Beginners Guide to C#

and the

.NET Micro Framework

September 24th 2012

Rev 2.1

www.GHIElectronics.com

Community: www.TinyCLR.com

Gus Issa

Table of Contents

1. About the Book

1.1. Intended Audience

This book is for beginners wanting to get started on .NET Micro Framework. No prior knowledge is necessary. The book covers the basics of .NET Micro Framework, Visual Studio and even C#!

If you're a hobbyist or an engineer, you will find a good deal of info in this book. This book makes no assumption about what you, the reader, knows so everything is explained extensively.

1.2. Disclaimer

This is a free book use it for your own knowledge and at your own risk. Neither the writer nor GHI Electronics is responsible for any damage or loss caused by this free eBook or by any information supplied by it. There is no guarantee any information in this book is valid.

2. Introduction

Have you ever thought of some great idea for a product but you couldn't bring it to life because technology wasn't on your side? Or maybe thought, “there's got to be an easier way!” Maybe you are a programmer that wanted to make a security system but then thought using a PC is too expensive to run a simple system? The answer is Microsoft's .NET Micro Framework!

Here is a scenario, you want to make a pocket-GPS-data-logger that saves positions, acceleration, and temperatures on a memory card and displays them on a small display. GPS devices can send position data over a serial port, so you can easily write some code on a PC to read the GPS data and save it on a file. But a PC won't fit in your pocket! Another problem is how would you measure temperature and acceleration on a PC? If you make this project using classic microcontrollers, like AVR, or PIC micro, all this can be done but then you need a compiler for the microcontroller you chose (which probably very expensive), a week to learn the processor, a week to write a serial driver, a month or more to figure out the FAT file system and more time for memory cards...etc. Basically, it can be done in few weeks or months of work.

2.1. Advantages

If you are using .NET Micro Framework then there are many advantages:

1. It runs on Microsoft's Visual C# Express, free and and high-end IDE.
2. .NET Micro Framework is open-source and free.
3. Your same code will run any NETMF device with almost no changes.
4. Full debugging capabilities. (Breakpoints, stepping in code, variables...etc.)
5. Has been tested in many commercial products, with assured quality.
6. Includes many bus drivers.(SPI, UART , I2C...etc.)
7. Eliminates the need to use complicated and long processors' datasheets.
8. If you are already a PC C# programmer then you are know NETMF.

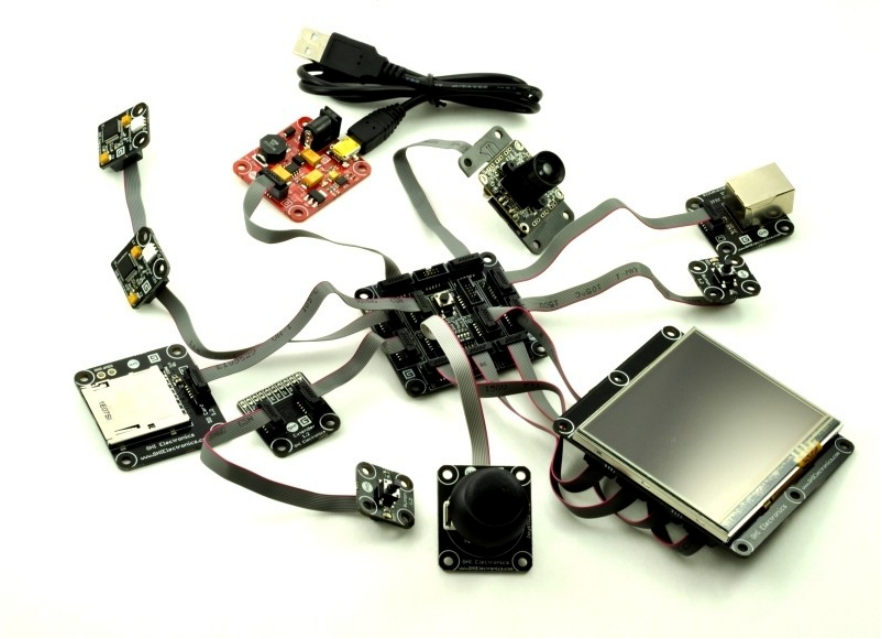
Throughout this document, I will refer to .NET Micro Framework as NETMF.

