**Expert .NET Micro Framework**



ISBN-13: 978-1-59059-973-0

ISBN-10: 1-59059-973-X

ISBN-13 (electronic): 978-1-4302-0608-8

ISBN-10 (electronic): 1-4302-0608-X

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1



Lead Editor: Dominic Shakeshaft
Technical Reviewer: Fabio Claudio Ferracchiati
Editorial Board: Clay Andres, Steve Anglin, Ewan Buckingham, Tony Campbell, Gary Cornell, Jonathan Gennick, Matthew Moodie, Joseph Ottinger, Jeffrey Pepper, Frank Pohlmann, Ben Renow-Clarke, Dominic Shakeshaft, Matt Wade, Tom Welsh
Project Manager: Richard Dal Porto
Copy Editor: Heather Lang
Associate Production Director: Kari Brooks-Copony
Production Editor: Liz Berry
Compositor: Susan Glinert Stevens
Proofreader: Lisa Hamilton
Indexer: Becky Hornyak
Artist: April Milne
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail orders-ny@springer-sbm.com, or visit http://www.springeronline.com.

For information on translations, please contact Apress directly at 2855 Telegraph Avenue, Suite 600, Berkeley, CA 94705. Phone 510-549-5930, fax 510-549-5939, e-mail info@apress.com, or visit http://www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales—eBook Licensing web page at http://www.apress.com/info/bulksales.



The source code for this book is available to readers at http://www.apress.com. You may need to answer questions pertaining to this book in order to successfully download the code.

CHAPTER 1

# Introducing the .NET Micro Framework

This chapter introduces the .NET Micro Framework; we will discuss its history, motivation, goals, and architecture. We will have a look at where the .NET Micro Framework fits in the story of Microsoft's offerings for embedded development, and you will learn about the benefits and limitations of the .NET Micro Framework. Finally, the chapter provides a technical overview of the .NET Micro Framework.

## What Is the .NET Micro Framework?

The Microsoft .NET Micro Framework is a small and efficient .NET runtime environment used to run managed code on devices that are too small and resource constrained for Windows CE and the .NET Compact Framework.

The .NET Micro Framework enables you to write embedded applications for small, connected, embedded devices with Visual Studio and C#. That means you can now use the same development tools and language that you use to build desktop and smart device (PDA and smartphone) applications to develop applications for microcontrollers. The .NET Micro Framework also provides an extensible hardware emulator for rapid prototyping and debugging.

The .NET Micro Framework requires no underlying operating system. A scaled-down version of the Common Language Runtime (TinyCLR) sits directly on the hardware, so the framework is often called a bootable runtime. The runtime has a small footprint; it uses only a few hundred kilobytes of RAM and does not require the processor to have a memory management unit (MMU). Therefore, the .NET Micro Framework can run on small and inexpensive 32-bit processors without consuming a lot of power.

## .NET Micro Framework History

Let's take a look at the history and versions of the .NET Micro Framework:

- Smart Personal Object Technology (SPOT) started at Microsoft Research in 2001. David Massarenti developed the first proof-of-concept version of a scaled-down and ECMA-compliant CLR, the TinyCLR.
- Smart watches first shipped in 2004 (see Figure 1-1) and Microsoft TV set top boxes in 2005.

- The .NET Micro Framework 1.0 on Sumo robots (see Figure 1-2), which included a Sumo robot emulator, was presented at the 2006 Mobile and Embedded Developers Conference (MEDC); the conference also featured a Sumo robot contest.
- In February 2007, the .NET Micro Framework 2.0, with a customizable emulator, was released. Some development boards (see Figure 1-3) and devices (see Figure 1-4) were available, and others followed in 2007. You will learn more about the available development boards and hardware platforms in the next chapter.
- In 2007, Microsoft presented Windows SideShow, which is based on the .NET Micro Framework, and hardware manufacturers started shipping Windows SideShow-capable devices (see Figure 1-5).
- Later in 2007, Service Pack 1 for the .NET Micro Framework 2.0 was released.
- In February 2008, Microsoft released the .NET Micro Framework 2.5.

