ schon grosse Dateien öffnen, aber statt die Datei gleich vollständig einzulesen, wird nur ein Teil eingelesen und zum Decoder geschickt, dann der nächste Teil usw. Details und Erklärungen folgen später.

```
using System.Threading;
using Microsoft.SPOT;
using System;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            byte[] file_data =
                 Resources.GetBytes(Resources.BinaryResources.labamba);
            Debug.Print("File Size is " + file_data.Length.ToString());
        }
    }
}
```

19 German Output Compare

This exclusive GHI feature allows developers to generate all kind of digital signals on any of the digital IOs. For example, OutputCompare can be used to generate a UART signal. Or, generate a signal to control a 38Khz infrared transmitter to simulate a TV remote control.

One very good example is the driver of 2x16 character serial display found on www.TinyCLR.com. The display is controlled by UART. Also, the display doesn't send any data back. All you need is to send some simple control commands serially to the display. Now, we can just connect the display to one of the available serial ports. But then we will lose the port over something so simple plus the UART serial interface uses 2 pins, for transmit and for receive. But, the display doesn't send any data back so basically you will lose the receive pin. The right way to control this serial display is by using OutputCompare. You will only need one pin and you can use any of the digital pins.

So how does OutputCompare work? Basically, you provide the method with an array of time values between each pin toggle. The OutputCompare then goes through the array of time values and generate the signal on the requested pin. So if are going to generate UART, we first need to pre-calculate the values needed to represent the transmitted byte and then provide that to the OutputCompare object.

The example driver should explain how this is done. This is a copy of the driver found on www.TinyCLR.com for the serial LCD driver

```
using System;
using System.Threading;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.Hardware;
namespace GHIElectronics.NETMF.FEZ
{
    public static partial class FEZ_Components
    {
        public class SerialLCD : IDisposable
        {
            const byte DISP_ON = 0xC;    // Turn visible LCD on
            const byte CLR_DISP = 0x01;  // Clear display
            const byte CUR_HOME = 2;     // Move cursor home
                                         // and clear screen memory
            const byte SET_CURSOR = 0x80;   // SET_CURSOR + X :
                                            // Sets cursor position to X
            const byte Move_CURSOR_LEFT = 0x10;

            OutputCompare oc;
            const int MAX_TIMINGS_BUFFER_SIZE = 10;
            uint[] buffer = new uint[MAX_TIMINGS_BUFFER_SIZE];
            const int BAUD_RATE = 2400;
            const int BIT_TIME_US = 1 * 1000 * 1000 / BAUD_RATE;
            readonly int BYTE_TIME_MS;
            public void Dispose()
            {
                oc.Dispose();
                buffer = null;
            }
            private void SendByte(byte b)
            {
                bool currentPinState;
                int currentBufferIndex = 0;
                uint currentStateTiming;
                // start bit
                currentPinState = false;
                currentStateTiming = BIT_TIME_US;
                // data bits
                for (int i = 0; i < 8; i++)
                {
                    bool neededState = (b & (1 << i)) != 0;

                    if (neededState != currentPinState)
                    {
                        buffer[currentBufferIndex] = currentStateTiming;
                        currentStateTiming = BIT_TIME_US;
                        currentPinState = neededState;
                        currentBufferIndex++;
                    }
                    else
                    {
                        currentStateTiming += BIT_TIME_US;
                    }
                }
                // stop bit
                if (currentPinState != true)
                {
                    buffer[currentBufferIndex] = currentStateTiming;
                    currentBufferIndex++;
                }
                oc.Set(false, buffer, 0, currentBufferIndex, false);
                // wait till data is sent
                Thread.Sleep(BYTE_TIME_MS);
            }
            public void PutC(char c)
            {
                SendByte((byte)c);
            }

            private void SendCommand(byte cmd)
            {
                SendByte(0xFE);
                SendByte(cmd);
            }
            public void Print(string str)
            {
                for (int i = 0; i < str.Length; i++)
                    PutC(str[i]);
            }
            public void ClearScreen()
            {
                SendCommand(CLR_DISP);
            }
            public void CursorHome()
            {
                SendCommand(CUR_HOME);
            }
            public void SetCursor(byte row, byte col)
            {
                SendCommand((byte)(SET_CURSOR | row << 6 | col));
            }
            public void MoveLeft()
            {
                SendCommand(Move_CURSOR_LEFT);
            }
            public SerialLCD(FEZ_Pin.Digital pin)
            {
                BYTE_TIME_MS = (int)Math.Ceiling((double)BIT_TIME_US *
                                MAX_TIMINGS_BUFFER_SIZE / 1000);
                oc = new OutputCompare((Cpu.Pin)pin, true,
                                       MAX_TIMINGS_BUFFER_SIZE);
                // Initilaize LCD
                SendCommand(DISP_ON);
                SendCommand(CLR_DISP);
            }
        }
    }
}
```

20 German Displays

20.1 Character Displays

Most character displays use the same interface. Those displays are mostly 2-lines by 16-characters, commonly known as 2x16 character displays. The display can be controlled using 8-bit or 4-bit interface. The 4-bit option is more favorable since it requires less IOs.

The interface uses RS (Data/Instruction), RW(Read/Write), E (Enable) and 4-bit data bus. The display manual is the best resource of information but this is a simple example class to get the display running quick.

GHI offers an alternative to this display. The SerialLCD display offered on www.TinyCLR.com will work on any single pin on FEZ. The included driver makes using these displays even easier.

Using the display driver

```
using GHIElectronics.NETMF.FEZ;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            FEZ_Components.SerialLCD LCD =
                new FEZ_Components.SerialLCD(FEZ_Pin.Digital.Di5);

            LCD.ClearScreen();
            LCD.CursorHome();
            LCD.Print("FEZ Rocks!");
        }
    }
}
```

20.2 Graphical Displays

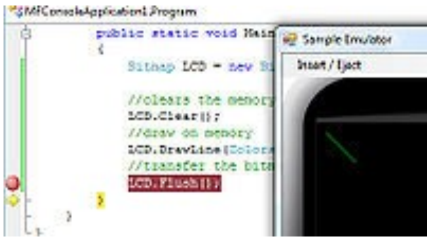
20.2.1 Native support

NETMF, with its bitmap class, can handle graphics very well. The bitmap class supports images from BMP, JPG and GIF file types. The images can be obtained from the file system or the network but an easier option is to include the image in a resource. The bitmap object can be used to draw images or shapes and to draw text using a font. NETMF supports creating fonts using TFConvert tool. I will not cover the use of TFConvert so we will use one of the available fonts. When we draw using a bitmap object, we are actually drawing on the object (on memory) nothing is visible on the screen. To transfer a bitmap object from memory to the screen, we need to ?flush? the bitmap object. An important note here is that flush will only work if the size of the bitmap is exactly the same size of the screen. If you have an image size 128x128 pixels and want to display it on the screen, we first need to create a new bitmap object with the size of the screen, then create a second bitmap object for the smaller image. Draw the smaller image on the large one and then flush! To eliminate confusion, I always have a one bitmap object called LCD and then everything gets drawn on this object. We will run all these tests on the emulator instead of hardware as your hardware may not support native graphics.

```
using System.Threading;
using Microsoft.SPOT;
using System;
using Microsoft.SPOT.Presentation;
using Microsoft.SPOT.Presentation.Media;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            Bitmap LCD = new Bitmap(SystemMetrics.ScreenWidth,
                                    SystemMetrics.ScreenHeight);

            //clears the memory and not the display
            LCD.Clear();
            //draw on memory
            LCD.DrawLine(Colors.Green, 1, 10, 10, 40, 40);
            //transfer the bitmap memory to the actual display
            LCD.Flush();
        }
    }
}
```

The code above requires Microsoft.SPOT.TinyCore assembly to run. Then we need to use the presentation namespace so we can get the ?SystemMetrics?.

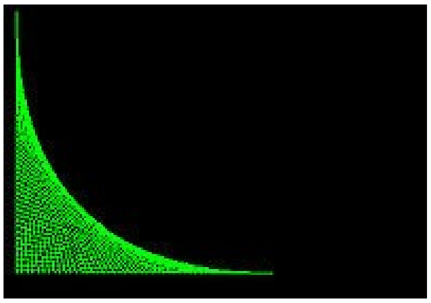
Run the code and you will see a green line on the emulator.



```
MFConsoleApplication1.Program
        public static void Main
        {
            Bitmap LCD = new Bi

            //clears the memory
            LCD.Clear();
            //draw on memory
            LCD.DrawLine(Colors
            //transfer the bitm
            LCD.Flush();
        }
    }
}
```

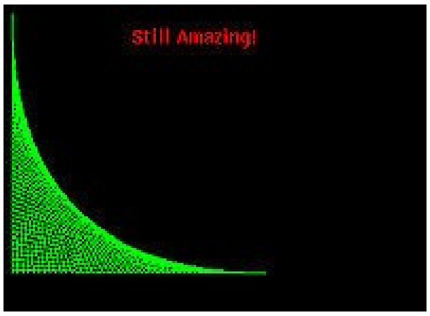
Try to use what we learned on for-loop to create multiple lines.

```
using System.Threading;
using Microsoft.SPOT;
using System;
using Microsoft.SPOT.Presentation;
using Microsoft.SPOT.Presentation.Media;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            Bitmap LCD = new Bitmap(SystemMetrics.ScreenWidth,
                                    SystemMetrics.ScreenHeight);

            //clears the memory and not the display
            LCD.Clear();
            int i;
            for (i = 10; i < 200; i += 4)
            {
                //draw on memory
                LCD.DrawLine(Colors.Green, 1, 10, i, i, 200);
            }
            //transfer the bitmap memory to the actual display
            LCD.Flush();
        }
    }
}
```



To draw text, we would need to have a font resource first. Add a new resource to the project. You can use one of the resource files coming with NETMF SDK examples. The samples at ...\Documents\Microsoft .NET Micro Framework 3.0\Samples\ I used ?NinaB.tinyfnt? font file. Add the file to your resources like explained in previous chapter. We can now run this code to print on the LCD.

```
sing System.Threading;
using Microsoft.SPOT;
using System;
using Microsoft.SPOT.Presentation;
using Microsoft.SPOT.Presentation.Media;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            Bitmap LCD = new Bitmap(SystemMetrics.ScreenWidth,
                                    SystemMetrics.ScreenHeight);
            Font MyFont = Resources.GetFont(Resources.FontResources.NinaB);

            //clears the memory and not the display
            LCD.Clear();
            int i;
            for (i = 10; i < 200; i += 4)
            {
                //draw on memory
                LCD.DrawLine(Colors.Green, 1, 10, i, i, 200);
            }
            // print some text on the screen
            LCD.DrawText("Still Amazing!", MyFont, Colors.Red, 100, 20);
            //transfer the bitmap memory to teh actual display
            LCD.Flush();
        }
    }
}
```



20.2.2 Non-native support

Many small graphical displays use SPI bus to receive images from the host. NETMF devices usually support large TFT displays that use a special bus to operate, like FEZ Cobra (EMX). But, even if the system doesn't support those displays, like FEZ Rhino (USBizi), user can connect an SPI-based display and display graphics this way. It is also possible to have two displays on system that support native TFT interface. A large display will run on the TFT interface and the smaller display will run on a SPI bus. Even tough SPI is very fast, displays can have millions of pixels so we need to take this under consideration when selecting a display. Www.TinyCLR.com offers many displays, SPI or TFT that are ideal to work with GHI offers. Below is one of the example. Also check the standard SPI-based display for FEZ Rhino on the website mentioned earlier.

20.2.3 Native-support on non-standard display

A better option is to use the bitmap class to draw text, draw shapes and then transfer the bitmap to your display. You can only use this if Bitmap (graphics) is supported on your device, USBizi doesn't support graphics. This code will display some data on the F-51852 128x64 pixel display found on the old non-TFT Embedded Master development system.

```
using System;
using System.Text;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Presentation;
using Microsoft.SPOT.Presentation.Media;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.System;

    public class Program
    {
        static SPI.Configuration conf = new SPI.Configuration((Cpu.Pin)33,
               false, 0, 0, false, true, 1000, SPI.SPI_module.SPI2);
        static SPI SPI_port = new SPI(conf);
        static OutputPort RST = new OutputPort((Cpu.Pin)9, true);
        static OutputPort DC = new OutputPort((Cpu.Pin)15, true);
        static byte[] _ba = new byte[1];
        static void OPTREX_SSD_WriteCmdByte(byte b)
        {
            DC.Write(false);

            Thread.Sleep(1);

            _ba[0] = b;
            SPI_port.Write(_ba);
        }

        static void OPTREX_SSD_WriteByte(byte b)
        {
            DC.Write(true);

            Thread.Sleep(1);

            _ba[0] = b;
            SPI_port.Write(_ba);
        }
        static void OPTREX_Locate(int x, int y)
        {
            if (y > 7)
                y = 7;

            if (x > 127)
                x = 127;

            OPTREX_SSD_WriteCmdByte((byte)(0X10 | (x >> 4)));//col up
            OPTREX_SSD_WriteCmdByte((byte)(0X00 | (x & 0xF)));//col down

            OPTREX_SSD_WriteCmdByte((byte)(0XB0 | y));//page addr

        }
        public static void Main()
        {

            OPTREX_SSD_WriteCmdByte(0XA2);//bias
            OPTREX_SSD_WriteCmdByte(0XA1);//adc   inverse
            OPTREX_SSD_WriteCmdByte(0Xc0);//common dir...normal
            OPTREX_SSD_WriteCmdByte(0X40);//initial line
            OPTREX_SSD_WriteCmdByte(0X81);//evr set
            OPTREX_SSD_WriteCmdByte(0X20);
            OPTREX_SSD_WriteCmdByte(0X29);//2B we have -10V........
                                          //wait for stable voltage
            Thread.Sleep(10);
            OPTREX_SSD_WriteCmdByte(0XA4);//turn all on
            OPTREX_SSD_WriteCmdByte(0XE7);//driver
            OPTREX_SSD_WriteCmdByte(0XAF);//lcd on

            //OPTREX_SSD_WriteCmdByte(0XA7);//inverse
            OPTREX_SSD_WriteCmdByte(0XA6);//no inverse


            OPTREX_SSD_WriteCmdByte(0XB0);//page addr
            OPTREX_SSD_WriteCmdByte(0X10);//col
            OPTREX_SSD_WriteCmdByte(0X00);//col

            int x = 20;
            int y = 50;
            int dx = -2;
            int dy = -3;

            Bitmap bb = new Bitmap(128, 64);
            byte[] bitmapbytes;
            Font fnt = Resources.GetFont(Resources.FontResources.small);
            byte[] vram = new byte[128 * 64 / 8];
            byte[] singleline = new byte[128];
            while (true)
            {
                bb.Clear();
                bb.SetPixel(0, 0, Color.White);
                bb.SetPixel(0, 2, Color.White);
                bb.SetPixel(2, 0, Color.White);
                bb.SetPixel(2, 2, Color.White);
                bb.DrawText("Rocks!", fnt, Color.White, 20, 45);
                bb.DrawEllipse(Color.White, x, y, 5, 3);

                x += dx;
                y += dy;
                if (x < 0 || x > 128)
                    dx = -dx;
                if (y < 0 || y > 64)
                    dy = -dy;

                bitmapbytes = bb.GetBitmap();
                Util.BitmapConvertBPP(bitmapbytes, vram,
                                      Util.BPP_Type.BPP1_x128);

                for (int l = 0; l < 8; l++)
                {
                    OPTREX_Locate(0, l);
                    DC.Write(true);
                    Array.Copy(vram, l * 128, singleline, 0, 128);
                    SPI_port.Write(singleline);
                }
                Thread.Sleep(1);
            }
```

```
        }
}
```

21 German Time Services

In computer systems, time can mean two things. The system time, which is the processor ticks, is used to handle threads timing and all timing management in the system. On the other hand, the real time clock is used to time human time, like minutes, hours and even dates.