3. .NET Gadgeteer

.NET Gadgeteer platform definitions take NETMF's flexibility to the next level. It is a set of rules on how hardware modules can interact with a mainboard. The complete specifications and software are open-source. GHI Electronics is proud to the first to offer a .NET Gadgeteer platform.

More details are found at http://www.ghielectronics.com/products/dotnet-gadgeteer

This book is not meant to be used for Gadgeteer devices but it is a great tool to understanding NETMF, which is Gadgeteer's core engine.



4. Getting Started

Important note: If you have just received your hardware or you are not sure what firmware is loaded on it, you MUST update the firmware. This page summarizes the firmware update steps http://wiki.tinyclr.com/index.php?title=Firmware_Update

Also, make sure to read the “firmware/assembly matching” section in this book.

4.1. System Setup

Before we try anything, we want to make sure the PC is setup with needed software. First download and install **Visual C# express 2010**

http://www.microsoft.com/express/vcsharp/

Now, download and install .NET Micro Framework 4.2 SDK (not the porting kit) and the GHI package installer. Both are found on this page

http://www.ghielectronics.com/support/dotnet-micro-framework

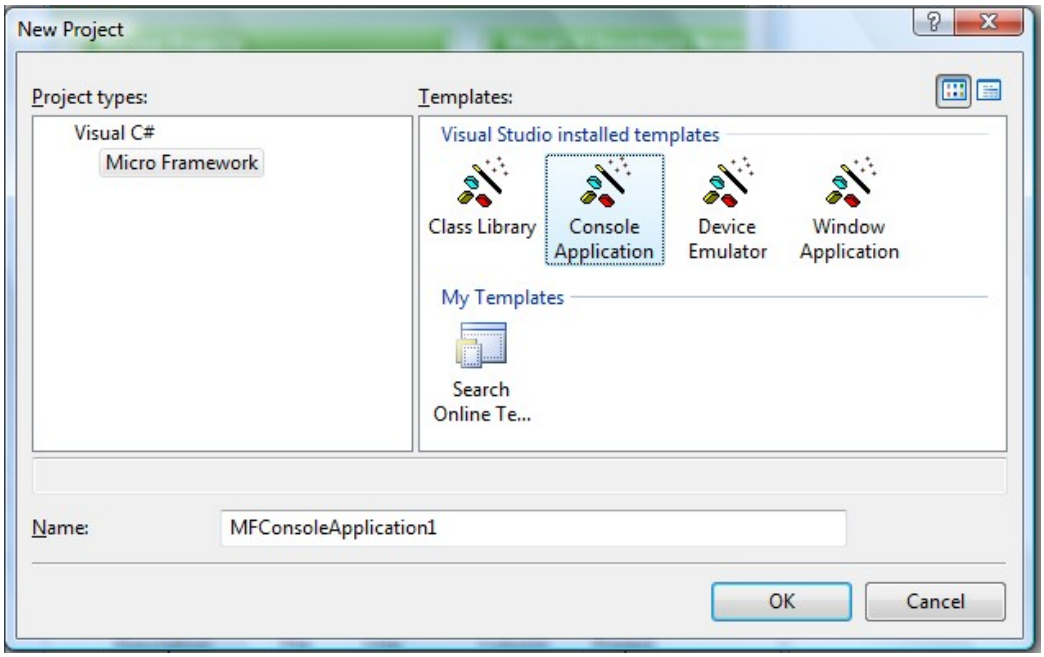
By the way, you may want to bookmark this page as it has about everything you need to use NETMF.

4.2. The Emulator

NETMF includes an emulator that allows you to run NETMF applications right on your PC. For our first project, we will use the emulator to run a very simple application.