**Figure 1-1.** *A smart watch*

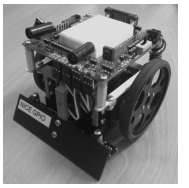

**Figure 1-2.** *A Sumo robot*

**Figure 1-3.** *A .NET Micro Framework development board (copyright of Freescale Semiconductor, Inc. 2008, used by permission)*

**Figure 1-4.** *A .NET Micro Framework network-enabled device*

**Figure 1-5.** *A notebook with an integrated display utilizing SideShow*

# Motivation

This section describes why Microsoft developed the .NET Micro Framework and the benefits of using managed code on devices.

## Embedded Development in the Past

Developing an embedded device from scratch and programming it with C++, C, or even assembly language can be challenging. Embedded developers are used to writing code that directly interfaces with the hardware. Running the software written for one controller on another platform will

not work, even if the CPU core is the same. Every board has different buses, interrupt controllers, memory, and I/O interfaces.

The tools and development environments for embedded development are not very comfortable and complete even today. Also, every CPU vendor provides its own compiler and development tools, and simulating hardware and debugging embedded applications on your development PC is hard.

## A Different Approach

Standard hardware platforms with the .NET Micro Framework already on board provide a different approach. The .NET Micro Framework hardware features ready-to-use single-board computers (see Chapter 2) with common hardware components such as memory, general purpose input/output (GPIO) ports, serial ports, and a display on board. The code to interface with the hardware components is already complete. You just need to write your application, and you can focus on domain-specific problems.

The .NET Micro Framework abstracts hardware access through its base class library and treats hardware components as objects. That enables you to program hardware components in an object-orientated way. Instead of dealing with hardware details and setting bit masks to configure peripheral hardware, you just need to set the properties of an object. This approach is also referred to as *managed drivers* and will help make your embedded application independent from a particular platform.

.NET Micro Framework applications can be programmed using Visual Studio and C#. Visual Studio is a widespread and state-of-the-art development tool. If you are already programming .NET applications for the desktop, smartphones, or PDAs, you can continue using your everyday development tool. C# is a modern high-level programming language that allows you to write clear and reusable object-orientated code. With the .NET Micro Framework, every developer familiar with .NET can also be an embedded developer!

Like the full .NET Framework and the .NET Compact Framework, the .NET Micro Framework runs managed code. The C# compiler generates a processor-independent intermediate language that TinyCLR executes on the device. The next section describes the benefits of managed code.

## Benefits of Managed Code

As stated earlier, the .NET Micro Framework contains the TinyCLR. Code that targets the CLR is referred to as *managed code*, whereas code that does not target the CLR is known as *unmanaged (native) code*. The CLR executes intermediate-language code and provides the following core services and benefits:

- Automatic memory management using a garbage collector
- Thread management and synchronization
- Exception handling
- Strict type safety
- Secure and robust managed code
- Debugging services

The CLR uses a garbage collector that automatically frees unused memory blocks and manages threads by providing time slices to the individual threads and methods to synchronize access to shared resources. Since managed code is executed under the control of the CLR, which takes care of references to objects, you do not have to deal with the unsafe pointers that are common with native programming. With managed code, you can't access memory blocks once they've been disposed of, because the CLR frees objects only when they are not referenced anymore. It also enforces strict type safety, and the TinyCLR prevents execution of unsafe, custom, native code. Together with exception handling, a modern way to handle errors, managed code enables secure and robust applications.

Managed code assemblies contain much information (metadata). Therefore, using managed code also allows you to use static code analysis tools like FxCop to detect and enforce the following to write better code:

- Enforcing consistent naming of classes and methods
- Detecting unused code
- Detecting performance issues
- Detecting design issues

## Where the .NET Micro Framework Fits

Now that we have discussed the benefits of managed code, let's look at where the .NET Micro Framework fits in the story of Microsoft's offerings (see Figure 1-6). Windows XP, Windows XP Embedded, and Windows Vista support the full .NET Framework, while Windows CE supports the .NET Compact Framework. There may be an overlap among the offerings, so two systems may fit well and be good choices for one application.