21.1 Real Time Clock

All GHI NETMF devices have a built in RTC

```
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using System;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            // set the time to 9/9/2010 at 9:09:09
            DateTime time = new DateTime(2010, 9, 9, 9, 9, 9);
            Utility.SetLocalTime(time);
            while (true)
            {
                Debug.Print(DateTime.Now.ToString());
                Thread.Sleep(100);
            }
        }
    }
}
```

To use the RTC hardware, we first need to check if the RTC hardware has the valid time or not. Maybe this is a new battery or the new system and the RTC has not been set yet. If RTC has a valid time then we can read the RTC (hardware) and use that to set the NETMF system time (software). If time is not valid, then you will need to set the RTC to the correct time.

```
using System;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.Hardware;

    public class Program
    {
        public static void Main()
        {
            // To keep track of time,
            // set it at the beginning of your application from the RTC.
            // If it was NOT set before and currently running using
            // the battery (not exhausted), set it to a fixed time.
            if (RealTimeClock.IsTimeValid == false)
                RealTimeClock.SetTime(new DateTime(2010, 03, 01, 12, 0, 0, 0));

            Utility.SetLocalTime(RealTimeClock.GetTime());
        }
    }
```

21.2 Timers

Micro Framework includes 2 timer classes, Timer and ExtendedTimes. Timer class is the same one included in full framework where ExtendedTimer is specific to NETMF with extra functionality. For basic beginner projects, I suggest you keep on using threading before getting into timers. I will only provide an example in this book. Our example will create a timer that will run after five seconds and then it will keep firing every second.

```
using System.Threading;
using Microsoft.SPOT;
using System;

namespace MFConsoleApplication1
{
    public class Program
    {
        static void RunMe(object o)
        {
            Debug.Print("From timer!");
        }
        public static void Main()
        {
            Timer MyTimer =
               new Timer(new TimerCallback(RunMe), null, 5000, 1000);
            Debug.Print(
                   "The timer will fire in 5 seconds and then fire
                    priodically every 1 second");
            Thread.Sleep(Timeout.Infinite);
        }
    }
}
```

22 German USB Host

There is almost always confusion between USB host and USB device. The USB host is the system that connects to multiple USB devices. For example, the PC is a USB host and it can connect to multiple USB devices like mice, keyboards and mass storage devices. Implementing a USB device is rather simple but implementing a host is far more complicated.

USB host is an exclusive feature from GHI Electronics. With this exclusive feature, you can connect almost any USB device to GHI's NETMF products (USBizi, Embedded Master, ChipworkX). This feature opens new possibilities for embedded systems. Your product can now connect to a standard USB keyboard and can also access files on a USB thumb drive! USB is a hot pluggable system which means any device can be connected or disconnected any time. There are events generated when devices are connected or disconnected. The developer should subscribe to these events and handle devices accordingly. Since this is a beginner book, I will assume that the device will always be connected to the system.

With USB HUB support, devices can be connected directly to the USB host port or a user may connect multiple USB devices through a USB hub. First, let us detect what devices are connected. The first thing to do is start the system manager then we can get a list of available devices. Remember that we need to add the GHI library assembly to our resources.

```
using System;
using System.Threading;
using Microsoft.SPOT;

using GHIElectronics.NETMF.USBHost;

namespace Test
{
  class Program
  {
    public static void Main()
    {
      // Subscribe to USBH events.
      USBHostController.DeviceConnectedEvent += DeviceConnectedEvent;
      USBHostController.DeviceDisconnectedEvent += DeviceDisconnectedEvent;

      // Sleep forever
      Thread.Sleep(Timeout.Infinite);
    }

    static void DeviceConnectedEvent(USBH_Device device)
    {
        Debug.Print("Device connected...");
       Debug.Print("ID: " + device.ID + ", Interface: " +
                   device.INTERFACE_INDEX + ", Type: " + device.TYPE);
       }

       static void DeviceDisconnectedEvent(USBH_Device device)
       {
          Debug.Print("Device disconnected...");
            Debug.Print("ID: " + device.ID + ", Interface: " +
                        device.INTERFACE_INDEX + ", Type: " + device.TYPE);
        }
    }
}
```

When we detect a device, we can communicate with it directly. This requires a lot of knowledge on USB devices. Fortunately, most devices fall under standard classes and GHI already provide drivers for them.

22.1 HID Devices

Human Interface Devices like mice, keyboards and joysticks are directly supported. Using HID is event based. Events are methods you create and then you subscribe them to a certain event. When that event fires, your method will get executed automatically.

```
using System;
using System.Threading;
using Microsoft.SPOT;
using GHIElectronics.NETMF.USBHost;

namespace Test
{
    public class Program
    {
        static USBH_Mouse mouse;
        public static void Main()
        {
            // Subscribe to USBH event.
            USBHostController.DeviceConnectedEvent += DeviceConnectedEvent;

            // Sleep forever
            Thread.Sleep(Timeout.Infinite);
        }

        static void DeviceConnectedEvent(USBH_Device device)
        {
            if (device.TYPE == USBH_DeviceType.Mouse)
            {
                Debug.Print("Mouse Connected");
                mouse = new USBH_Mouse(device);
                mouse.MouseMove += MouseMove;
                mouse.MouseDown += MouseDown;
            }
        }

        static void MouseMove(USBH_Mouse sender, USBH_MouseEventArgs args)
        {
            Debug.Print("(x, y) = (" + sender.Cursor.X + ", " +
                        sender.Cursor.Y + ")");
        }

        static void MouseDown(USBH_Mouse sender, USBH_MouseEventArgs args)
        {
            Debug.Print("Button down number: " + args.ChangedButton);
        }
    }
}
```

Accessing Joysticks is very similar. Here is the example modified to work with joysticks

```
using System;
using System.Threading;
using Microsoft.SPOT;
using GHIElectronics.NETMF.USBHost;
namespace Test
{
    public class Program
    {
        static USBH_Joystick j;
        public static void Main()
        {
            // Subscribe to USBH event.
            USBHostController.DeviceConnectedEvent += DeviceConnectedEvent;

            // Sleep forever
            Thread.Sleep(Timeout.Infinite);
        }

        static void DeviceConnectedEvent(USBH_Device device)
        {
            if (device.TYPE == USBH_DeviceType.Joystick)
            {
                Debug.Print("Joystick Connected");
                j = new USBH_Joystick(device);
                j.JoystickXYMove += JoystickXYMove;
                j.JoystickButtonDown += JoystickButtonDown;
            }
        }

        static void JoystickButtonDown(USBH_Joystick sender,
                                       USBH_JoystickEventArgs args)
        {
            Debug.Print("Button Pressed: " + args.ChangedButton);
        }

        static void JoystickXYMove(USBH_Joystick sender,
                                   USBH_JoystickEventArgs args)
        {
            Debug.Print("(x, y) = (" + sender.Cursor.X + ", " +
                        sender.Cursor.Y + ")");
        }
    }
}
```

22.2 Serial Devices

Serial (UART) communication is a very common interface. There are many companies that create chips that convert USB to serial. GHI supports chipsets from FTDI, Silabs and Prolific. Also,there is a standard USB class defined for serial communication called CDC (Communication Device Class). This class is supported as well. Note here that the USB chipsets are made to be somewhat customized. So a company can use a FTDI chip to make their product run on USB and they will change the strings in USB descriptors so when you plug in their device to a PC you will see the company name not FTDI name. They can also change the USB VID/PID, vendor ID and product ID. A good example is a USB GPS device. Almost all those USB GPS devices use prolific chip, which is supported by GHI. Many of the interface products on the market use FTDI chipset.

```
using System;
using System.Text;
using System.Threading;

using Microsoft.SPOT;

using GHIElectronics.NETMF.USBHost;
```

```
namespace Test
{
    class Program
    {
        static USBH_SerialUSB serialUSB;
        static Thread serialUSBThread; // Prints data every second

         public static void Main()
        {
            // Subscribe to USBH event.
            USBHostController.DeviceConnectedEvent += DeviceConnectedEvent;

            // Sleep forever
            Thread.Sleep(Timeout.Infinite);
        }

        static void DeviceConnectedEvent(USBH_Device device)
        {
            Debug.Print("Device connected");

            switch (device.TYPE)
            {
                case USBH_DeviceType.Serial_FTDI: // FTDI connected
                    serialUSB = new USBH_SerialUSB(device, 9600,
                         System.IO.Ports.Parity.None, 8,
                         System.IO.Ports.StopBits.One);
                    serialUSB.Open();
                    serialUSBThread = new Thread(SerialUSBThread);
                    serialUSBThread.Start();

                    break;

                case USBH_DeviceType.Unknown: // SiLabs but not recognized
                    // force SiLabs
                    USBH_Device silabs = new USBH_Device(device.ID,
                         device.INTERFACE_INDEX,
                         USBH_DeviceType.Serial_SiLabs, device.VENDOR_ID,
                         device.PRODUCT_ID, device.PORT_NUMBER);
                    serialUSB = new USBH_SerialUSB(silabs, 9600,
                         System.IO.Ports.Parity.None, 8,
                         System.IO.Ports.StopBits.One);
                    serialUSB.Open();
                    serialUSBThread = new Thread(SerialUSBThread);
                    serialUSBThread.Start();

                    break;
            }
        }

        static void SerialUSBThread()
        {
            // Print "Hello World!" every second.
            byte[] data = Encoding.UTF8.GetBytes("Hello World!\r\n");
            while (true)
            {
                Thread.Sleep(1000);

                serialUSB.Write(data, 0, data.Length);
            }
        }
    }
}
```

22.3 Mass Storage

Storage devices like USB hard drives and USB thumb memory drives use the same USB class, MSC (Mass Storage Class). GHI library directly support those devices. USB only defines how to read/write raw sectors on the media. The operating system then handles the file system. NETMF supports FAT32 and FAT16 file system. To access the files on a USB media, we first need to detect it then we need to mount the media.

```
using System;
using System.IO;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.IO;
using GHIElectronics.NETMF.IO;
using GHIElectronics.NETMF.USBHost;

namespace Test
{
    class Program
    {
        public static void Main()
        {
            // Subscribe to RemovableMedia events
            RemovableMedia.Insert += RemovableMedia_Insert;
            RemovableMedia.Eject += RemovableMedia_Eject;
            // Subscribe to USB events
            USBHostController.DeviceConnectedEvent += DeviceConnectedEvent;

            // Sleep forever
            Thread.Sleep(Timeout.Infinite);
        }

        static void DeviceConnectedEvent(USBH_Device device)
        {
            if (device.TYPE == USBH_DeviceType.MassStorage)
            {
                Debug.Print("USB Mass Storage detected...");
                //....
                //....
            }
        }
    }
}
```

The next section explains how to access the files on the USB memory.

23 German File System

File system is supported starting in NETMF 3.0. GHI adds more functionality to the standard support. For example, a SD card can be mounted to the file system or mounted to the USB device MSC service. When mounted to the filer system, developers can access the files. But, when mounted to the USB device MSC, a PC connected to the USB port will see a USB card reader with SD card. This is good for creating data logger for example. The device will log data on the SD card and when the device is plugged to a PC, the device will become a card reader for the same SD memory card. GHI's persistent storage class is used to handle mounting devices on the system.

This section will only cover using persistent storage device with internal file system.

23.1 SD Cards

First we need to detect the SD card insertion. SD card connectors usually have a little switch internally that closes when a card is inserted. In this example, I will assume the card is always inserted and there is no need to look for detection. The example will list all files available in the root directory.

```
using System;
using System.IO;
using System.Threading;

using Microsoft.SPOT;
using Microsoft.SPOT.IO;

using GHIElectronics.NETMF.IO;

namespace Test
{
    class Program
    {
        public static void Main()
        {
            // ...
            // SD Card is inserted
            // Create a new storage device
            PersistentStorage sdPS = new PersistentStorage("SD");

            // Mount the file system
            sdPS.MountFileSystem();


            // Assume one storage device is available, access it through
            // Micro Framework and display available files and folders:
            Debug.Print("Getting files and folders:");
            if (VolumeInfo.GetVolumes()[0].IsFormatted)
            {
                string rootDirectory =
                    VolumeInfo.GetVolumes()[0].RootDirectory;
                string[] files = Directory.GetFiles(rootDirectory);
                string[] folders = Directory.GetDirectories(rootDirectory);

                Debug.Print("Files available on " + rootDirectory + ":");
                for (int i = 0; i < files.Length; i++)
                    Debug.Print(files[i]);

                Debug.Print("Folders available on " + rootDirectory + ":");
                for (int i = 0; i < folders.Length; i++)
                    Debug.Print(folders[i]);
            }
            else
            {
                Debug.Print("Storage is not formatted. Format on PC with
                             FAT32/FAT16 first.");
            }

            // Unmount
            sdPS.UnmountFileSystem();
        }
    }
}
```

There is more than one way to open files. I will only cover FileStream objects. This example will open a file and write a string to it. Since FileStream will only take byte arrays, we need to convert our string to byte array.

```
using System.Threading;
using System.Text;
using Microsoft.SPOT;
using System.IO;
using Microsoft.SPOT.IO;
using GHIElectronics.NETMF.IO;

namespace MFConsoleApplication1
{
    public class Program
    {
        static void Main()
        {
            // ... check if SD is inserted

            // SD Card is inserted
            // Create a new storage device
            PersistentStorage sdPS = new PersistentStorage("SD");

            // Mount the file system
            sdPS.MountFileSystem();

            // Assume one storage device is available,
            // access it through NETMF
            string rootDirectory = VolumeInfo.GetVolumes()[0].RootDirectory;
            FileStream FileHandle = new FileStream(rootDirectory +
                                          @"\hello.txt", FileMode.Create);
            byte[] data =
               Encoding.UTF8.GetBytes("This string will go in the file!");
            // write the data and close the file
            FileHandle.Write(data, 0, data.Length);
            FileHandle.Close();

            // if we need to unmount
            sdPS.UnmountFileSystem();

            // ...
            Thread.Sleep(Timeout.Infinite);

        }
    }
}
```

Verify the file if it is on the card using a PC and memory card reader if you like. Now, we want to open the same file and read the string we stored earlier.

```
using System.Threading;
using System.Text;
using Microsoft.SPOT;
using System.IO;
using Microsoft.SPOT.IO;
using GHIElectronics.NETMF.IO;

namespace MFConsoleApplication1
{
    public class Program
    {
        static void Main()
        {
            // ... check if SD is inserted

            // SD Card is inserted
            // Create a new storage device
            PersistentStorage sdPS = new PersistentStorage("SD");

            // Mount the file system
            sdPS.MountFileSystem();

            // Assume one storage device is available,
            // access it through NETMF
            string rootDirectory = VolumeInfo.GetVolumes()[0].RootDirectory;
            FileStream FileHandle = new FileStream(rootDirectory +
                    @"\hello.txt", FileMode.Open, FileAccess.Read);
            byte[] data = new byte[100];
            // write the data and close the file
            int read_count = FileHandle.Read(data, 0, data.Length);
            FileHandle.Close();
            Debug.Print("The size of data we read is: " +
                        read_count.ToString());
            Debug.Print("Data from file:");
            Debug.Print(new string(Encoding.UTF8.GetChars(data), 0,
                  read_count));

            // if we need to unmount
            sdPS.UnmountFileSystem();

            // ...
            Thread.Sleep(Timeout.Infinite);

        }
    }
}
```

23.1.1 Auto-Mounting SD Cards

SD cards must be mounted before using them. Afterwards, they are accessed using Insert and Eject events. There are two options for automatically detecting the SD cards using software and hardware.