Create a Project

Open Visual C# Express and, from the menu, select **file -> New Project.** The wizard should now have the “Micro Framework” option in the left menu. Click on it, and from the templates, select “Console Application”.



Click the “OK” button and you will have a new project that is ready to run. The project only has one C# file, called Program.cs, which contains very few lines of code. The file is shown in “Solution Explorer” window. If this window is not showing then you can open it by clicking “View->Solution Explorer” from the menu.

```
using System;
using Microsoft.SPOT;

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

    }
}
```

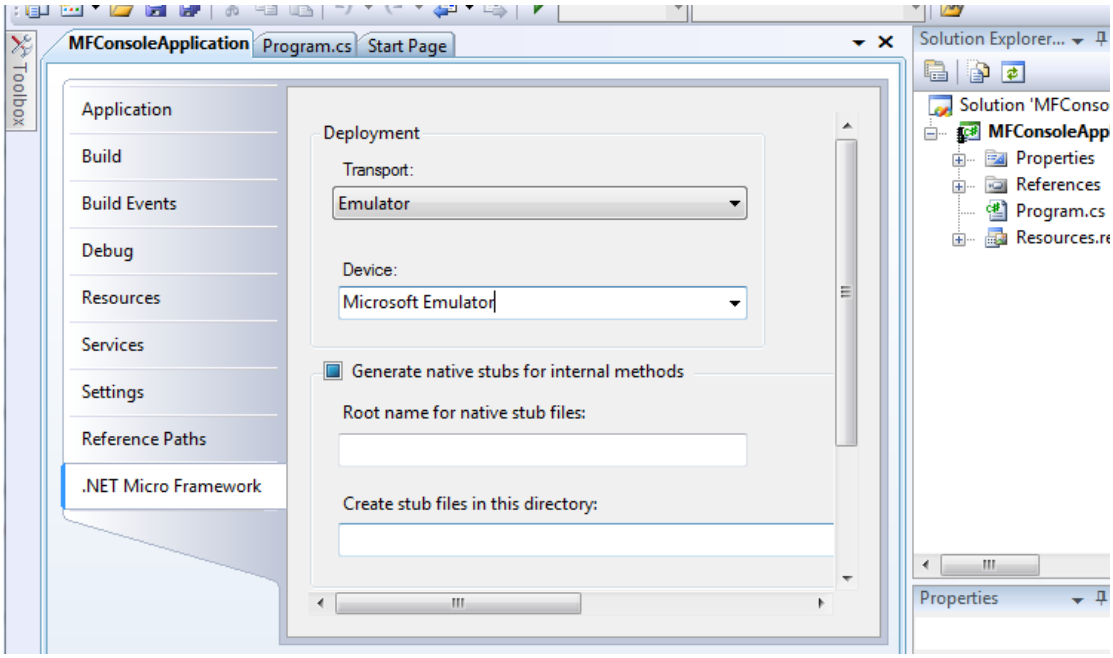
For simplicity change the code to look like the listing below

```
using System;
using Microsoft.SPOT;

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

    }
}
```

Selecting Transport

Don't worry if you do not understand the code. I will explain it later. For now, we want to run it on the emulator. Let's make sure you have everything setup properly. Click on “Project->Properties” from the menu. In the new showing window, we want to make sure we select the emulator. On the left side tabs, select “.NET Micro Framework” and make sure the window looks like the image below.

Transport: Emulator



Device: Microsoft Emulator

Make sure the output window is visible, click on

View->Output

If you are not seeing such option, then you need to enable expert settings first, found at

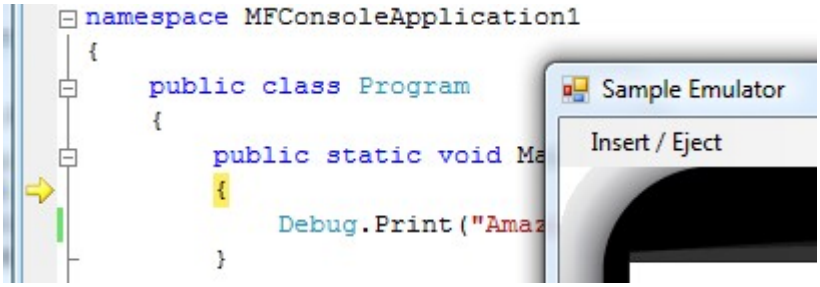
Tools-> Settings-> Expert Settings.

