

Chapter 1

Compact Framework Architecture

The .NET Compact Framework is a programming interface and runtime library created as a combination of two Microsoft technologies: Windows CE, an operating system for mobile and embedded 'smart devices', and .NET, Microsoft's reinvention of its programming interfaces and its developer tools. This chapter covers key elements of Windows CE and key elements of .NET to show how they come together in the .NET Compact Framework [\[Comment 1.1\]](#).

- Windows CE Overview
- What is .NET?
- The .NET Compact Framework

Windows CE Overview	3
Design Goals	4
Small	4
Modular	5
Portable	6
Compatible	7
Connected	8
Real-time	9
Platforms and Platform Builder.....	10
Embedded Visual C++.....	10
Remote Tools	12
Documentation	13
Obtaining Embedded Visual C++	13
What is .NET?.....	13
Available .NET Implementations	14
The Scale and Scope of .NET	14
The Scale of .NET	14
The Scope of .NET	16
Three Application Classes.....	16
Windows Forms and Web Forms.....	17
XML Web services.....	18
Common Programming Elements	20
Well Designed Programming Interface	20
Common Intermediate Language (CIL).....	21
Common Language Runtime (CLR).....	23
Common Language Specification (CLS).....	23
Common Type System (CTS)	24
Common Language Infrastructure (CLI)	24
The .NET Compact Framework.....	24
Design Goals	25
Build on the Benefits of the .NET Framework	Error! Bookmark not defined.
Maintain Consistency with the desktop	26
Run well on mobile and embedded devices.....	28
Expose the richness of target platforms.....	28
Preserve the look & feel of platforms	28

Compact Framework Files	29
Compact Framework Capabilities	33
Supported Runtime Elements	33
Supported .NET Application Classes	34
Graphical Output	35
Supported ADO.NET Elements	36
XML Support	37
Compact Framework Limitations	39
Conclusion	40

In the early 1990's – Microsoft declared that two technologies were core to its long-term strategic plans: Win32 and COM¹. Microsoft created Win32, the 32-bit Windows Application Programming Interface (API), as a successor to its then-dominant Win16 API. COM, Microsoft's component software architecture, was introduced as a way to snap together software components in a fashion analogous to the way that electronic integrated circuits are snapped together.

[\[Comment 1.2\]](#)

The first version of Windows CE shipped in 1996, with the Win32 API at its core. Windows CE is Microsoft's only operating system with Win32 as the primary API for both application development and device driver development². In some respects, Windows CE was developed as a smaller, lightweight version of Microsoft's 32-bit operating systems – a kind of Windows XP-Lite. But Windows CE shares little if any code with the heavier desktop operating systems, having been written from scratch to address the needs of hardware and software engineers working to develop mobile and embedded smart devices. On Windows CE today, Win32 is still a valid option for application development. In some cases, Win32 provides the only way to accomplish some things. [\[Comment 1.3\]](#)

COM provides the building blocks for a wide range of components. COM first supported compound documents, the *Object Linking and Embedding* (OLE) specification. OLE provides the ability to embed an object such as a spreadsheet or a bitmap into a container document, such as a word processing document. COM was renamed to ActiveX when the Internet Explorer, and its support for ActiveX controls, was introduced. COM/ActiveX components allow third-parties to add new components – in the form of controls, plug-ins, and other kinds of extensions. [\[Comment 1.4\]](#)

COM provides the glue for connecting together components. As such, it addresses some critical development issues, including code-reuse and programmer productivity. But COM is complex and quirky, requiring both developers and consumers of components to handle administrative chores which would be better handled if automated. For example, COM components must track connections to external clients with a reference count. Although simple in principle, COM programming in practice requires a serious investment of time and energy to master. [\[Comment 1.5\]](#)

When the .NET initiative was announced in 2000, it heralded a replacement for both Win32 and COM. While Microsoft would continue adding Win32 extensions to its various operating systems, the Win32 would no longer be the core programming interface. That role would fall to the .NET Framework on desktop versions of Windows, and the .NET Compact Framework on Windows CE platforms such as the Pocket PC. And while support for COM would continue³, the

¹ COM = 'Component Object Model', an architecture for building distributed components.