Option 1: Some platforms, like USBizi and EMX, can support software SD card detection. This works on most SD cards. Simply call DetectSDCard method to check if an SD card exists. You can put this call in a loop for continuous detection. Just copy this example and drop in any project you have to get insert and eject events.

```
using System;
using System.IO;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.IO;

using GHIElectronics.NETMF.IO;

namespace AutoMount
{
    public class Program
    {
        public static void Main()
        {
            RemovableMedia.Insert += new InsertEventHandler(RemovableMedia_Insert);
            RemovableMedia.Eject += new EjectEventHandler(RemovableMedia_Eject);

            // Start auto mounting thread
            new Thread(SDMountThread).Start();

            // Your program goes here
            // ...

            Thread.Sleep(Timeout.Infinite);
        }

        static void RemovableMedia_Eject(object sender, MediaEventArgs e)
        {
            Debug.Print("SD card ejected");
        }

        static void RemovableMedia_Insert(object sender, MediaEventArgs e)
        {
            Debug.Print("SD card inserted");

            if (e.Volume.IsFormatted)
            {
                Debug.Print("Available folders:");
                string[] strs = Directory.GetDirectories(e.Volume.RootDirectory);
                for (int i = 0; i < strs.Length; i++)
                    Debug.Print(strs[i]);

                Debug.Print("Available files:");
                strs = Directory.GetFiles(e.Volume.RootDirectory);
                for (int i = 0; i < strs.Length; i++)
                    Debug.Print(strs[i]);
            }
            else
            {
                Debug.Print("SD card is not formatted");
            }
        }

        public static void SDMountThread()
        {
            PersistentStorage sdPS = null;
            const int POLL_TIME = 500; // check every 500 millisecond

            bool sdExists;
            while (true)
            {
                try // If SD card was removed while mounting, it may throw exceptions
                {
                    sdExists = PersistentStorage.DetectSDCard();

                    // make sure it is fully inserted and stable
                    if (sdExists)
                    {
                        Thread.Sleep(50);
                        sdExists = PersistentStorage.DetectSDCard();
                    }

                    if (sdExists && sdPS == null)
                    {
                        sdPS = new PersistentStorage("SD");
                        sdPS.MountFileSystem();
                    }
                    else if (!sdExists && sdPS != null)
                    {
                        sdPS.UnmountFileSystem();
                        sdPS.Dispose();
                        sdPS = null;
                    }
                }
                catch
                {
                    if (sdPS != null)
                    {
                        sdPS.Dispose();
                        sdPS = null;
                    }
                }

                Thread.Sleep(POLL_TIME);
            }
        }
    }
}
```

Option 2: There is an SD detect pin on the card socket on several platforms like ChipworkX and FEZ Cobra. FEZ Domino and FEZ Mini do not have the pin and you have to use option 1. The pin state changes when a card is inserted or ejected. Connect the SD card detect pin from the SD socket to an I/O pin on your hardware and read it to check if an SD card exists. Just copy this example and drop in any project you have to get insert and eject events. Make sure to change the I/O pin number to whatever I/O pin is actually connected to the detect pin.

```
using System;
using System.IO;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.IO;
using Microsoft.SPOT.Hardware;

using GHIElectronics.NETMF.IO;
using GHIElectronics.NETMF.Hardware;

namespace AutoMount
{
    public class Program
    {
        public static void Main()
        {
            RemovableMedia.Insert += new InsertEventHandler(RemovableMedia_Insert);
            RemovableMedia.Eject += new EjectEventHandler(RemovableMedia_Eject);

            // Set the SD detect pin
            sdDetectPin = new InputPort(EMX.Pin.IO36, false, Port.ResistorMode.PullUp);

            // Start auto mounting thread
            new Thread(SDMountThread).Start();

            // Your program goes here
            // ...

            Thread.Sleep(Timeout.Infinite);
        }

        static void RemovableMedia_Eject(object sender, MediaEventArgs e)
        {
            Debug.Print("SD card ejected");
        }

        static void RemovableMedia_Insert(object sender, MediaEventArgs e)
        {
            Debug.Print("SD card inserted");

            if (e.Volume.IsFormatted)
            {
                Debug.Print("Available folders:");
                string[] strs = Directory.GetDirectories(e.Volume.RootDirectory);
                for (int i = 0; i < strs.Length; i++)
                    Debug.Print(strs[i]);

                Debug.Print("Available files:");
                strs = Directory.GetFiles(e.Volume.RootDirectory);
                for (int i = 0; i < strs.Length; i++)
                    Debug.Print(strs[i]);
            }
            else
            {
                Debug.Print("SD card is not formatted");
            }
        }

        static InputPort sdDetectPin;
        public static void SDMountThread()
        {
            PersistentStorage sdPS = null;
            const int POLL_TIME = 500; // check every 500 millisecond

            bool sdExists;
            while (true)
            {
                try // If SD card was removed while mounting, it may throw exceptions
                {
                    sdExists = sdDetectPin.Read() == false;

                    // make sure it is fully inserted and stable
                    if (sdExists)
                    {
                        Thread.Sleep(50);
                        sdExists = sdDetectPin.Read() == false;
                    }

                    if (sdExists && sdPS == null)
                    {
                        sdPS = new PersistentStorage("SD");
                        sdPS.MountFileSystem();
                    }
                    else if (!sdExists && sdPS != null)
                    {
                        sdPS.UnmountFileSystem();
                        sdPS.Dispose();
                        sdPS = null;
                    }
                }
                catch
                {
                    if (sdPS != null)
                    {
                        sdPS.Dispose();
                        sdPS = null;
                    }
                }

                Thread.Sleep(POLL_TIME);
            }
        }
    }
}
```

23.2 USB Mass Storage

Files are handled on USB exactly the same way it is done on SD. The only difference is in how we detect a USB device and mount it. For SD, we could use an input pin to detect the card. On USB, we use events to detect a new media.

```
using System;
using System.IO;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.IO;
using GHIElectronics.NETMF.IO;
using GHIElectronics.NETMF.USBHost;

namespace Test
{
    class Program
    {

        // Hold a static reference in case the GC kicks in and disposes it
        // automatically, note that we only support one in this example!
        static PersistentStorage ps;

        public static void Main()
        {
            // Subscribe to RemovableMedia events
            RemovableMedia.Insert += RemovableMedia_Insert;
            RemovableMedia.Eject += RemovableMedia_Eject;

            // Subscribe to USB events
            USBHostController.DeviceConnectedEvent += DeviceConnectedEvent;

            // Sleep forever
            Thread.Sleep(Timeout.Infinite);
        }

        static void DeviceConnectedEvent(USBH_Device device)
        {
            if (device.TYPE == USBH_DeviceType.MassStorage)
            {
                Debug.Print("USB Mass Storage detected...");
                ps = new PersistentStorage(device);
                ps.MountFileSystem();
            }
        }

        static void RemovableMedia_Insert(object sender, MediaEventArgs e)
        {
            Debug.Print("Storage \"" + e.Volume.RootDirectory +
                        "\" is inserted.");
            Debug.Print("Getting files and folders:");
            if (e.Volume.IsFormatted)
            {
                string[] files = Directory.GetFiles(e.Volume.RootDirectory);
                string[] folders =
                           Directory.GetDirectories(e.Volume.RootDirectory);

                Debug.Print("Files available on " +
                            e.Volume.RootDirectory + ":");
                for (int i = 0; i < files.Length; i++)
                    Debug.Print(files[i]);

                Debug.Print("Folders available on " +
                            e.Volume.RootDirectory + ":");
                for (int i = 0; i < folders.Length; i++)
                    Debug.Print(folders[i]);
            }
            else
            {
                Debug.Print("Storage is not formatted. Format on PC with
                            FAT32/FAT16 first.");
            }
        }

        static void RemovableMedia_Eject(object sender, MediaEventArgs e)
        {
            Debug.Print("Storage \"" + e.Volume.RootDirectory +
                        "\" is ejected.");
        }
    }
}
```

You can see from the code above how after we mount the USB drive to the file system, everything work exactly the same as SD cards.

23.3 File System Considerations

NETMF support for FAT File System is only capable of FAT32 and FAT16. A media formatted as FAT12 will not work. This shouldn't be an issue since FAT12 is no longer in use.

The file system does a lot of data buffering internally to speed up the file access time and to increase the life of the flash media. When you write data to a file, it is not necessary that the data is written on the card. It is probably saved somewhere in the internal buffers. To make sure the data is stored on the media, we need to ?flush? the data. Flushing or closing a file is the only way to guarantee that the data you are trying to write are now on the actual media. That is on file level. On media level, there are also information that may not take immediate effect. For example, if you delete a file and remove the card from the system, the file is probably not actually erased. To guarantee the file is erased (media is updated) you need to run VolumeInfo.FlushALL

Ideally, you would unmounted the media **before** it is removed from the system. This may not be always possible and so a flush on a file or a FlushAll on media will guarantee your data is saved so there is no lost data if the media was removed at some point.

24 German Networking

Networks are an essential part of our work and living. Almost every home is connected to a network (internet) and most businesses can't function without an internal network (LAN or WiFi) that is connected to an external network (internet). All these networks have a standard way for communication, they all use TCP/IP protocol. There are actually a few protocols that handle different tasks in the network DNS, DHCP, IP, ICMP, TCP, UDP, PPP...and many more! NETMF supports TCP/IP networks through standard .NET sockets. A socket is a virtual connection between 2 devices on a network.

GHI extended the TCP/IP support to cover PPP and WiFi. Through PPP, two devices can connect through serial connection. Serial connection can be a phone-line modem or a 3G/GPRS modem. With PPP, NETMF devices can connect to the internet using 3G cell-phone networks. It is also possible to connect two NETMF devices through wired or wireless (XBee/Bluetooth/others) serial connection. Also, with WiFi support, NETMF devices can connect to standard secure or unsecured wireless networks.

NETMF also support SSL for secure connection.

The support for networks is standard and complete (HTTP, SSL, Sockets...etc.) on EMX, Embedded Master and for ChipworkX.

24.1 USBizi Network Support

When it comes to ram-hungry networking, USBizi will require some external support. For example, USBizi wired networking is done through Wiznet W5100 Hardwired TCP/IP chip. When USBizi want to make a network connection, all it has to do is send a request to Wiznet W5100 of the connection an then the chip will do the rest. Same as for data transfer, where USBizi will hand the data to W5100 and then this chip will take care of the rest.

GHI Provide complete commercially-supported drivers for Wiznet W5100. The drivers have the same interface as standard .NET sockets.

Here is a TCP example

```
using System;
using System.Text;
using System.Threading;

using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;

using GHIElectronics.NETMF.FEZ;
using GHIElectronics.NETMF.Net;
using GHIElectronics.NETMF.Net.Sockets;
using GHIElectronics.NETMF.Net.NetworkInformation;

using Socket = GHIElectronics.NETMF.Net.Sockets.Socket;

/// <summary>
/// This is a simple web server. Given a request, it returns an HTML document.
/// The same document is returned for all requests and no parsing of
/// the request is done.
/// </summary>
public static class MySocketServer
{
    public static void Main()
    {
        const Int32 c_port = 80;

        byte[] ip = { 192, 168, 0, 200 };
        byte[] subnet = { 255, 255, 255, 0 };
        byte[] gateway = { 192, 168, 0, 1 };
        byte[] mac = { 43, 185, 44, 2, 206, 127 };

        WIZnet_W5100.Enable(SPI.SPI_module.SPI1,
                            (Cpu.Pin)FEZ_Pin.Digital.Di10,
                            (Cpu.Pin)FEZ_Pin.Digital.Di9, true);

        NetworkInterface.EnableStaticIP(ip, subnet, gateway, mac);
        NetworkInterface.EnableStaticDns(new byte[] { 192, 168, 0, 1 });

        Socket server = new Socket(AddressFamily.InterNetwork,
        SocketType.Stream, ProtocolType.Tcp);

        IPEndPoint localEndPoint = new IPEndPoint(IPAddress.Any, c_port);

        server.Bind(localEndPoint);
        server.Listen(1);

        while (true)
        {
            // Wait for a client to connect.
            Socket clientSocket = server.Accept();

            // Process the client request. true means asynchronous.
            new ProcessClientRequest(clientSocket, true);
        }
    }
    /// <summary>
    /// Processes a client request.
    /// </summary>
    internal sealed class ProcessClientRequest
    {
        private Socket m_clientSocket;
        /// <summary>
        /// The constructor calls another method to handle the request,
        /// but can optionally do so in a new thread.
        /// </summary>
        /// <param name="clientSocket"></param>
        /// <param name="asynchronously"></param>
        public ProcessClientRequest(Socket clientSocket,
                                    Boolean asynchronously)
        {
            m_clientSocket = clientSocket;
            if (asynchronously)
                // Spawn a new thread to handle the request.
                new Thread(ProcessRequest).Start();
            else ProcessRequest();
        }
        /// <summary>
        /// Processes the request.
        /// </summary>
        private void ProcessRequest()
        {
            const Int32 c_microsecondsPerSecond = 1000000;
            // 'using' ensures that the client's socket gets closed.
            using (m_clientSocket)
            {
                // Wait for the client request to start to arrive.
                Byte[] buffer = new Byte[1024];
                if (m_clientSocket.Poll(5 * c_microsecondsPerSecond,
                SelectMode.SelectRead))
                {
                    // If 0 bytes in buffer,
                    // then the connection has been closed,
                    // reset, or terminated.
                    if (m_clientSocket.Available == 0)
                        return;
                    // Read the first chunk of the request
                    // (we don't actually do anything with it).
                    Int32 bytesRead = m_clientSocket.Receive(buffer,
                    m_clientSocket.Available, SocketFlags.None);

                    // Return a static HTML document to the client.
                    String s =
                    "HTTP/1.1 200 OK\r\nContent-Type: text/html; charset=utf-8\r\n\r\n<html><head><title>.NET Micro Framework Web Server on U
                    + "<body><bold><a href=\"http://www.tinyclr.com/\">Learn more about the .NET Micro Framework with FEZ by clicking here</a
                    byte[] buf = Encoding.UTF8.GetBytes(s);
                    int offset = 0;
                    int ret = 0;
                    int len = buf.Length;
                    while (len > 0)
                    {
                        ret = m_clientSocket.Send(buf, offset, len,
                                                  SocketFlags.None);
                        len -= ret;
                        offset += ret;
                    }
                    m_clientSocket.Close();
                }
            }
        }
    }
}
```