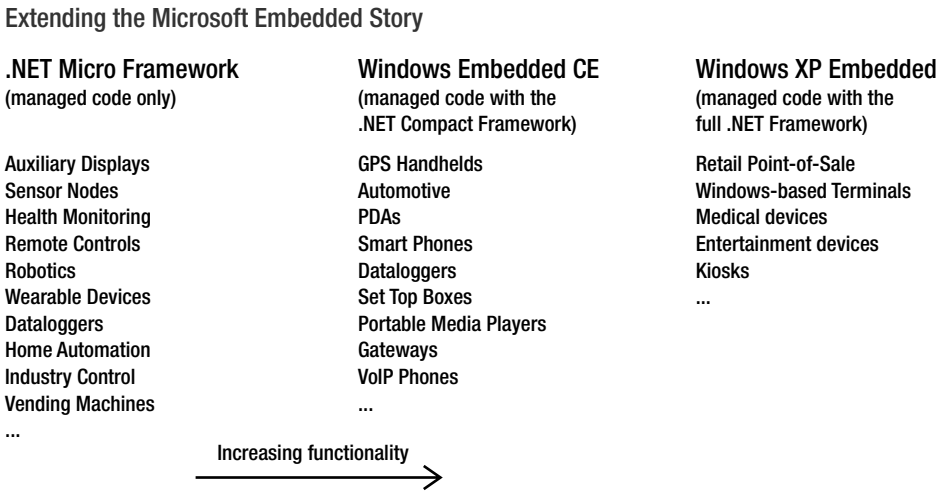


**Figure 1-6.** *Microsoft's embedded offerings*

The .NET Micro Framework is the smallest .NET platform to date and provides only a subset of the full .NET Framework and .NET Compact Framework. While Windows CE configurations supporting managed code with the.NET Compact Framework require at least 12MB of RAM, the .NET Micro Framework running directly on the hardware requires only about 300KB of RAM. The smart watches, for example, came with an ARM7 chipset clocked with 27 MHz and having 384KB RAM and 1MB flash memory (a suggested minimal configuration). The currently available .NET Micro Framework devices start at 55 MHz and require 8MB of RAM and 1MB of flash memory.

The .NET Micro Framework requires a 32-bit processor. Although common embedded programming in assembly language or C still takes place on 16-bit or even 8-bit processors, improved production techniques have reduced the manufacturing costs and power requirements of 32-bit processors. However, since the .NET Micro Framework does not require an MMU, it can run on less expensive processors than Windows CE can.

## What the Framework Is Not

The .NET Micro Framework is not a real-time system. Although it is very fast and suitable for most applications, do not expect real-time deterministic behavior. A timer event may not be triggered exactly after the specified amount of time (varying by some milliseconds), or the runtime environment may take a few milliseconds to react to an interruption by calling the interrupt service routine.

In addition, the garbage collector will run to free unused memory when memory gets low. Garbage collection might block all threads for a few milliseconds.

Also, executing managed code is a bit slower than executing native code. With the .NET Micro Framework, all managed code will be interpreted. There is no just-in-time compiler that compiles managed code parts to native machine code on the first execution as there is for the full .NET Framework and .NET Compact Framework.

## Licensing

The .NET Micro Framework SDK is available for download at no cost. To develop for the .NET Micro Framework, you need Visual Studio 2005 Standard Edition or better (not Express Edition). A fee is charged for each device containing the .NET Micro Framework, but the fee is lower than the one for Windows CE. Also, you are not required to pay the fee directly. When you buy ready-to-use prebuilt modules or development boards containing the .NET Micro Framework, the license fee has already been paid to Microsoft.