Executing

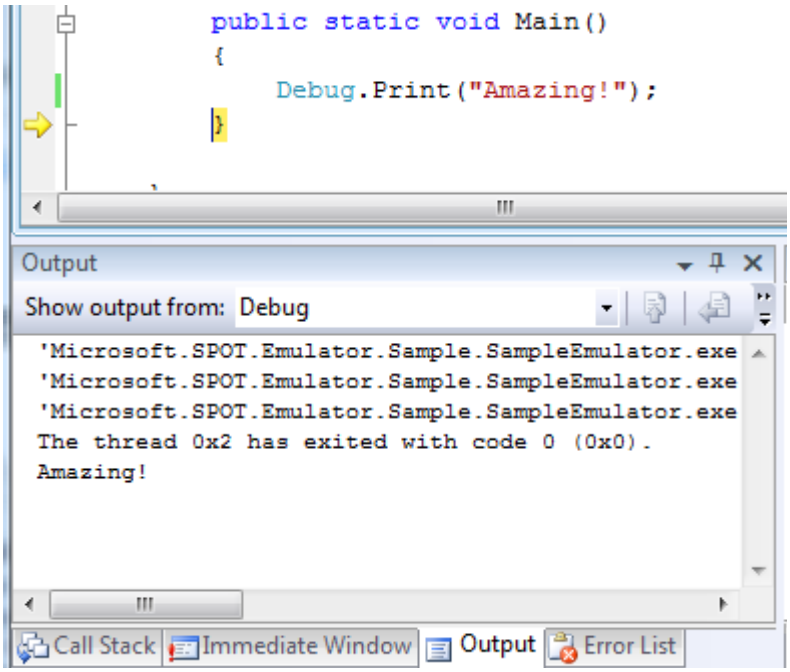
Finally, we are ready to run our first application. Press F5 key on the computer. This is a very useful shortcut and you will be using it a lot to run your applications. After you press F5, the application will be compiled and loaded on the emulator and in couple seconds everything will stop! That is because our program had finished execution so fast that we didn't see much.

We want to “debug” the code now. Debugging means that you are able to step in the code and see what it is doing. This is one of the greatest values of NETMF.

This time use F11 instead of F5, this will “step” in the application instead of just running it. This will deploy the application on the emulator and stop at the very first line of code. This is indicated by the yellow arrow.



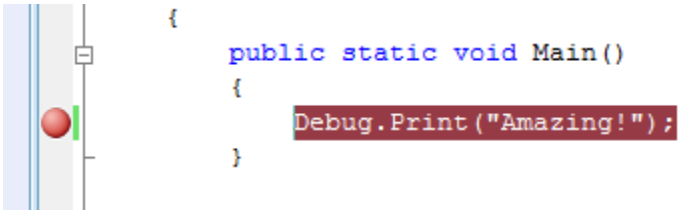
C# applications always start from a method called Main and this is where the arrow stopped. Press F11 again and the debugger will run the next line of code, which is the line you changed before. You probably have guessed it right, this line will print “Amazing!” to the debug window. The debug window is the output window on Visual C# Express. Make sure Output window is visible like explained earlier and press F11 one more time. Once you step on that line, you will see the word “Amazing!”, showing in the output window.



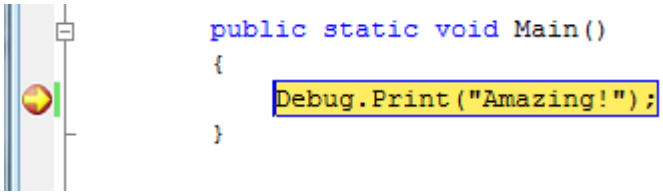
If you now press F11 again, the program will end and the emulator will exit.