Here is a UDP example

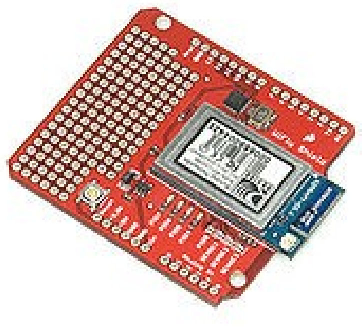
```
using System;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.FEZ;
using GHIElectronics.NETMF.Net;
using GHIElectronics.NETMF.Net.Sockets;
using GHIElectronics.NETMF.Net.NetworkInformation;
using System.Text;
using Socket = GHIElectronics.NETMF.Net.Sockets.Socket;
namespace FEZ_Panda_UDP
{
    public class Program
    {
        public static void Main()
        {
            byte[] ip = { 192, 168, 0, 200 };
            byte[] subnet = { 255, 255, 255, 0 };
            byte[] gateway = { 192, 168, 0, 1 };
            byte[] mac = { 43, 185, 44, 2, 206, 127 };
            // WIZnet interface on FEZ Panda
            WIZnet_W5100.Enable(SPI.SPI_module.SPI1,
                                (Cpu.Pin)FEZ_Pin.Digital.Di10,
                                (Cpu.Pin)FEZ_Pin.Digital.Di9, true);

            NetworkInterface.EnableStaticIP(ip, subnet, gateway, mac);
            NetworkInterface.EnableStaticDns(new byte[] { 192, 168, 0, 1 });

            Socket serversocket = new Socket(AddressFamily.InterNetwork,
                                             SocketType.Dgram,
                                             ProtocolType.Udp);

            EndPoint remoteEndPoint = new IPEndPoint(IPAddress.Any, 2000);
            serversocket.Bind(remoteEndPoint);
            int i = 1;
            while (true)
            {
                if (serversocket.Poll(-1, SelectMode.SelectRead))
                {
                    byte[] inBuf = new byte[serversocket.Available];
                    int count = serversocket.ReceiveFrom(inBuf,
                                                         ref remoteEndPoint);
                    Debug.Print(new String(Encoding.UTF8.GetChars(inBuf)));
                }
            }
        }
    }
}
```

Or see, http://www.tinyclr.com/downloads/Shield/Broch_EthernatShield.pdf

24.2 Raw TCP/IP vs. Sockets

This is an area when most designers miss few important points. When selecting a WiFi or GPRS/3G module, there are two main module categories to select from. Modules with sockets and modules with raw TCP/IP. The modules with sockets handle all TCP/IP work internally and give you a high level socket access. This means that all the work is being done already inside the module. Once you set your IP and port number, all you have to do is send and receive data from the module. The catch is, this is very limited. The module will have many limitations, like socket count. Even if your system is very powerful, with megabytes of RAM, you still is limited by the module's features. For this reason, using these high level modules is ideal for small systems.

Let me give you an example, Roving Networks provides a module called WiFly. This module has a built in TCP/IP stack and only one socket support. Using the module is very easy as you only need to connect it to one of the serial ports on your system then, with simple serial commands, you can read/write data from the one available socket. This is enough to make a simple web server or telnet connection, for configuration or data transfer. This is perfect for small systems with limited memory and low resources, like USBizi (FEZ). The module does everything for you, just send and receive data serially.



If you are implementing a web server that provides some values like temperature and humidity then this is all you need and can be easily implemented using USBizi (FEZ) at a very low cost. An easy prototype is done by connecting the WiFly shield to FEZ Domino. The SparkFun WiFly shield is showing in the image to the right.

The negative side about using these simple modules is that you are limited very few sockets, very few connections. What if you need more? What if you want to implement SSL (secure sockets)? For those, you will need a WiFi module that doesn't handle TCP/IP internally. The TCP/IP work will need to be done outside the module. This is where devices like EMX and ChipworkX come in. These devices are powerful, with a lot of resources. They have built in TCP/IP stack with SSL/HTTP/DHCP...etc. So connecting module like ZeroG to EMX or ChipworkX will empower the device with full blown and secure WiFi connection.

What about GPRS and 3G modems? The same applies to these modules. Some have built in sockets like SM5100B but others work over PPP just like any PC modem, like Telit modules for example. If you need a real network connection with full blown TCP/IP stack then you need EMX or ChipworkX with a standard PPP modems, just like how would your PC connect to the internet using a standard modem. If you need a simple and low-cost connection then USBizi (FEZ) can be used with SM5100B. The SparkFun Cellular Shield showing on the right, plugs right into FEZ Domino.

24.3 Standard .NET Sockets

The socket supported on NETMF is very similar to the full .NET Framework. The NETMF SDK includes many examples for using sockets, client and server. Also, many projects are available showing the different possibilities

Twitter client: http://www.microframeworkprojects.com/index.php?title=MFTwitter

Google maps: http://www.microframeworkprojects.com/index.php?title=GoogleMaps

RSS client: http://www.microframeworkprojects.com/index.php?title=RSS_n_Weather

MP3 internet radio: http://www.microframeworkprojects.com/index.php?title=ShoutcastClient

Web server: http://www.microframeworkprojects.com/index.php?title=WebServer

24.4 Wi-Fi (802.11)

WiFi is the most common way for computer networking. It allows secure data transfers at high rates to multiple connections. WiFi allows connections between two nodes, called Ad-Hoc. The more common way is to connect multiple computers to an access point. WiFi is not simple and not designed to be embedded friendly. GHI Electronics is the only company that offers WiFi option for its NETMF devices. WiFi is designed to work with TCP/IP stacks (network sockets on NETMF) and so it can only be used with systems that already support TCP/IP, like EMX and ChipworkX.

GHI's WiFi support uses ZeroG's ZG2100 and ZG2101 that use internal or external antennas respectively. For prototyping, the WiFi expansion board is a good start.

As explained in earlier section, USBizi (FEZ) can be used with modules with built in TCP/IP stack and socket support like the SparkFun shield with WiFly modules from Roving Networks.

24.5 GPRS and 3G Mobile Networks

EMX and ChipworkX have built in TCP/IP stack and PPP support. You can connect any standard modem and use it with few simple steps. As far as USBizi, a connection can be made to a mobile network using modems with built in TCP/IP like SM5100B. SparkFun Cellular shield with SM5100B is shown below.

25 German Cryptography

Cryptography has been an important part of technology for long years. Modern cryptography algorithms can be very resource hungry. Since NETMF is made with little devices in mind, the NETMF team had to be careful selecting what algorithms to support. The algorithms are XTEA and RSA.

25.1 XTEA

XTEA, with its 16byte (128bit) key, is considered to be very secure and at the same time it doesn't require a lot of processing power. Originally, XTEA was designed to work on chunks of eight bytes only. This can be a problem if encrypting data that it size is not multiple of 8. The implementation of XTEA on NETMF allows for encrypting data of any size. Encrypting and decrypting data is straight forward. Here is an example:

```
using System;
using System.Text;
using Microsoft.SPOT;
using Microsoft.SPOT.Cryptography;
public class Program
{
    public static void Main()
    {
        // 16-byte 128-bit key
        byte[] XTEA_key = new byte[] { 1, 2, 3, 4, 5, 6, 7, 8,
                                       9, 10, 11, 12, 13, 14, 15, 16 };
        Key_TinyEncryptionAlgorithm xtea =
                 new Key_TinyEncryptionAlgorithm(XTEA_key);
        // The data we want to encrypt
        string original_string = "FEZ is so easy!";
        //must be more than 8 bytes

        //convert to byte array
        byte[] original_data = UTF8Encoding.UTF8.GetBytes(original_string);

        //Encrypt the data
        byte[] encrypted_bytes = xtea.Encrypt(original_data, 0,
                                              original_data.Length, null);
        //Decrypt the data
        byte[] decrypted_bytes = xtea.Decrypt(encrypted_bytes, 0,
                                              encrypted_bytes.Length, null);
        //print the decrypted data
        string decrypted_string =
            new string(Encoding.UTF8.GetChars(decrypted_bytes));
        Debug.Print(original_string);
        Debug.Print(decrypted_string);
    }
}
```

The encrypted data has the same size as the unencrypted data.

25.1.1 XTEA on PCs

Now you can share data between NETMF devices securely using XTEA but what about sharing data with a PC or other systems? The book ?Expert .NET Micro Framework?, Second Edition by Jens Kuhner includes source examples showing how to implement XTEA on a PC. Using Jen's code, you can encrypt data then send to any NETMF device or vice-versa. Even if you do not own the book, the source code is online, found in chapter 8.

Download the source code from here http://apress.com/book/view/9781590599730

25.2 RSA

XTEA is very secure but has an important limitation, a key must be shared. For example, in order for any two systems to share encrypted data, they first both must share the same key. If a system tried to send the key to the other system then anyone spying on the data can get the key and use it to decrypt the data. Not very secure anymore!

RSA overcomes this problem by providing a private and public key combination. This may not make sense but the key used for encryption can't be used to decrypt the data. A different key is needed to decrypt the data. Lets say system 'A' needs to read some secure data from system 'B'. The first thing system 'A' will do is send a public key to system 'B'. System 'B' will now encrypt the data using the public key and send the encrypted data to the PC. A hacker can see the encrypted data and can see the public key but without the private key, decrypting the data is near impossible. Finally, system 'A' can decrypt the data with its private key.

By the way, this is how secure websites work.

NETMF devices can't generate keys. The keys are generated on a PC using a tool called MetaDataProcessor. Open the command prompt and enter this command cd ?C:\Program Files (x86)\Microsoft .NET Micro Framework\v4.0\Tools? Note that you may need to change this folder depending on where you have installed .NET Micro Framework.

Generate the keys using the following:

```
MetadataProcessor.exe -create_key_pair c:\private.bin c:\public.bin
```

The keys are in binary but the tool can convert the keys to readable text using these commands

```
MetadataProcessor.exe -dump_key c:\public.bin >> c:\public.txt
MetadataProcessor.exe -dump_key c:\private.bin >> c:\private.txt
```

Now, copy the key to our example program.

Note that the public key always starts with 1,0,1 and the rest are zeros. We can use this info to optimize our code as showing.

```
using System;
using System.Text;
using Microsoft.SPOT;
using Microsoft.SPOT.Cryptography;
public class Program
{
    public static void Main()
    {
        //this is shared between public and private keys
        byte[] module = new byte[] { 0x17, 0xe5, 0x27, 0x40, 0xa9, 0x15, 0xbd, 0xfa, 0xac, 0x45, 0xb1, 0xb8, 0xe1, 0x7d, 0xf7, 0x8b, 0x6c, 0x
        //the private key...for dycrypting
        byte[] private_key = new byte[] { 0xb9, 0x1c, 0x24, 0xca, 0xc8, 0xe8, 0x3d, 0x35, 0x60, 0xfc, 0x76, 0xb5, 0x71, 0x49, 0xa5, 0x0e, 0xd
        // the public key, always starts with 0x01, 0x00, 0x01,...
        // and the reast are all zeros
        byte[] public_key = new byte[128];
        public_key[0] = public_key[2] = 1;

        Key_RSA rsa_encrypt = new Key_RSA(module, public_key);
        Key_RSA rsa_decrypt = new Key_RSA(module, private_key);

        // The data we want to encrypt
        string original_string = "FEZ is so easy!";
        //convert to byte array
        byte[] original_data = UTF8Encoding.UTF8.GetBytes(original_string);
        //Encrypt the data
        byte[] encrypted_bytes =
           rsa_encrypt.Encrypt(original_data, 0, original_data.Length, null);
        //Decrypt the data
        byte[] decrypted_bytes =
           rsa_decrypt.Decrypt(encrypted_bytes, 0, encrypted_bytes.Length,
                               null);
        //print the decrypted data
        string decrypted_string =
            new string(Encoding.UTF8.GetChars(decrypted_bytes));
        Debug.Print("Data Size= " + original_string.Length +
                    " Data= " + original_string);
        Debug.Print("Encrypted Data size= " + encrypted_bytes.Length +
                    " Decrypted Data= " + decrypted_string);
    }
}
```

RSA encrypted data size is not the same as the raw data size. Keep this in mind when planning on transferring or saving RSA encrypted data.

RSA is far more processor intensive than XTEA which can be a problem for small systems. My suggestion is to start an RSA session to exchange XTEA key and some security info and then switch to XTEA.

26 German XML

26.1 XML in Theory

Extensible Markup Language (XML) is a standard for containing electronic data. When you want to transfer some info between two devices, you can set some rules on how the data is to be packed and sent from device A. On the other side, device B receives the data and knows how to unpack it. This created some difficulties in the past before XML. What if you were sending the data to a system implemented by different designer? You will have to explain the other designer how you have packed your data so he/she can unpack it. Now, designers can select to use XML to pack and unpack the data.

XML is extensively used daily in many ways. For example, when a website's shopping cart wants to know how much will be the shipping cost on a certain package, you will need to pack your shipment details in XML format and then send to FedEx (for example). Then FedEx website will read the info and send the cost info back in XML format as well.

The usefulness of XML can also be utilized in other ways. Let's say you are designing a data logger. Let's also assume the end users will need to configure the data logger to fit their needs. When a user configures the device, you need to store the info internally somehow. You can save the data with your own format which requires extra coding and debugging, or better just use XML. All GHI Electronics NETMF devices have built in XML reader and writer (packer and un-packer).

Here is an example XML file that will help in our data logger design.

```
<?xml version="1.0" encoding="utf-8" ?>
<NETMF_DataLogger>
  <FileName>Data</FileName>
  <FileExt>txt</FileExt>
  <SampleFreq>10</SampleFreq>
</NETMF_DataLogger>
```

The previous XML example includes a root element and three child elements. I chose for the file to look that way but you can, for example, make all info to be root elements. XML is very flexible, sometimes too flexible actually! Back to our example, the root element "NETMF_DataLogger" contains three pieces of info that are important for our logger. It contains the file name, the file extension and a frequency of our saved data. With this example, the logger will create a file called Data.txt and then will log data into that file 10 times every second.

Other important use for us "Embedded developers" is sharing data with the big system, I mean your PC. Since PCs with all operating systems do support XML in a way or another, you can send/receive data from the PC using XML.