## Benefits of the .NET Micro Framework

For embedded development, the advantages of the .NET Micro Framework can be summarized as follows:

- Lower development costs than other embedded platforms
- Faster time to market than other embedded platforms
- Lower hardware costs than other managed platforms
- Smaller hardware size than other managed platforms
- Lower power consumption than other managed platforms

- Not bound to a specific chipset or board vendor
- Important part of Microsoft's embedded strategy

The following features of the .NET Micro Framework allow you to reduce development costs and bring your products to market faster in comparison to other traditional embedded platforms:

- Managed code
- Modern language with Visual C#
- Rich base class library (a subset of common .NET and extensions for embedded development)
- Ability to touch the hardware with managed drivers
- Integration with Visual Studio, a first-class and widespread development tool
- Many .NET developers available
- Sharing code with existing solutions
- Rapid prototyping and debugging using the extensible emulator
- Live debugging on devices

## The .NET Micro Framework and Windows SideShow

You often hear about Microsoft Windows SideShow together with the .NET Micro Framework. SideShow is an environment based on the .NET Micro Framework that runs on devices containing the framework; it can get content from a host computer and display that content on small devices (see Figure 1-7). SideShow-enabled devices can be permanently connected to a computer, or you can use them independently of the host computer that provides the content. For example, a SideShow-enabled auxiliary display on a notebook cover enables you to browse your appointments or read e-mail while your notebook is turned off (see Figure 1-5). The next time the notebook is turned on, the two devices synchronize.


**Figure 1-7.** *SideShow screen*

Devices enabled to use SideShow include the following:

- An auxiliary display for a PC or notebook (see Figure 1-5)
- A remote control with a display (see Figure 1-8)
- A designer bag with a built-in display (see Figure 1-9)
- A cordless phone
- A picture frame
- A portable media player

You can see more SideShow-enabled devices at `www.SideShowDevices.com`.


**Figure 1-8.** *A SideShow-capable remote control*

The content displayed by SideShow-enabled devices is described in the Simple Content Format (SCF), which is based on XML. You can create a custom gadget application for SideShow in native or managed code that runs on your Windows host computer and provides SCF content to a connected SideShow-enabled device. The device's SideShow-enabled environment, which is written with the .NET Micro Framework, shows the SCF content as well as supporting interaction, like menu navigation and responding to other user input.

The SideShow-enabled environment on the devices could theoretically be ported to platforms other than the .NET Micro Framework. However, no other port is available now, and the .NET Micro Framework does the job well.

Microsoft provides a Windows SideShow Device SDK for the .NET Micro Framework that allows original equipment manufacturers (OEMs) to write built-in applications with the .NET Micro Framework and install them on their SideShow-enabled devices (see Figure 1-10).

**Figure 1-9.** *Designer bag with a built-in display using SideShow*

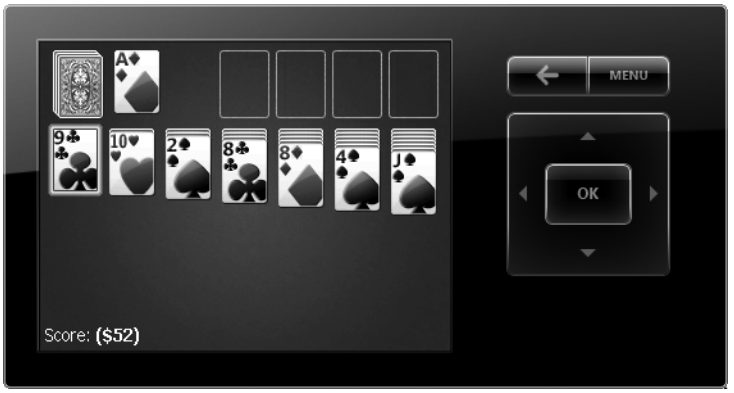


**Figure 1-10.** *A built-in application using SideShow written with the .NET Micro Framework*

# Technical Overview

Now that you know why Microsoft developed the .NET Micro Framework, let's take a look at how it works.

## Introducing the Bootable Runtime Environment

The .NET Micro Framework does not require an underlying operating system and can run directly on the hardware. It provides the following services that are usually provided by an operating system:

- Boot code
- Code execution

- Thread management
- Memory management
- Hardware I/O

However, the .NET Micro Framework is not a full-featured operating system; it's a bootable runtime environment tailored for embedded development. That's how the .NET Micro Framework can run directly on the hardware without an underlying operating system. However, the .NET Micro Framework can still use an underlying operating system and its services; the extensible emulator running on Windows is an example of that.