Breakpoints

Breakpoints are another useful feature when debugging code. While the application is running, the debugger checks if execution has reached a breakpoint. If so, the execution will pause. Click the bar to the left of the line that prints “Amazing!” This will show a red dot which indicates a breakpoint.



Now press F5 to run the software and when the application reaches the breakpoint, the debugger will pause, as shown in the image below



Now, you can step in the code using F11 or continue execution using F5.

4.3. Running on Hardware

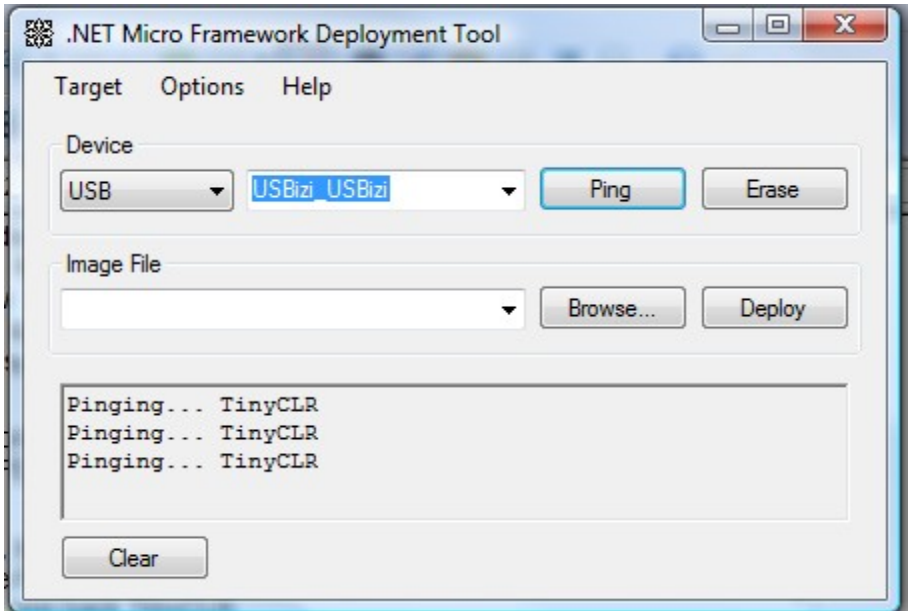
Running NETMF applications on hardware is very simple. Instructions can be slightly different on different hardware. This book uses FEZ for demonstration purposes but any other hardware will work similarly.

MFDeploy can Ping!

Before we use the hardware, let us make sure it is properly connected. The NETMF SDK comes with software from Microsoft called MFDeploy. There are many good uses for MFDeploy but for now we only need it to “ping” the device. Basically, “ping” means that MFDeploy will say “Hi” to the device and then checks if the device will respond with “Hi” back. This is good to make sure the device is connected properly and transport with it has no issues.