Spaces and layout do not mean anything to XML, we (humans) need them to make things easier to read. The previous example can be stored without the spaces and layout like this.

```
<?xml version="1.0" encoding="utf-8" ?>
 <NETMF_DataLogger>
  <FileName>Data</FileName>
  <FileExt>txt</FileExt>
  <SampleFreq>10</SampleFreq>
 </NETMF_DataLogger>
```

See why spaces are important to us human being! You can also add comments inside XML files, comments do not mean anything to XML but can help in manual reading of the files

```
<?xml version="1.0" encoding="utf-8" ?>

<NETMF_DataLogger>
  <FileName>Data</FileName>
  <FileExt>txt</FileExt>
  <SampleFreq>10</SampleFreq>
</NETMF_DataLogger>
```

Finally, XML support attributes. An attribute is an extra info given to an element. But why do you need an attribute if you can add another element to describe this extra information? You really do not need to use attributes and I would say if you do not have a good reason then just do not use them. I will not be explaining attributes in this book

26.2 Creating XML

GHI Electronics' NETMF devices support reading and writing XML format. Reading and writing XML files work over streams which means any stream you already have or implement can work with XML. For example, we will use the built in MemoryStream and FileStream but you can create your own stream as well, which is not covered in this book.

This code shows how to make an XML document in memory. The code represent the our earlier XML example

```
using System.IO;
using System.Xml;
using System.Ext.Xml;
using Microsoft.SPOT;

public class Program
{
    public static void Main()
    {
        MemoryStream ms = new MemoryStream();

        XmlWriter xmlwrite = XmlWriter.Create(ms);

        xmlwrite.WriteProcessingInstruction("xml",
              "version=\"1.0\" encoding=\"utf-8\"");
        xmlwrite.WriteComment("This is just a comment");
        xmlwrite.WriteStartElement("NETMF_DataLogger");//root element
        xmlwrite.WriteStartElement("FileName");//child element
        xmlwrite.WriteString("Data");
        xmlwrite.WriteEndElement();
        xmlwrite.WriteStartElement("FileExt");
        xmlwrite.WriteString("txt");
        xmlwrite.WriteEndElement();
        xmlwrite.WriteStartElement("SampleFeq");
        xmlwrite.WriteString("10");
        xmlwrite.WriteEndElement();
        xmlwrite.WriteEndElement();//end the root element

        xmlwrite.Flush();
        xmlwrite.Close();
        //////// display the XML data ///////////
        byte[] byteArray = ms.ToArray();
        char[] cc = System.Text.UTF8Encoding.UTF8.GetChars(byteArray);
        string str = new string(cc);
        Debug.Print(str);
    }
}
```

Important note: On NETMF, XML writer and XML reader libraries are available from two different libraries. The XML reader comes from assembly "System.Xml" but the XML writer comes from "MFDpwsExtensions"! If you want to know why then you need to check with Microsoft! Also, the XML reader is in the "System.Xml" namespace but the XML writer is in "System.Ext.Xml". To make life easier, just include "System.Xml" and MFDpwsExtensions" assemblies whenever you need to get started with XML. Also your code should include the two needed namespaces just like I did in previous example.

Note: When you try to add an assembly you will notice that there are two assemblies for XML, the "System.Xml" and "System.Xml.Legacy". Never use the "legacy" driver, it is slow and needs a lot of memory. It is there for older systems that doesn't have a built in support for XML. All GHI Electronics; NETMF devices have a built in XML support (very fast!) and so you should always use "System.Xml"

When running the example above, we will see the output XML data at the end. The data is correct but it is not formatted to be "human" friendly. Note that we are reading and writing XML files on a very small system so the less info (spaces/formatting) the better it is. So it is actually better not to have any extra spaces or formatting but for the sake of making things look pretty, we will add new lines as follows

```
using System.IO;
using System.Xml;
using System.Ext.Xml;
using Microsoft.SPOT;

public class Program
{
    public static void Main()
    {
        MemoryStream ms = new MemoryStream();

        XmlWriter xmlwrite = XmlWriter.Create(ms);

        xmlwrite.WriteProcessingInstruction("xml",
            "version=\"1.0\" encoding=\"utf-8\"");
        xmlwrite.WriteComment("This is just a comment");
        xmlwrite.WriteRaw("\r\n");
        xmlwrite.WriteStartElement("NETMF_DataLogger");//root element
        xmlwrite.WriteString("\r\n\t");
        xmlwrite.WriteStartElement("FileName");//child element
        xmlwrite.WriteString("Data");
        xmlwrite.WriteEndElement();
        xmlwrite.WriteRaw("\r\n\t");
        xmlwrite.WriteStartElement("FileExt");
        xmlwrite.WriteString("txt");
        xmlwrite.WriteEndElement();
        xmlwrite.WriteRaw("\r\n\t");
        xmlwrite.WriteStartElement("SampleFeq");
        xmlwrite.WriteString("10");
        xmlwrite.WriteEndElement();
        xmlwrite.WriteRaw("\r\n");
        xmlwrite.WriteEndElement();//end the root element

        xmlwrite.Flush();
        xmlwrite.Close();

        //////// display the XML data ///////////
        byte[] byteArray = ms.ToArray();
        char[] cc = System.Text.UTF8Encoding.UTF8.GetChars(byteArray);
        string str = new string(cc);
        Debug.Print(str);
    }
}
```

26.3 Reading XML

Creating XML files is actually easier than parsing (reading) them. There are many ways to read the XML file but basically you can just go through the file and read one piece at the time till you reach the end. This code example creates an XML data and it reads it back.

```
using System.IO;
using System.Xml;
using System.Ext.Xml;
using Microsoft.SPOT;

public class Program
{
    public static void Main()
    {
        MemoryStream ms = new MemoryStream();

        XmlWriter xmlwrite = XmlWriter.Create(ms);

        xmlwrite.WriteProcessingInstruction("xml",
           "version=\"1.0\" encoding=\"utf-8\"");
        xmlwrite.WriteComment("This is just a comment");
        xmlwrite.WriteRaw("\r\n");
        xmlwrite.WriteStartElement("NETMF_DataLogger");//root element
        xmlwrite.WriteString("\r\n\t");
        xmlwrite.WriteStartElement("FileName");//child element
        xmlwrite.WriteString("Data");
        xmlwrite.WriteEndElement();
        xmlwrite.WriteRaw("\r\n\t");
        xmlwrite.WriteStartElement("FileExt");
        xmlwrite.WriteString("txt");
        xmlwrite.WriteEndElement();
        xmlwrite.WriteRaw("\r\n\t");
        xmlwrite.WriteStartElement("SampleFeq");
        xmlwrite.WriteString("10");
        xmlwrite.WriteEndElement();
        xmlwrite.WriteRaw("\r\n");
        xmlwrite.WriteEndElement();//end the root element

        xmlwrite.Flush();
        xmlwrite.Close();

        //////// display the XML data ///////////
        byte[] byteArray = ms.ToArray();
        char[] cc = System.Text.UTF8Encoding.UTF8.GetChars(byteArray);
        string str = new string(cc);
        Debug.Print(str);

        ///////////read xml
        MemoryStream rms = new MemoryStream(byteArray);

        XmlReaderSettings ss = new XmlReaderSettings();
        ss.IgnoreWhitespace = true;
        ss.IgnoreComments = false;
        //XmlException.XmlExceptionErrorCode.
        XmlReader xmlr = XmlReader.Create(rms,ss);
        while (!xmlr.EOF)
        {
            xmlr.Read();
            switch (xmlr.NodeType)
            {
                case XmlNodeType.Element:
                    Debug.Print("element: " + xmlr.Name);
                    break;
                case XmlNodeType.Text:
                    Debug.Print("text: " + xmlr.Value);
                    break;
                case XmlNodeType.XmlDeclaration:
                    Debug.Print("decl: " + xmlr.Name + ", " + xmlr.Value);
                    break;
                case XmlNodeType.Comment:
                    Debug.Print("comment " +xmlr.Value);
                    break;
                case XmlNodeType.EndElement:
                    Debug.Print("end element");
                    break;
                case XmlNodeType.Whitespace:
                    Debug.Print("white space");
                    break;
                case XmlNodeType.None:
                    Debug.Print("none");
                    break;
                default:
                    Debug.Print(xmlr.NodeType.ToString());
                    break;
            }
        }
    }
}
```

27 German Expanding IOs

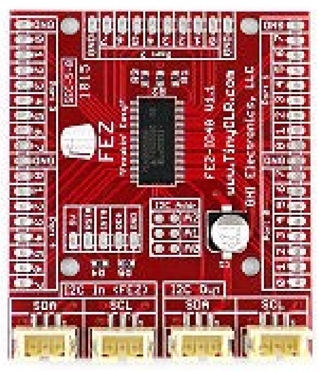
An application may require more digital pins or more analog pins than what is available on the processor. There are ways to expand what is available.

27.1 Digital

The easiest way to expand digital pins is by using a shift register. The shift register will then connect to the SPI bus. Using SPI, we can send or get the state of its pins. Shift registers can be connected in series so in theory we can have unlimited digital pins.

Shift registers usually have eight digital pins. If we connect three of them to a device over SPI, we will have 24 new digital pin but we only use the SPI pins on the processor.

Another good example is the IO40 board that runs on I2C bus. Also, this board can be chained for a maximum of 320 IOs!



27.1.1 Button Matrix

Devices like microwave ovens have many buttons on the front. A user will never need to press two buttons at the same time so we can ?matrix? those buttons. If we have 12 buttons on our system then we will need 12 digital inputs from the processor to read them all. Connecting these buttons in a 4x3 matrix will still give us 12 buttons but we are only using seven pins from the processor instead of 12. There are many off-the-shelf button matrix that can be integrated in your product.

To connect buttons in a matrix, we will wire our circuit so there are rows and columns. Each button will connect to one row and one column. That is all for hardware! Note how if we are not using a matrix then the button will connect to an input pin and ground.

To read the buttons state, make all processor pins connecting to rows outputs and the ones connecting to columns inputs. Set one and only one of the rows high and the rest of all rows to low. We are now selecting what row of buttons we will read. Now, read the state of all buttons in that row (you are reading the columns now). When complete, set the one row back to low and then go to the next one and set it high then go back to read the columns. Keep repeating until every row have been set to high once.

This example assumes we have a matrix with these connections. three rows connected to pins (1, 2, 3) and three columns connected to pins (4, 5, 6)

```
using System.Threading;
using System;
using Microsoft.SPOT.Hardware;
using Microsoft.SPOT;

namespace MFConsoleApplication1
{
    public class Program
    {
        static OutputPort[] Rows = new OutputPort[3];
        static InputPort[] Colms = new InputPort[3];

        static bool ReadMatrix(int row, int column)
        {
            bool col_state;

            //select a row
            Rows[row].Write(true);
            //read the column
            col_state = Colms[column].Read();
            // deselectt the row
            Rows[row].Write(false);

            return col_state;
        }
        static void Main()
        {
            // initialize rows to outputs and low
            Rows[0] = new OutputPort((Cpu.Pin)1, false);
            Rows[1] = new OutputPort((Cpu.Pin)2, false);
            Rows[2] = new OutputPort((Cpu.Pin)3, false);

            //initialize inputs with pull down
            Colms[0] = new InputPort((Cpu.Pin)4, true,
                                     Port.ResistorMode.PullDown);
            Colms[1] = new InputPort((Cpu.Pin)5, true,
                                     Port.ResistorMode.PullDown);
            Colms[2] = new InputPort((Cpu.Pin)6, true,
                                     Port.ResistorMode.PullDown);

            while (true)
            {
                bool state;

                // read the button on the first row and first column
                state = ReadMatrix(0, 0);//we count from zero
                Debug.Print("Buton state is: " + state.ToString());

                // read the button on the third row and second column
                state = ReadMatrix(2, 1);//we count from zero
                Debug.Print("Buton state is: " + state.ToString());

                Thread.Sleep(100);
            }
        }
    }
}
```

27.2 Analog

There are hundreds or thousands of analog chipsets available that run on SPI, I2C, one wire...etc. Some read 0V to 5V and other read -10V to +10V. Basically, there are too many options for reading more analog inputs to your device. Some chips have specific tasks. If we need to measure temperature, we can connect a temperature sensor to an analog pin and then read the analog value and convert that to temperature. That is an option but a better option will be to use a digital temperature sensor that run on I2C, one wire or SPI. This will give us a more accurate temperature reading and will also save an analog input for other uses.

27.2.1 Analog Buttons

One trick to connect many buttons using a single pin is by using an analog pin. The buttons will be connected to resistors. This allows each button to set out a different voltage if pressed. That voltage can then be measured to determine which button has been pressed.

28 German USB Client

I want to start this section by pointing out that this is a little advanced topic. If you just started out with NETMF then you may want to save it for later.

28.1 Serial (COM) Debugging

By default, all GHI's NETMF devices use USB for deploying and debugging. Optionally, a developer may want to use the USB client (not the host) for something other than debugging. This is actually supported by NETMF and GHI adds much functionality making it very easy to setup.

Let me give you an example on the USB client usage. Let us say you are making a device that that reads temperature and humidity...etc. and logs all this data on an SD card. Also, this device can be configured, like to set the time or give file names...etc. You want the device to be configured over USB. So when your device plugs into a USB port, you want it to show as a virtual serial port. This way, anyone can open a terminal software (like TeraTerm) to connect to your device to configure it. This is where USB client becomes very useful. There is no need to add extra cost to the design but adding additional RS232 serial ports or USB<->serial chipsets. The built-in USB client port can be configured to act as a CDC device, virtual COM port. But, there is a catch! The catch is, you still need to connect the PC to your device serially for debugging and deploying applications since the USB client port is used by your end application. The good news is that you only need the serial interface in the development stage but, when you deploy the product, the serial port is no longer needed. For example, you may use the RS232 shield on FEZ Domino in development stage but then you do not need it when you are done debugging.

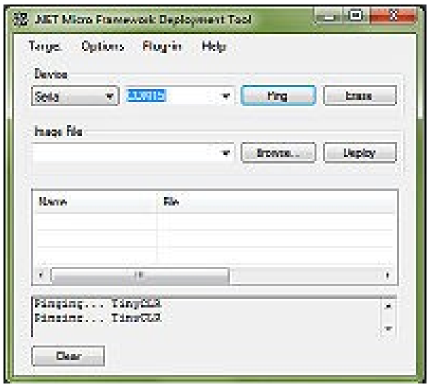
Now, you have COM1 connected to your device and want to use that for debugging instead of USB. The next step is to configure the device to use serial for debugging. This is a device specific so you need to check your device's user manual. For example, if you are using FEZ Domino, there is jumper labeled MODE that you can place to select serial debugging. If you are using FEZ Cobra then the pin LMODE need to be connected to 3.3V to select serial debugging. Remember that once you connect the pin high or low then you can no longer use it in your application. For example, the MODE pin on FEZ Domino is the same pin used for LED and it is also a PWM pin. Once you short the pin to ground (through the jumper), you should never attempt to use this pin, never try to use the LED or you may damage your device!

28.2 The Setup

Okay, enough talking and let us set up. I am using FEZ Domino with RS232 shield but you can use any device of your choice. I have the RS232 shield plugged into FEZ Domino and also connected to my PC using an RS232<->USB cable (no serial port on my PC). I also did place the MODE jumper and connected the USB cable from domino to this PC as well. After, placing the MODE jumper, connecting the USB cable from FEZ Domino to a PC will not load any drivers on windows (you will not hear the ?USB connect? sound).

Finally, we want to make sure we can ping the device using MFDeploy tool. We did that before using USB and now we want to use serial. Note that even though I have FEZ Domino+RS232 shield connected to my PC's USB port through the RS232<->USB cable, this is COM and not USB, it is a virtual COM to be exact.