## Architecture

As I said, the .NET Micro Framework is not a full-featured operating system but a bootable runtime environment tailored for embedded development; it consists of the four layers shown in Figure 1-11.

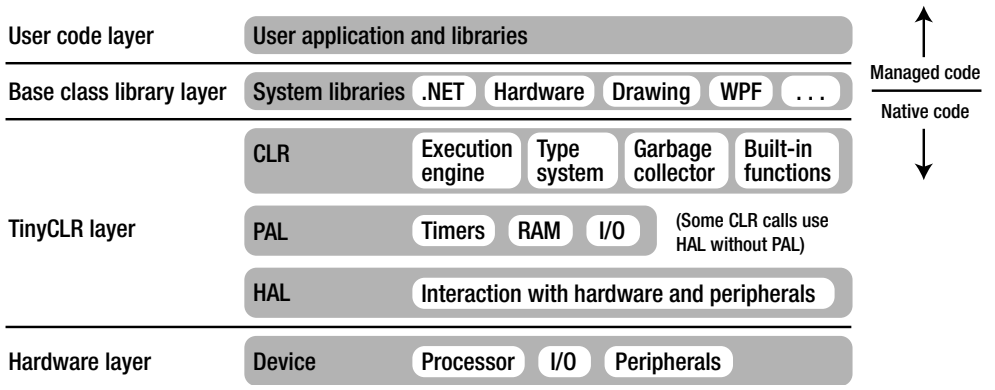


**Figure 1-11.** *Layered architecture of the .NET Micro Framework*

### User Code Layer

The topmost layer is the user code layer. This layer consists of your managed application written in C#. Your application may use your reusable class libraries shared with other projects.

### Base Class Library Layer

The next layer is the .NET Micro base class library layer, which provides a subset of the common .NET libraries and domain-specific extensions, such as libraries for touching the hardware, simple drawing, complex graphical user interfaces, and networking.

The .NET Micro Framework provides a subset of the full .NET Framework base class library (BCL). Microsoft tried to stay in line with the full .NET Framework class library wherever possible. If the functionality or interface of a class deviates from the full .NET Framework, that class or interface was moved to the new namespace `Microsoft.SPOT`.

The .NET Micro Framework base class libraries use a mix of managed and native code. The class libraries do not contain native code. The native methods are built into the runtime environment and are used to perform time-critical tasks or access hardware.

### TinyCLR

The TinyCLR layer consists of the following three components:

- *Hardware abstraction layer (HAL)*: The HAL is tightly coupled to the hardware and provides functions to access the hardware and peripherals. When running on an operating system, the HAL provides functionality by using the services of the underlying OS.
- *Platform abstraction layer (PAL)*: The PAL provides additional abstractions of the HAL, such as timers and memory blocks.
- *.NET Micro Framework CLR*: The CLR consists of the execution engine that manages code at execution time. The managed type system ensures strict type safety. The CLR also contains the garbage collector, which is responsible for automatically freeing unused memory blocks, and the internal native methods called by the base class library. In addition, there is a boot loader that loads and starts the TinyCLR.

### Hardware Layer

The hardware layer consists of the actual hardware platform. This is either the target device (microprocessor and peripheral) or the hardware emulator running on a windows PC.

## Compiling for the .NET Micro Framework

In .NET, compilers, such as the C# compiler, transform the source code to assemblies—executables (`.exe` files) and libraries (`.dll` files)—containing the common intermediate language and metadata for a self-describing code. The CLR then executes the managed code.

The .NET Micro Framework uses a special intermediate language representation optimized for size. A global, shared string table for text and metadata such as type, method, and field names is used to reduce ROM and RAM usage on the devices.

Therefore, the .NET Micro Framework metadata processor tool generates the optimized pe-files to be deployed out of the managed .NET assemblies. Visual Studio and the .NET Micro Framework plug-in hide all these steps from you (see Figure 1-12). You will not see that the metadata processor is used, but you should know that Visual Studio actually deploys special optimized assemblies that the .NET Micro Framework CLR will interpret.