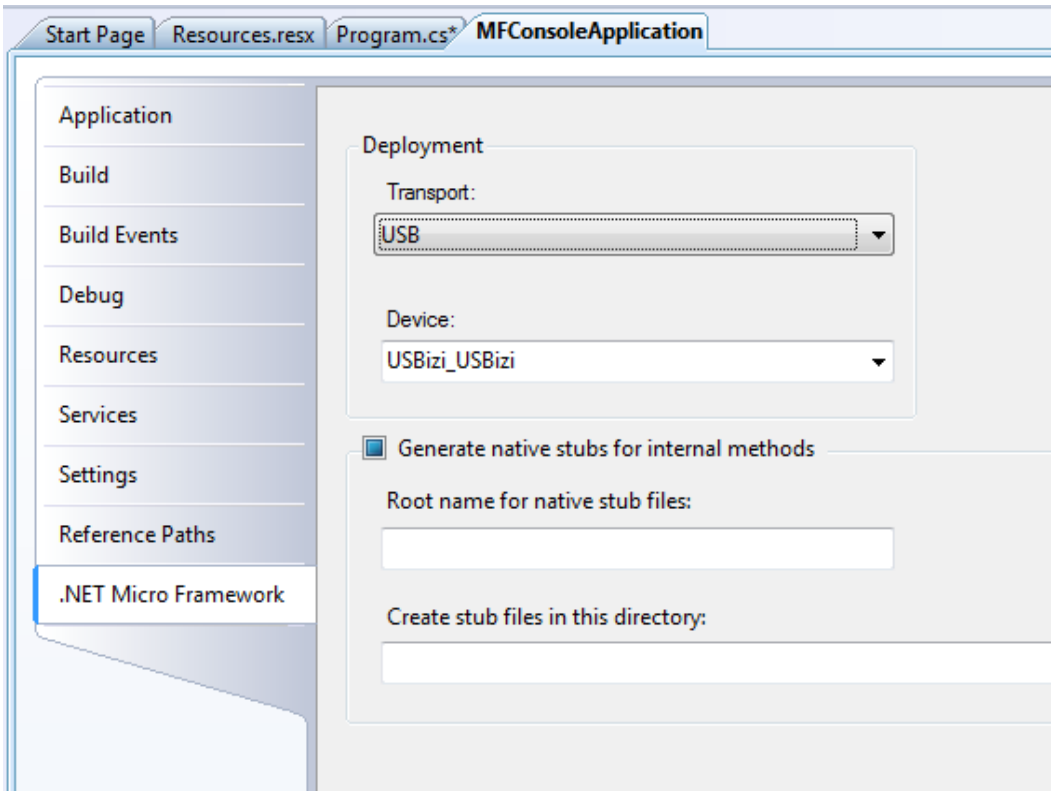
Open MFDeploy and connect FEZ using the included USB cable to your PC. If this is the first time you plugged in FEZ, Windows will look for the drivers and automatically find them. If not, supply the driver from the SDK folder and wait till windows is finished.

In the drop-down menu, select USB. You should see your device showing in the device list. In my example, I see USBizi. Select your device and click the “Ping” button. You should now see TinyCLR.



Deploying to Hardware

Now that we checked that the hardware is connected using MFDeploy, we need to go back to Visual Studio. From the project properties, select USB for transport and then your device (mine is USBizi). Make sure your setup looks similar to the image below.



Pressing F5 will now send our simple application to FEZ and it will run right inside the real hardware. Switching from emulator to real hardware is that simple!

Try the steps we did with the emulator, like setting breakpoints and using F11 to step in the code. Note that “Debug.Print” will still forward the debug messages from the hardware back to the output window on Visual Studio.

5. C# Level 1

This book is not meant to cover C# in details but I will cover most of basics to help you get started.

5.1. What is .NET?

.NET Framework was developed to standardize programming. (Note how I am talking about the full .NET Framework and not the **Micro** Framework.) Once you program in .NET, you are no longer concerned of the underlying operating system. It offers a set of libraries that developers can use from many programming languages.

The .NET Framework runs on PCs and not on smaller devices, because it is a very large framework. Also, the full framework has many things (methods) that wouldn't be very useful on smaller devices. This is how .NET Compact Framework was born. The compact framework removed unneeded libraries to shrink down the size of the framework. This smaller version runs on Windows CE and smart phones. The compact framework is smaller than the full framework but it is still too large for mini devices because of its size and because it requires an operating system to run.

.NET Micro Framework is the smallest version of those frameworks. It removed more libraries and it became OS independent. Because of the similarity among these three frameworks, almost the same code can now run on PCs and small devices, with little or no modifications.

For example, using the serial port on a PC, WinCE device or FEZ works the same way, when using .NET.

5.2. What is C#?

C and C++ are the most popular programming languages. C# is an updated and modernized version of C and C++. It includes everything you would expect from a modern language, like garbage collector and run-time validation. It is also object-oriented which makes programs more portable and easier to debug and port. Although C# puts a lot of rules on programming to shrink down the bug-possibilities, it still offers most of the powerful features C/C++ have.