So, open MFDeploy and select COM and then you will have a list of the available serial ports. Select the one connected to your device and click ping. You should see ?TinyCLR? back. If you don't then go back and check your setup.



```
Pinging... TinyCLR
Pinging... TinyCLR
```

28.3 Mouse, the Perfect Prank

Here is the master plan! You want to prank someone, get it video taped and then send me the funny video. This is how you do it. Setup your FEZ to emulate a USB mouse then make the mouse move in a circle every few minutes. I bet you it will feel like there is a ghost on your machine. The good news is that windows can have multiple mouse devices so FEZ will be hidden in the back.

```
using System;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.USBClient;
using GHIElectronics.NETMF.Hardware;
using GHIElectronics.NETMF.FEZ;

public class Program
{
    public static void Main()
    {
        FEZ_Components.LED led = new FEZ_Components.LED(FEZ_Pin.Digital.Di3);
        led.StartBlinking(100, 200);


        // Check debug interface
        if (Configuration.DebugInterface.GetCurrent() == Configuration.DebugInterface.Port.USB1)
            throw new InvalidOperationException("Current debug interface is USB. It must be changed to something else before proceeding. Refe

        // Start Mouse
        USBC_Mouse mouse = USBClientController.StandardDevices.StartMouse();

        // Move pointer in a swirl
        const int ANGLE_STEP_SIZE = 15;
        const int MIN_CIRCLE_DIAMETER = 50;
        const int MAX_CIRCLE_DIAMETER = 200;
        const int CIRCLE_DIAMETER_STEP_SIZE = 1;

        int diameter = MIN_CIRCLE_DIAMETER;
        int diameterIncrease = CIRCLE_DIAMETER_STEP_SIZE;
        int angle = 0;
        int factor;
        Random rnd = new Random();
        int i = 0;

        while (true)
        {
            // we want to do it every sometime randomely
            i = rnd.Next(5000) + 5000;//between 5 and 10 seconds
            Debug.Print("Delaying for " + i + " ms");
            Thread.Sleep(i);
            i = rnd.Next(200) + 100;//do it for a short time
            Debug.Print("Looping " + i + " times!");

            while (i-- > 0)
            {
                // Check if connected to PC
                if (USBClientController.GetState() ==
                    USBClientController.State.Running)
                {
                    // Note Mouse X, Y are reported as change in position
                    // (relative position, not absolute)
                    factor = diameter * ANGLE_STEP_SIZE *
                             (int)System.Math.PI / 180 / 2;
                    int dx = (-1 * factor *
                              (int)Microsoft.SPOT.Math.Sin(angle) / 1000);
                    int dy = (factor *
                              (int)Microsoft.SPOT.Math.Cos(angle) / 1000);

                    angle += ANGLE_STEP_SIZE;
                    diameter += diameterIncrease;
                    if (diameter >= MAX_CIRCLE_DIAMETER ||
                        diameter <= MIN_CIRCLE_DIAMETER
                        )
                        diameterIncrease *= -1;

                    // report mouse position
                    mouse.SendData(dx, dy, 0, USBC_Mouse.Buttons.BUTTON_NONE);
                }

                Thread.Sleep(10);
            }
        }
    }
}
```

28.4 Keyboard

Emulating a keyboard is as easy as emulating a mouse. The following example will create a USB Keyboard and send "Hello world!" to a PC every second.

```
using System;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;

using GHIElectronics.NETMF.USBClient;
using GHIElectronics.NETMF.Hardware;

public class Program
{
    public static void Main()
    {
        // Check debug interface
        if (Configuration.DebugInterface.GetCurrent() ==
            Configuration.DebugInterface.Port.USB1)
            throw new InvalidOperationException("Current debug interface is USB.
            It must be changed to something else before proceeding.
            Refer to your platform user manual to change the debug interface.");
        // Start keyboard
        USBC_Keyboard kb = USBClientController.StandardDevices.StartKeyboard();
        Debug.Print("Waiting to connect to PC...");
        // Send "Hello world!" every second
        while (true)
        {
            // Check if connected to PC
            if (USBClientController.GetState() ==
                USBClientController.State.Running)
            {
                // We need shift down for capital "H"
                kb.KeyDown(USBC_Key.LeftShift);
                kb.KeyTap(USBC_Key.H);
                kb.KeyUp(USBC_Key.LeftShift);
                // Now "ello world"
                kb.KeyTap(USBC_Key.E);
                kb.KeyTap(USBC_Key.L);
                kb.KeyTap(USBC_Key.L);
                kb.KeyTap(USBC_Key.O);
                kb.KeyTap(USBC_Key.Space);
                kb.KeyTap(USBC_Key.W);
                kb.KeyTap(USBC_Key.O);
                kb.KeyTap(USBC_Key.R);
                kb.KeyTap(USBC_Key.L);
                kb.KeyTap(USBC_Key.D);
                // The "!"
                kb.KeyDown(USBC_Key.LeftShift);
                kb.KeyTap(USBC_Key.D1);
                kb.KeyUp(USBC_Key.LeftShift);
                // Send an enter key
                kb.KeyTap(USBC_Key.Enter);
            }
            Thread.Sleep(1000);
        }
    }
}
```

28.5 CDC - Virtual Serial

Serial ports are the most common interface, especially on the embedded system world. This is an ideal solution for devices to transfer data between PCs and embedded devices (FEZ). To combine the popularity and usefulness of USB with the easiness of serial, we have virtual USB devices. To windows applications or to devices, a virtual serial port works just like a serial port but, in reality, it is actually a USB port. One important thing I want to mention here is that, usually, CDC drivers handle one transaction in every frame. The max EP size on USB is 64 bytes and there 1000 frames/seconds on full-speed USB. This means, the maximum transfer rate for CDC drivers is 64KB/sec. I think, Microsoft realized the needs for CDC and higher transfer rate and did enhance this limitation. Last time I tested the transfer speed on my win7 machine, I was able to see about 500KB/sec. The following will create a USB CDC and send "Hello world!" to PC every second.

```
using System;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;

using GHIElectronics.NETMF.USBClient;
using GHIElectronics.NETMF.Hardware;

public class Program
{
    public static void Main()
    {
        // Check debug interface
        if (Configuration.DebugInterface.GetCurrent() ==
            Configuration.DebugInterface.Port.USB1)
            throw new InvalidOperationException("Current debug interface is USB.
             It must be changed to something else before proceeding.
             Refer to your platform user manual to change the debug interface.");

        // Start CDC
        USBC_CDC cdc = USBClientController.StandardDevices.StartCDC();

        // Send "Hello world!" to PC every second. (Append a new line too)
        byte[] bytes = System.Text.Encoding.UTF8.GetBytes("Hello world!\r\n");
        while (true)
        {
            // Check if connected to PC
            if (USBClientController.GetState() !=
                USBClientController.State.Running)
            {
                Debug.Print("Waiting to connect to PC...");
            }
            else
            {
                cdc.Write(bytes, 0, bytes.Length);
            }
            Thread.Sleep(1000);
        }
    }
}
```

28.6 USB Debugging with Virtual COM Channel

There are cases where developers may prefer to use the same USB cable for deploying/debugging, and at the same time, enable CDC (Virual Serial) port on the same USB interface. GHI exclusively supports just that.

When running the code below, windows will ask for new driver, which are found at
http://www.ghielectronics.com/downloads/NETMF/Library%20Documentation/GHI_NETMF_Interface_with_CDC.zip

```
using System;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;

using GHIElectronics.NETMF.USBClient;
using GHIElectronics.NETMF.Hardware;

namespace USBClient_Example
{
    public class Program
    {
        public static void Main()
        {

            // Check debug interface
            if (Configuration.DebugInterface.GetCurrent() != Configuration.DebugInterface.Port.USB1)
                throw new InvalidOperationException("Interface must be USB.");

            // Start CDC
            USBC_CDC cdc = USBClientController.StandardDevices.StartCDC_WithDebugging();

            // Send "Hello world!" to PC every second. (Append a new line too)
            byte[] bytes = System.Text.Encoding.UTF8.GetBytes("Hello world!\r\n");
            while (true)
            {
                // Check if connected to PC
                if (USBClientController.GetState() != USBClientController.State.Running)
                {
                    Debug.Print("Waiting to connect to PC...");
                }
                else
                {
                    cdc.Write(bytes, 0, bytes.Length);
                }

                Thread.Sleep(1000);
            }
        }
    }
}
```

28.7 Mass Storage

One of the great GHI unique features of USB client is supporting Mass Storage Class (MSC). This feature allows access to the connected media right from USB. Let me explain this through some example. A data logger application needs to save data to an SD card or USB memory. When the user is done collecting data, they can plug the USB data logger to the PC and now the PC can detect the device as a mass storage device. The user can then transfer the files using standard operating system. Think of the device as a memory card reader. We can even enhance our logger where the USB client interface can be CDC to configure the device and later dynamically switch to MSC to transfer files.

Once very common question on GHI support is ?Why can't I access the media while the media is also accessed externally (from windows)??. I will try to explain this better. A media is accessed though a class called PersistentStorage. The PersistentStorage object can then be accessed by the internal file system or externally through the USB MSC. Okay, but why not both? It can't be both because files systems cash a lot of info about the media to speed up file access time. Accessing the media simultaneously will definitely cause corruption to the media and therefor, simultaneous access is not allowed. Note that you can easily switch back and forth between internal file system and USB MSC.

This example code assumes an SD card is always plugged in. It enables MSC showing the device (I am using FEZ Domino) as a card reader.

```
using System;
using System.IO;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;

using GHIElectronics.NETMF.USBClient;
using GHIElectronics.NETMF.IO;
using GHIElectronics.NETMF.Hardware;

namespace USBClient_Example
{
    public class Program
    {
        public static void Main()
        {
            // Check debug interface
            if (Configuration.DebugInterface.GetCurrent() ==
                Configuration.DebugInterface.Port.USB1)
                throw new InvalidOperationException("Current debug interface is
                USB. It must be changed to something else before proceeding.
                Refer to your platform user manual to change the debug
                interface.");

            // Start MS
            USBC_MassStorage ms =
                   USBClientController.StandardDevices.StartMassStorage();

            // Assume SD card is connected
            PersistentStorage sd;
            try
            {
                sd = new PersistentStorage("SD");
            }
            catch
            {
                throw new Exception("SD card not detected");
            }
            ms.AttachLun(0, sd, " ", " ");


            // enable host access
            ms.EnableLun(0);

            Thread.Sleep(Timeout.Infinite);
        }
    }
}
```

28.8 Custom Devices

The GHI USB client Support even allows you to control the USB client in anyway you like. This feature require good knowledge of USB. If you do not know what is EndPoint and Pipe then do not attempt to create custom devices. Also, it is very important to have the device configured correctly the first time it is plugged into windows. This is important because windows stores a lot of information in its registry. So, if you change the configuration of your device after you had it plugged into windows the first time then windows may not see the changes since it will be using the old configuration from its registry.

Basically, stay away from USB Client Custom Devices unless you really have good reason to use them and you have very good knowledge in USB and in windows drivers.

I am not going to explain Custom Devices any further but just wanted to clear out why I decided not to do so. The standard built in classes, explained earlier, should be enough to cover most of your needs.

29 German Low Power

Battery powered devices must limit power usage as much as possible. Devices may lower the power consumption in many ways:

1. Reduce processor clock
2. Shutdown the processor when system is idle (keep peripherals and interrupts running)
3. Shutdown specific peripherals
4. Hibernate the system

A NETMF device may optionally support any of these methods. Consult with your device's user manual to learn more about what is directly supported. In general, all GHI NETMF devices shutdown the processor when in idle state. For example, this will reduce the power on FEZ Rhino by about 45mA. You really do not have to do anything special, as soon as the system is idle, the processor is automatically shut off while interrupts and peripherals are still active. Also, all peripherals (ADC, DAC, PWM, SPI, UART...) are disabled by default to lower power consumption. They are automatically enabled once they are used. If all that is not enough, you can completely hibernate the system. Just remember that when the system is hibernating, it is not executing and peripherals are not functional. For example, data coming in on UART will NOT wake up the system. You will simply lose the incoming data. Waking up from hibernate is a system dependent feature, but usually specific pin(s) can be toggled to wake the system up.

Note: GHI realized that the standard NETMF power mode is not suitable for hibernate so a GHI specific method is implemented to handle the system hibernation.

Important Note: When the device is in hibernation, USB stops working. You can't step in code or access the device. If your program always puts the device in sleep then you will not be able to load a new program. Basically, lock out the device. You need to enter the boot loader to erase all to unlock the device from its hibernation!

Important Note: When you hibernate, the system clock stops (not RTC) so the NETMF time will be off. You will need to read the time from RTC and set the NETMF sytsem time after waking up from hibernation. Waking up from hibernate is only possible on few events. Check your device user manual or the library documentation for details. An easy way to wake up from hibernate is by using the alarm feature (when available). For example, FEZ Rhino includes the 32Khz clock needed for the built in RTC/alarm but FEZ Mini doesn't. See this link for more details

```
http://www.tinyclr.com/compare
```

In this example, I set the RTC to some random but valid time and then I blink an LED. Every three seconds, the device sets the alarm to wake up in 10 seconds and then it put the device into hibernation. The device will be completely dead (again, no USB debugging) but you should be able to see this on the LED. When the device wakes up, it continue on blinking the LED. At the end, the LED will blink for 3 seconds and then stops for 10 seconds.

```
using System;
using System.Threading;
using Microsoft.SPOT;
using GHIElectronics.NETMF.Hardware;
using GHIElectronics.NETMF.Hardware.LowLevel;

using GHIElectronics.NETMF.FEZ;
public class Program
{
    public static void Main()
    {
        //blink LED
        FEZ_Components.LED led = new FEZ_Components.LED(FEZ_Pin.Digital.LED);
        led.StartBlinking(100, 100);
        //set to any random but valid time
        RealTimeClock.SetTime( new DateTime(2010, 1, 1, 1, 1, 1));
        while (true)
        {
            Thread.Sleep(3000);//blink LED for 3 seconds
            RealTimeClock.SetAlarm(RealTimeClock.GetTime().AddSeconds(10));
            Debug.Print("Going to sleep for 10 seonds!");
            // sleep for 10 seconds
            Power.Hibernate(Power.WakeUpInterrupt.RTCAlarm);

            Debug.Print("Good Morning!");
        }
    }
}
```

Another option is to wake up on interrupt port. You have to be careful with this because any interrupt on any pin will cause this wake up. For example, the WiFi module on FEZ Cobra internally uses one of the interrupt pins and so this will wake up the system. You need to disable WiFi before hibernating. This is not the only trick that you need to be aware of! Look at the following or try it. It will not work!

```
using System;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.Hardware;
using GHIElectronics.NETMF.Hardware.LowLevel;

using GHIElectronics.NETMF.FEZ;
public class Program
{
    public static void Main()
    {
        //blink LED
        FEZ_Components.LED led = new FEZ_Components.LED(FEZ_Pin.Digital.LED);
        led.StartBlinking(100, 100);
        //setup the interrupt pin
        InterruptPort LDR = new InterruptPort((Cpu.Pin)0,false,
                                Port.ResistorMode.PullUp,
                                Port.InterruptMode.InterruptEdgeLow);
        while (true)
        {
            Thread.Sleep(3000);//blink LED for 3 seconds
            // sleep
            Power.Hibernate(Power.WakeUpInterrupt.InterruptInputs);
            //we get here when we wakeup
        }
    }
}
```