Currently, the .NET Micro Framework supports only Visual C#. Theoretically, every .NET language compiler could be supported, since the metadata processor parses common .NET assemblies that each .NET compiler should be able to generate. In practice, Visual Basic uses a special Visual Basic runtime library that has not yet been ported to the .NET Micro Framework.

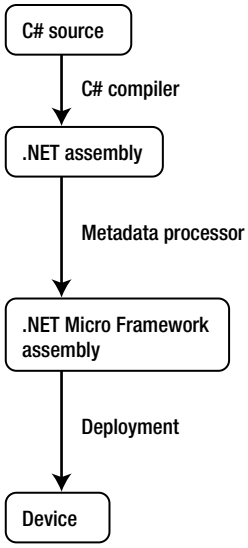


**Figure 1-12.** *Compiling for the .NET Micro Framework*

## Target platforms

Currently, the .NET Micro Framework runs on ARM7 and ARM9 chipsets. Support has been announced for the Blackfin processor, from Analog Devices. The extensible emulator is another port of the CLR on X86 processors; it uses an underlying operating system, such as Windows XP or Vista.

## Platform Porting

The .NET Micro Framework separates all code that interfaces with the hardware in the hardware and platform abstraction layers (HAL and PAL). Theoretically, porting to a new platform seems like it ought to be an easy task, but it is complex and requires a complete understanding of the hardware being used. In practice, a full porting effort will be done by hardware platform vendors rather than by individual embedded developers.

Microsoft provides a porting kit and porting workshops. The porting kit includes the source code for a reference HAL and PAL implementation. The porting kit requires a porting agreement along with payment of an associated fee.

The chipset itself is not so important when programming with the .NET Micro Framework, since the code is processor independent. There are several platforms available that provide all the peripheral hardware supported by the .NET Micro Framework. The devices just vary in display and networking support. This is an active area of development, and we will certainly see interesting new platforms in the future. The next chapter presents available modules and development boards supporting the .NET Micro Framework.

# Links for Further Information

For further general information, visit the following web sites:

- *Official .NET Micro Framework site*: The official .NET Micro Framework site at the Microsoft Developer Network (MSDN) contains news such as porting announcements, the download link for the latest SDK, and a link to the .NET Micro Framework newsgroups at MSDN: `www.microsoft.com/netmf`.
- *.NET Micro Framework team blog*: The blog of the .NET Micro Framework team can be found here: `http://blogs.msdn.com/netmfteam`.
- Here are some .NET Micro Framework–related pages and blogs:
  - *Jan Kuera's web site*: `www.microframework.eu`
  - *Pavel Bánský's blog*: `http://bansky.net/blog`
  - *Sean Liming's web page*: `www.seanliming.com/NETMicroFramework.html`
  - *My blog*: `http://bloggingabout.net/blogs/jens`
- *Windows SideShow–related pages*: For more information on Widows SideShow, visit the following web pages:
  - *Windows SideShow team blog*: `http://blogs.msdn.com/sideshow`
  - *Windows SideShow development forum*: `http://forums.microsoft.com/MSDN/ShowForum.aspx?ForumID=1296&SiteID=1`
  - *Windows SideShow devices examples*: `www.SideShowDevices.com`

# Summary

Now you know what the .NET Micro Framework is. This chapter discussed the history and background of the framework, including the motivation that led to its creation.

This chapter also illustrated how the .NET Micro Framework fits into the story of Microsoft's offerings for embedded development. We discussed the benefits of the .NET Micro Framework compared to other managed embedded options and to traditional embedded development. You learned about the benefits of managed code in general and the limitations of managed code on devices. This chapter also provided a technical overview of the .NET Micro Framework, as well as a description of its architecture.

Regardless of whether you are a .NET developer or a traditional embedded developer, after reading this chapter, you should understand the benefits that the .NET Micro Framework provides. Now that your interest in learning more about this different embedded development approach is piqued, let's continue the tour in the next chapter, which presents all the currently available .NET Micro Framework devices.