“Main” is the Starting Point

Like we seen before, programs always start at a method called Main. A method is a little chunk of code that does a certain task. Methods start and finish with open/close curly brackets. In our first program, we only had one line of code between our curly brackets.

The line was Debug.Print("Amazing!");

You can see how the line ends with a semicolon. All lines must end the same way.

This line calls the Print method that exists in the Debug object. It calls it while passing the string “Amazing!”

Confused? Let's try to clear it out a bit. Let's say you are an object. You also have multiple methods to control you, the object. One method can be “Sit” and another can be “Run”. Now what if I want you to “Say” amazing? I will be calling your speak method with the sentence (string) “Amazing!”. So the code will look like:

You.Say(“Amazing!”);

Now, why do we need the quotes before and after the word Amazing? That is because C# doesn't know if the text you are writing is actually a command or it is actually text (strings). You can see how it is colored in red when you add quotes, which makes reading code easier for us, humans.

Comments

What if you want to add comments/notes/warnings in your code? Those comments will help you and others understand what the code means. C# completely ignores these comments. There are 2 ways to create comments, line comments and block comments. Comments (Ignored text) are shown in green.

To comment a line, or part of a line, add // before the comment text. The color of the text will change to green indicating that the text is now comment and is ignored by C#.

```
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!
        }

    }
}
```


You can also comment a whole block. Start the comment with /* and then end it with */ symbols

```
using System;
using Microsoft.SPOT;

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

    }
}
```

while-loop

It is time for our first keyword, “while”. The while-loop starts and ends with curly brackets to contain some code. Everything inside will continuously run while a statement is true. For example, I can ask you to keep reading this book “while” you are awake!

So, let's make a program that continuously prints “Amazing!” endlessly. This endless loop has no ending so it will always be “true”.

```
using System;
using Microsoft.SPOT;

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

    }
}
```


In the code above, execution will start at the “Main” method as usual and then it will go to the next line, which is the while-loop. The while-loop is telling the run time to execute the code inside its brackets while the statement is “true”. Actually, we do not have a statement there, but we have “true” instead which means this loop will always run.

Do not hit F5 to run the program or you will flood the output window with the word “Amazing!”. Instead, hit F11 and step in the code to understand how the loop works. Note that this program will never end so you will need to force stop using shift+F5.

Note: You can reach all these debug shortcuts from the menu under Debug.

Variables

Variables are places in memory reserved for your use. The amount of memory reserved for you depends on the type of the variable. I will not cover every single type here but any C# book will explain this in details.

We will be using an int variable. This type of variable is used to hold integer numbers.

Simply saying:

int MyVar;

will tell the system that you want some memory reserved for you. This memory will be referenced to as MyVar. You can give it any name you'd like, as long as the name doesn't contain spaces. Now, you can put any integer number into this memory/variable.

MyVar = 1234;

You can also use mathematical operations to calculate numbers:

MyVar = 123 + 456;

or you can increment the number by one:

MyVar++;

or decrement it by one:

MyVar- -;

With all that, can we make a program that prints the word 'Amazing!' three times. Here is the 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!");
            }
        }

    }
}
```

Notice how the while-loop statement is not “true” anymore, but it is MyVar>0. This means keep looping as long as MyVar's value is greater than 0.

In the very first loop MyVar is 3. Inside every loop, we decrement MyVar by one. This will result in the loop running exactly three times and therefore printing “Amazing!” three times.

Let's make things more interesting. I want to print the numbers 1 through 10. OK, we know how to make a variable and we know how to increment it but how do we print a number to the debug output window? Simply giving MyVar to Debug.Print will give you an error and it won't work. This is because Debug.Print will only accept strings, not integers. How do we convert an integer variable “ToString”? It is very simple, call MyVar.ToString(). That was easy!

```
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());
            }
        }

    }
}
```

Last thing to add is that we want to make the program print