Why did the example above not work? When you create an interrupt (or input) pin, interrupts are enabled only if glitch filter is used or if an event handler is installed. So, to make the work above work, you only need to enable the glitch filter. Here is the code that works.

```
using System;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.Hardware;
using GHIElectronics.NETMF.Hardware.LowLevel;

using GHIElectronics.NETMF.FEZ;
public class Program
{
    public static void Main()
    {
        //blink LED
        FEZ_Components.LED led = new FEZ_Components.LED(FEZ_Pin.Digital.LED);
        led.StartBlinking(100, 100);
        //setup the interrupt pin with glitch filter enableled
        InterruptPort LDR = new InterruptPort((Cpu.Pin)0, true,
                                  Port.ResistorMode.PullUp,
                                  Port.InterruptMode.InterruptEdgeLow);
        while (true)
        {
            Thread.Sleep(3000);//blink LED for 3 seconds
            // sleep
            Power.Hibernate(Power.WakeUpInterrupt.InterruptInputs);
            //we get here when we wakeup
        }
    }
}
```

Another option is to install an event handler for the button like the following.

```
using System;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.Hardware;
using GHIElectronics.NETMF.Hardware.LowLevel;

using GHIElectronics.NETMF.FEZ;
public class Program
{
    public static void Main()
    {
        //blink LED
        FEZ_Components.LED led = new FEZ_Components.LED(FEZ_Pin.Digital.LED);
        led.StartBlinking(100, 100);
        //setup the interrupt pin
        InterruptPort LDR = new InterruptPort((Cpu.Pin)0, false,
                                 Port.ResistorMode.PullUp,
                                 Port.InterruptMode.InterruptEdgeLow);
        LDR.OnInterrupt += new NativeEventHandler(LDR_OnInterrupt);

        while (true)
        {
            Thread.Sleep(3000);//blink LED for 3 seconds
            // sleep
            Power.Hibernate(Power.WakeUpInterrupt.InterruptInputs);
            //we get here when we wakeup
        }
    }

    static void LDR_OnInterrupt(uint data1, uint data2, DateTime time)
    {
        // empty for now!
    }
}
```

Please note that using InputPort is as good as using the InterruptPort since, internally, the interrupts are used when glitch filter is enabled. Here is the example using InputPort instead of InterruptPort.

```
using System;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.Hardware;
using GHIElectronics.NETMF.Hardware.LowLevel;

using GHIElectronics.NETMF.FEZ;
public class Program
{
    public static void Main()
    {
        //blink LED
        FEZ_Components.LED led = new FEZ_Components.LED(FEZ_Pin.Digital.LED);
        led.StartBlinking(100, 100);
        //setup the interrupt pin
        InterruptPort LDR = new InterruptPort((Cpu.Pin)0, false,
                                    Port.ResistorMode.PullUp,
                                    Port.InterruptMode.InterruptEdgeLow);
        LDR.OnInterrupt += new NativeEventHandler(LDR_OnInterrupt);

        while (true)
        {
            Thread.Sleep(3000);//blink LED for 3 seconds
            // sleep
            Power.Hibernate(Power.WakeUpInterrupt.InterruptInputs);
            //we get here when we wakeup
        }
    }

    static void LDR_OnInterrupt(uint data1, uint data2, DateTime time)
    {
        // empty for now!
    }
}
```

This example will blink an LED when up and then when the LDR button is pressed, the system will go into deep sleep for 10 se This is an example for FEZ Domino (USBizi) that will go into deep sleep and wake up when I press a button??????????

```
using System;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.Hardware;
using GHIElectronics.NETMF.Hardware.LowLevel;

using GHIElectronics.NETMF.FEZ;
public class Program
{
    public static void Main()
    {
        //blink LED
        FEZ_Components.LED led = new FEZ_Components.LED(FEZ_Pin.Digital.LED);
        led.StartBlinking(100, 100);
        //setup the interrupt pin
        InputPort LDR = new  InputPort((Cpu.Pin)0, true,
                                Port.ResistorMode.PullUp);
        while (true)
        {
            Thread.Sleep(3000);//blink LED for 3 seconds
            // sleep
            Power.Hibernate(Power.WakeUpInterrupt.InterruptInputs);
            //we get here when we wakeup
        }
    }
}
```

Finally, you can wake up on more than one events. For example, you can wake up if a button is pressed or the alarm is fired.

```
Power.Hibernate(Power.WakeUpInterrupt.InterruptInputs |
 Power.WakeUpInterrupt.RTCAlarm);
```

30 German Watchdog

In embedded system world, devices are usually always running and with no user interaction. So if something went wrong, it would be very beneficial if we have an automatic reset button. Watchdog is your reset button!

30.1 System Execution Recovery

Assume you are making a smart vending machine that reports its inventory over the network. If your code raises an exception that wasn't handled properly then your program will end. Program ended means the vending machine no longer work. Someone will have to drive to the vending machine to reset it, or better, just use watchdog.

When you enable the watchdog, you give it a timeout to reset after. Then, you keep resetting the watchdog timer periodically. If the system locks up then the watchdog counter will reach the timeout, which in turn, will reset the system. Important note: GHI realized that the built-in watchdog in NETMF is not what customers need; therefore, GHI implemented its own version of watchdog. Do not use the watchdog from Microsoft.SPOT.Hardware, instead, use the watchdog from GHIElectronics.NETMF.Hardware.LowLevel. If both namespaces must be used then you will have ambiguity errors and so you need to specifically call out the full path of the watchdog you need. For example, instead of using Watchdog.Enable(timeout), use

```
GHIElectronics.NETMF.Hardware.LowLevel.Watchdog.Enable(timeout).
```

This example shows how to set the watchdog to 5 seconds timeout and create a thread to clear the watchdog every 3 seconds. Should anything go wrong, the device will reset in 5 seconds.

Important Note: Once you enable the Watchdog, it cannot be disabled. So you have to keep resetting the timeout. This is done to assure that no system corruption will disable watchdog accidentally.

```
using System;
using System.Threading;
using GHIElectronics.NETMF.Hardware.LowLevel;

public class Program
{
    public static void Main()
    {
        // Timeout 5 seconds
        uint timeout = 1000 * 5;

        // Enable Watchdog
        Watchdog.Enable(timeout);

        // Start a time counter reset thread
        WDTCounterReset = new Thread(WDTCounterResetLoop);
        WDTCounterReset.Start();

        // ....
        // your program starts here

        // If we exit the program,
        // the thread will stop working and the system will reset!
        Thread.Sleep(Timeout.Infinite);
    }

    static Thread WDTCounterReset;
    static void WDTCounterResetLoop()
    {
        while (true)
        {
            // reset time counter every 3 seconds
            Thread.Sleep(3000);

            Watchdog.ResetCounter();
        }
    }
}
```

You may be thinking, if the software locked up then how would the code that handles watchdog ever run? On low level, the watchdog is supported in hardware not software. This means that the counter and the and reset mechanism is done inside the processor, without the need for any software.

30.1.1 Limiting Time-Critical Tasks

Back to our vending machine example, but this time we want to handle a different possible problem. Since NETMF is not real time, tasks can take longer than expected. If a person stepped up to the vending machine and entered what they want to buy, the machine will now turn a motor on which in turn will push the item out to the user. Let us say this was timed so the motor has to be on for one second. Now, what if it happened that at the same time the motor is running another thread started using the SD card. Let us also assume that the card had some problem which caused 3 second delay in reading the SD card. The 3 seconds delay while the motor is running will result in 3 items being pushed out to the buyer, but we only wanted to push one! If we used watchdog and set it to one second, then the user will have one item and when time is exceeding one second, the vending machine will reset, which will stop the motor from pushing more items out. Here is a simple example.

```
// ....
// ....
// Timeout 1 second
uint timeout = 1000;

//....user buys something

// Enable Watchdog
Watchdog.Enable(timeout);
//turn motors on
//...
//stop motors
//...
// We don't need Watchdog anymore, we have to keep resetting the timeout
Watchdog.Enable(Watchdog.MAX_TIMEOUT);
WDTCounterReset.Start();    // See this in the earlier example

// ....
// your program starts here
// ....
// ....
```

30.1.2 Detecting Watchdog Cause

In some cases, you need to know if the system did reset because of a watchdog to log this info or run some recovery procedures. This is how it works

```
using System;
using System.Threading;
using Microsoft.SPOT;
using GHIElectronics.NETMF.Hardware.LowLevel;


public class Program
{
    public static void Main()
    {
        // you can read this flag ***ONLY ONCE*** on power up
        if (Watchdog.LastResetCause == Watchdog.ResetCause.WatchdogReset)
        {
            Debug.Print("Watchdog did Reset");
        }
        else
        {
            Debug.Print("Reset switch or system power");
        }
    }
}
```

31 Objects in Custom Heap

Note that this topic is about very large memory allocation and so it doesn't apply to USBizi.

Managed systems like NETMF requires very complex memory management. To limit the overhead on small systems, NETMF heap only supports allocating objects up to 700KBs. Larger objects are not possible. Beginning in NETMF version 4.0, larger buffers can be allocated using a separate heap called custom heap. Now, you can create very large buffers and very large bitmaps. Internally, these objects are not in the standard managed heap but they are in custom heap. This introduced a new important question. How much memory is reserved for custom heap and how much for managed heap?

GHI provides APIs allowing you to set the size of each heap, custom and managed.

```
using System;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.Hardware;
class Program
{

    public static void Main()
    {
        // set the heap size to 4MB
        if (Configuration.Heap.SetCustomHeapSize(4 * 1024 * 1024))
        {
            // this will only take effect after resetting the system
            PowerState.RebootDevice(false);
        }
        // ...
        // you can now use large objects up to 4MB
    }
}
```

31.1 Management of Custom Heap

Unlike the regular heap which is fully managed, the custom heap is managed but with few important points that need to be considered. In order for objects in custom heap to be cleared automatically, you must fulfill three requirements: The object reference must be lost. (This is all you need for regular objects). The garbage collector must run, you probably need to force it. The system must come idle in order for finalize to run and ?Dispose? object. I will not go into detail on garbage collectors, dispose, or finalizer...etc. What you need to know is that large object that are on custom heap are not easily cleared from the system so you must always dispose the object when you are done using it.

This is how you can dispose large objects

```
using System;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.Hardware;
class Program
{

    public static void Main()
    {
        // allocate a 1MB buffer
        LargeBuffer lb = new LargeBuffer(1024 * 1024);
        // use the buffer
        lb.Bytes[5] = 123;
        // ....
        // when done, dispose the object to empty the memory
        lb.Dispose();
    }
}
```

The better option is to use the ?using? statement. This statement will automatically call dispose once the execution is leaving the ?using? curly brackets.

```
using System;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.Hardware;
class Program
{
    public static void Main()
    {
        // allocate a 1MB buffer
        using (LargeBuffer lb = new LargeBuffer(1024 * 1024))
        {
            // use the buffer
            lb.Bytes[5] = 123;
            // ...
        }
        // Dispose was called automatically
        // ...
    }
}
```

31.2 Large Bitmaps

Bitmaps larger than 750KB/2 are automatically allocated on the custom heap. For example, a 800x480 bitmap will need 800x480x4 bytes, that is about 1.5MB. This large bitmap is on the custom heap and so you should always dispose it when you are done using it, or utilize the ?using? statement.

```
using System;
using Microsoft.SPOT;
using Microsoft.SPOT.Presentation.Media;
class Program
{
    public static void Main()
    {
        using(Bitmap largeBitmap = new Bitmap(800,480))
        {
            // assumign we have a display that is 800x480
            // draw a circle
            largeBitmap.DrawEllipse(Colors.Green, 100, 100, 10, 10);
            // draw other things
            // ...
            largeBitmap.Flush();//flush the Bitmap object to the display
        }
        // once we are here largeBitmap Dispose was automatically called
        // ...
    }
}
```

31.3 LargeBuffer

We already used LargeBuffer in earlier example. Only use this if you need to allocate a buffer that is larger than 750KB. On an embedded system, you shouldn't need or use such very large buffers. Although not recommended, this is available in case it is needed.

```
using System;
using Microsoft.SPOT.Hardware;
using GHIElectronics.NETMF.Hardware;
class Program
{
    public static void Main()
    {
        // allocate a 1MB buffer
        using (LargeBuffer lb = new LargeBuffer(1024 * 1024))
        {
            // use the buffer
            lb.Bytes[5] = 123;
            byte b = lb.Bytes[5];
        }
        // Dispose was called automatically
        // ...
    }
}
```

32 German Wireless

Wireless technologies are becoming an essential part of our life. Some applications require high rate data transfer, others require very low power. Some require a point to point connection, others require a mesh network. The biggest challenge when designing a wireless device is certification. Not only this needs to be done for different countries, it is very costly. You can easily spend 50,000 USD on certifying a wireless device. Fortunately, there are companies who offer certified wireless modules. When using a certified module, your product may not need any certification or the certification will be much easier.

32.1 Zigbee (802.15.4)

Zigbee is designed to be used in low-power battery-operated sensors. Many sensors can connected on one Zigbee network. Also, it requires very low power but data transfer rate is not very fast.

One very common implementation of Zigbee is XBee modules offered from Digi. There are many types of XBee modules. Some are very low power and other provides a very high output capable of transferring data up to 15 miles! Those modules are also offered on on-board antenna or with a connector for external antenna. The modules have a very simple interface that runs on UART. With UART, the modules can be interfaced to any NETMF device. If creating a connection between two modules then XBee modules establish a connection automatically. If connecting multiple nodes then we would need to send some serial commands to configure our network. I suggest you start with automatic point-to-point connection.

Connecting XBee to FEZ Mini or FEZ Domino can be easily done using the Xbee expansion component.

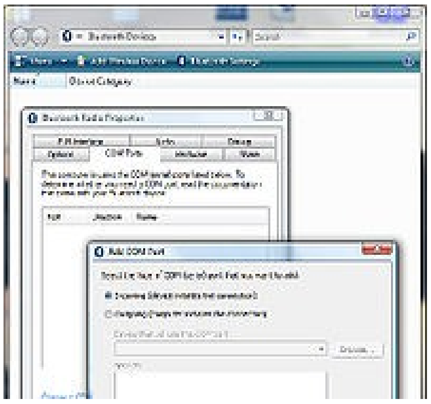
32.2 Bluetooth

Almost every cellphone has everything needed to connect to Bluetooth devices. Bluetooth technology defines multiple profiles for connections. The audio profile is useful to connect the Bluetooth ear-piece to the phone. For data transfers, SPP (Serial Port Profile) is used to establish a connection that simulates a serial connection. This is very similar to how XBee modules connect. Most phones have Bluetooth but most do not implement SPP so creating a serial connection from your phone is probably not possible.

On windows, we can create a serial Bluetooth connection with few clicks.

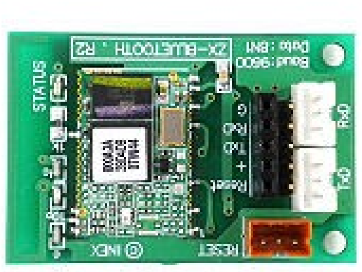
1. Find ?Bluetooth Devices? in ?Control Panel?
2. Click on ?Bluetooth Settings?
3. View the ?COM ports? tab
4. If you have COM ports installed then you already have SPP enables on your PC.
5. To add new ports, click on ?Add...?
6. Create one port for incoming data and one for outgoing.

Windows creates two ports, one for incoming and one for outgoing data. This can be confusing to Windows users because they can't use the same port to send and receive data!



On the embedded side, there are many serial modules that already include Bluetooth hardware and software, including SPP (Serial Port Profile). Those can easily connect to any NETMF device's serial port.

Connecting Bluetooth to FEZ Mini or FEZ Domino can be easily done using the Bluetooth interface component.



32.3 Nordic

Nordic semiconductor created it own digital wireless chips, NRF24L01. These low-power chips use 2.4Ghz which is a free band in many countries. Nordic wireless chips allow point-to- point or multi-point wireless connections.

Olimex offers breakout boards for NRF24L01. Those boards can connect directly to most GHI's NETMF boards.

This is a project (and video) showing two NETMF devices using NRF24L01 http://www.microframeworkprojects.com/index.php?title=SimpleWireless

33 German Managed Drivers

All NETMF devices provided by GHI have register access class for complete control over the underlaying hardware. But what is a "register"?

A register is a memory location (not memory) that is used to control a specific task. For example, there is a register for UART transmit. Whenever you write something in the UART transmit register, that data you write gets transfered out on the UART TX pin.

Let's say we have a power control register called PCONP and it is located at 0xE01FC0C4. Each bit of this register controls internal power to one of the internal peripherals. A logic one means the power is on. We want to also assume that we want to turn on timer2 and timer3 which are located at bits 22 and 23 (each register is 32-bit). My code should look like this

```
Register PCONP = new Register(0xE01FC0C4);
PCONP.SetBits((1 << 22) | (1 << 23));//enable timer2 and timer3
```

But why (1<<22)?

If I say the first bit then it is obvious it is 0x01 and the second would be 0x02...third 0x04. This is easy but what about 0x00400000? Which bit is this one? I do not want to even think about it. Instead, let the compiler do the work.

```
(1<<3) is exactly the same as (0x08)
```

33.1 Pulse counter

NETMF doesn't have a way to count pulses nor does GHI add such feature so how can I count pulses?

The processor used on most FEZ devices includes counters that can be controlled from an IO. I can use this hardware to count pulses. All I have to do is setup some registers then I can read the pulse count easily.

You need these files:

- Manual: http://www.keil.com/dd/docs/datashts/philips/lpc23xx_um.pdf
- Datasheet: http://www.nxp.com/documents/data_sheet/LPC2478.pdf
- EMX pinout: http://www.ghielectronics.com/downloads/EMX/EMX_Broch_Pinout.pdf

First, select a free timer. I know GHI used timer0 and possibly timer1 so 2 and 3 should be free. I will just use 3 and leave 2 for a sec.

Now, we need to select what pin on the processor to use for timer3 pin capture feature. According to datasheet, the pins are same as analog0 and analog1 but these are used for touch screen on FEZ Cobra(EMX). These pins would be perfect for Domino so maybe keep 2 options in your class where the user will select to use these pins on those devices.

Now for cobra, we will need to check timer2 which looks like it is connected to P0.4 and p0.5. What a good luck we have as P0.4 (IO0) is connected to the down button on FEZ Cobra. This means I can use the button to test what I am writing.

Take a look at the code below the processor user manual above to understand what registers can do and how they are controlled.

```
using System;
using System.Threading;
using Microsoft.SPOT;

using GHIElectronics.NETMF.Hardware.LowLevel;
using GHIElectronics.NETMF.Hardware;

namespace MFConsoleApplication1
{
    public class Program
    {
        public static void Main()
        {
            Register PCONP = new Register(0xE01FC0C4);
            PCONP.SetBits(1 << 22);//enable timer2

            // Select IO0 on EMX CAP2.0
            Register PINSEL0 = new Register(0xE002C000);
            PINSEL0.SetBits((3 << 8));//set bits 8 and 9

            // To enable timer/counter
            Register T2TCR = new Register(0xE0070004);
            T2TCR.Write(1);

            // set prescale to 0
            Register T2PR = new Register(0xE007000C);
            T2PR.Write(0);

            Register T2CTCR = new Register(0xE0070070);
            T2CTCR.Write(2 << 0 | 0 << 2);//count on falling edge and use CAPn.0

            // should be 0 for a counter
            Register T2CCR = new Register(0xE0070028);
            T2CCR.ClearBits(0x07);

            // Don't do anything on match
            Register T2MCR = new Register(0xE0070014);
            T2MCR.Write(0);

            // To reset the counter
            T2TCR.SetBits((1 << 1));
            T2TCR.ClearBits((1 << 1));

            // To read
            Register T2TC = new Register(0xE0070008);
            while (true)
            {
                uint count = T2TC.Read();

                Debug.Print("Total count: " + count);

                Thread.Sleep(1000);
            }
        }
    }
}
```

34 German Thinking Small

Many NETMF developers come from the PC world. They are used to write code that runs fine on a PC but then it will not run efficiently on an embedded device. The PC can be 4GHz with 4GB of RAM. NETMF devices have less than 1% of the resources available on a PC. I will cover different areas where you should always think small.

34.1 Memory Utilization

With limited RAM, developers should only use what they really need. PC programmers tend to make a large buffer to handle the smallest task. Embedded Developers study what they need and only allocate the needed memory. If I am reading data from UART, I can very well use 100 byte buffer to read the data and 1000 byte buffer will work as well. While I am analyzing the code, I noticed that I always read about 40 bytes from UART in my program loop. I do send a large buffer but I only get back 40 bytes. So, why would I want to use a buffer larger than 40 bytes? Maybe I will make it a bit large just in case but defiantly not 1000 bytes!

On some drivers, the NETMF system does a lot of buffering internally. For example, file system, UART, USB drivers all have internal buffers in native code, to keep the data ready until the developer uses the data from managed code. If we need a 1 megabyte file, we do not need a large buffer at all. We create a small buffer and then send the data in chunks to the file system. To play a 5 megabyte MP3 file, we only need 100 byte buffer that will read chunks from the file and pass to the MP3 decoder.

34.2 Object Allocation

Allocating and freeing objects is very costly. Only allocate objects when you really needs them. Also, you are making an embedded device; therefore, a lot of objects that you will be using are always used. For example, you will always use the LCD or always use the SPI. Consider the following code

```
using System.Threading;
using System;
using Microsoft.SPOT.Hardware;
using Microsoft.SPOT;

namespace MFConsoleApplication1
{
    public class Program
    {
        static void WriteRegister(byte register_num, byte value)
        {
            SPI _spi = new SPI(new
                  SPI.Configuration(Cpu.Pin.GPIO_NONE,false,0,0,false,
                  true,1000,SPI.SPI_module.SPI1));

            byte[] buffer = new byte[2];
            buffer[0] = register_num;
            buffer[1] = value;
            _spi.Write(buffer);
        }
        public static void Main()
        {
            WriteRegister(5, 100);
        }
    }
}
```

In order for me to write a single byte to a register on a SPI-chip, I had allocated SPI object, SPI.Configuration object and a byte array. Three objects for sending one byte! This is okay if you only need to do this a few times at initialization stage but if you are continuously using the WriteRegister method then this is not the right way. For starters, this method will run very slow so you wouldn't be able to ?WriteRegister? fast enough. Maybe this is sending graphics to the display or sending MP3 data to a decoder. This means that our function will be called few thousand times every second. As second problem is that these objects are created used and then left for the garbage collector to remove. The garbage collector will have to jump in and remove all these unused objects from memory which will stop the program execution for few milliseconds. Here is the code modified to test the method when called 1000 times.

```
using System.Threading;
using System;
using Microsoft.SPOT.Hardware;
using Microsoft.SPOT;

namespace MFConsoleApplication1
{
    public class Program
    {
        static void WriteRegister(byte register_num, byte value)
        {
            SPI _spi =
                new SPI(new SPI.Configuration(Cpu.Pin.GPIO_NONE,
                        false,0,0,false,
                        true,1000,SPI.SPI_module.SPI1));
            byte[] buffer = new byte[2];
            buffer[0] = register_num;
            buffer[1] = value;
            _spi.Write(buffer);
            _spi.Dispose();
        }
        public static void Main()
        {
            long ms;
            long ticks = DateTime.Now.Ticks;
            for (int i = 0; i < 1000; i++)
                WriteRegister(5, 100);
            ticks = DateTime.Now.Ticks - ticks;
            ms = ticks / TimeSpan.TicksPerMillisecond;
            Debug.Print("Time = " + ms.ToString());
        }
    }
}
```

When running the code on on FEZ (USBizi) we notice that the Garbage Collector had to run 10 times. The garbage collector prints it activity on the output window. Time taken for the code to run is 1911 ms, which is about 2 seconds! Now, let us modify the code as showing below. We now have the SPI object created globally and will always be there. We are still allocating the buffer in every loop.

```
using System.Threading;
using System;
using Microsoft.SPOT.Hardware;
using Microsoft.SPOT;

namespace MFConsoleApplication1
{
    public class Program
    {
        static SPI _spi = new SPI(new SPI.Configuration(
            Cpu.Pin.GPIO_NONE, false, 0, 0, false,
            true, 1000, SPI.SPI_module.SPI1));

        static void WriteRegister(byte register_num, byte value)
        {
            byte[] buffer = new byte[2];
            buffer[0] = register_num;
            buffer[1] = value;
            _spi.Write(buffer);
            _spi.Dispose();
        }
        public static void Main()
        {
            long ms;
            long ticks = DateTime.Now.Ticks;
            for (int i = 0; i < 1000; i++)
                WriteRegister(5, 100);
            ticks = DateTime.Now.Ticks - ticks;
            ms = ticks / TimeSpan.TicksPerMillisecond;
            Debug.Print("Time = " + ms.ToString());
        }
    }
}
```

In the second example, the garbage collector had to run only twice and it took only 448 milliseconds, about half a second to run. We only moved one line of code and it is 4 times faster. Let us move the buffer globally and see.

```
using System.Threading;
using System;
using Microsoft.SPOT.Hardware;
using Microsoft.SPOT;

namespace MFConsoleApplication1
{
    public class Program
    {
        static SPI _spi = new SPI(new SPI.Configuration(
            Cpu.Pin.GPIO_NONE, false, 0, 0, false,
            true, 1000, SPI.SPI_module.SPI1));
        static byte[] buffer = new byte[2];
        static void WriteRegister(byte register_num, byte value)
        {
            buffer[0] = register_num;
            buffer[1] = value;
            _spi.Write(buffer);
            _spi.Dispose();
        }
        public static void Main()
        {
            long ms;
            long ticks = DateTime.Now.Ticks;
            for (int i = 0; i < 1000; i++)
                WriteRegister(5, 100);

            ticks = DateTime.Now.Ticks - ticks;
            ms = ticks / TimeSpan.TicksPerMillisecond;
            Debug.Print("Time = " + ms.ToString());
        }
    }
}
```

We now have 368 milliseconds and garbage collector didn't run at all! One quick check you can do on your device is to check the output to see how often does the garbage collector run. On systems with large memory like ChipworkX this will not help much so you still need to analyze the code manually.

34.3 Missing Topics

These are topics not covered by this book. I will give a very quick review to cover what the topic is about. I may expand this book to cover these topics in future.

34.3.1 WPF

Windows Presentation Foundation is a new and flexible way to create graphical user interface applications. GHI's EMX and ChipworkX support WPF. USBizi and FEZ do not support this feature.

34.3.2 DPWS

Device Profile for Web Services allows networked devices to be automatically detected and used on the network. DPWS requires full .NET sockets support. Extended Weak Reference Extended Weak Reference(EWR) allows developers to save little data in nonvolatile memory. EWR is used more before File System was introduces to NETMF.

34.3.3 Serialization

Serialization is a way to convert an object to a series of bytes that represent that object. An object Mike made from a Human type can be serialized into a byte array and then this data is transferred to another device. The other device knows what a human type is but doesn't know anything about Mike. It will take this data to construct a new object based on it and it now has a copy of the object Mike. Using serialization in NETMF is a very bad idea unless you really needs it. There are few methods that will help you in extract or place values in array. You should be using these methods instead of serialization which is too slow.

34.3.4 Runtime Loadable Procedures

Runtime Loadable Procedures(RLP) is a GHI exclusive feature that allows users to write native (assembly/C) code for a device then load it and use it through managed (C#) at runtime. Native code is thousands times faster but it is not easy to manage. Specific tasks like calculating CRC that is a very processor intensive function are a perfect for for RLP. The complete application is made using manage code (C#) but then only CRC calculating method is written in native code (assembly/C).

34.3.5 Databases

A database stores data in a way where queering for data is easy. Looking up a product or sorting numbers is very fast because of the indexing databases do internally.

34.3.6 Touch Screen

NETMF supports touch screens. Touch screens are a good combination with TFT displays. A developer can create a graphical application using WPF and then the user can control it using the touch screen.

34.3.7 Events

If we have a project that receives data from serial ports, we need to read the serial port continuously. We may not have any data but we do not know will we check for the data. It will be more efficient if we can be notified if the serial driver had received data. This notification comes from an event that fires when the serial driver receives data. The same applies to interrupt ports covered before. This book doesn't cover the creation of events but we already seen how they are used in interrupt ports and used in using a mouse with USB host support.

34.3.8 USB Host Raw

We have learned how to access some USB devices using the GHI exclusive USB host support. GHI allows users to write managed drivers for almost any USB device. Accessing USB directly is considered a very advanced feature and is left out of this book.

This is a project that uses USB raw access to read an XBOX Controller: http://www.microframeworkprojects.com/index.php?title=Xbox_Controller
Another interesting project is the NXT drivers allowing users to control LEGO NXT Mindstorm right from C# and Visual Studio.
http://www.microframeworkprojects.com/index.php?title=NXT_Mindstorm

35 Final Words

If you found this book useful and it saved you few minutes of research then I have accomplished what I had in mind. I very much thank you for your downloading and reading this book.

35.1 Further Reading

This book only covers the basics of C# and .NET Micro Framework. This is a list of some resources to learn more:

- My blog is always a good place to visit

http://tinyclr.blogspot.com/

- The Micro Framework Project website is an excellent resource

http://www.microframeworkprojects.com/

- A good and free eBook to continue learning about C# is available at

http://www.programmersheaven.com/2/CSharpBook

- Jens Kuhner excellent book on .NET Micro Framework

http://www.apress.com/book/view/9781430223870

- USB complete is an excellent book on USB

http://www.lvr.com/usbc.htm

- Wikipedia is my favorite place for information about everything!

http://en.wikipedia.org/wiki/.NET_Micro_Framework

- .NET Micro Framework main page on Microsoft's website

http://www.microsoft.com/netmf

36 License & Disclaimer