² By contrast, other members of the Microsoft Windows operating system family use either VxDs (for Windows 95, 98, Me) or a kernel-mode driver interface (for Windows NT, 2000, XP, and 2003 Server) that is decidedly not the Win32 API.

³ The .NET Framework supports COM with a feature referred to as *COM Interop*. Version 1.0 of the Compact Framework has no COM support; a future version will add COM interop support.

role of new software components would be played by .NET assemblies and not by COM components. [\[Comment 1.6\]](#)

The .NET Compact Framework brings the technologies of the .NET Framework to the mobile and embedded world of Windows CE. To better understand the dynamic of this mix, this chapter starts with a discussion of Windows CE. We next look at the broader scope of .NET as implemented in the .NET Framework with its support for Windows Forms, Web Forms, and XML Web services. We conclude the chapter with a discussion of the Compact Framework – what it is, how it works, and where it is going. [\[Comment 1.7\]](#)

Windows CE Overview

As of this writing, the Pocket PC is the most successful Windows CE-powered device. As a result of that success, Compact Framework supports all versions of the Pocket PC, including the first two generations which are built on Windows CE 3.0 (Pocket PC 2000 and the Pocket PC 2002). This is noteworthy because, except for the Pocket PC, all Windows CE-powered platforms must run Windows CE version 4.2 or later to support the Compact Framework. (The third generation of Pocket PC, the Pocket PC 2003, runs this later version of the operating system.) [\[Comment 1.8\]](#)

To many people, Pocket PC and Windows CE are one and the same. But that view misses the point about Windows CE's wider impact and how it addresses the needs of the embedded world. Windows CE is the foundation for a wide range of other smart devices, including the SmartPhone. As we were finishing this book in the summer of 2003, Microsoft launched a marketing campaign with a new brand – Windows Mobile Devices – to tie the Pocket PC and the SmartPhone together as mobile extensions to the Windows brand name. [\[Comment 1.9\]](#)

Beyond the Pocket PC and the SmartPhone, many other smart devices are powered by Windows CE, including barcode scanners from companies like Intermec, Psion Teklogix, and Symbol Technologies, smart displays from companies like View Sonic and Philips, and automobile-based navigation and entertainment systems – running CE for Automotive – in cars from BMW (developed by Siemens VDO Automotive AG), Citroen, Fiat, Mitsubishi, Subaru, Toyota and Volvo. [\[Comment 1.10\]](#)

Because the Pocket PC and Windows CE are connected in the minds of so many, developers new to Windows CE often assume that a required Pocket PC feature is also required for all Windows CE devices. But Windows CE is a highly configurable operating system, with hundreds of system components and device drivers to choose from. While many Windows CE-powered devices have display screens, headless configurations – such as network routers and set-top boxes – are also possible. Many CE-powered smart devices run as mobile, battery-powered devices; but wall-powered, stationary smart devices are also possible, and Windows CE has been incorporated into Automatic Teller Machines (ATMs) and computer printers. And while some Windows CE-powered devices rely solely on RAM with no rotating storage media, Windows CE supports ATA drives and other *installable file systems* to extend available storage beyond what is available in the *object store*⁴. [\[Comment 1.11\]](#)

We wrote our sample code for the Pocket PC

Because of the success of the Pocket PC, we wrote this book expecting that Pocket PC programmers would make up the majority of our readers. Most of our examples, in fact, are built for quarter-VGA portrait-mode

⁴ The object store is a RAM-based storage area with three elements: a file system, registry, and the CEDB property databases. These are discussed in detail in chapter 11, *Storage*.

display screens – the size and orientation which, up until the present, has been standard on a Pocket PC. [\[Comment 1.12\]](#)

For developers writing programs for devices other than Pocket PC, all of our samples should run with few if any changes. The only thing needed is that a platform be compatible with the Compact Framework, either installed in ROM or at runtime into the object store. [\[Comment 1.13\]](#)

Design Goals

When starting to look at any technology, it helps to know what the creators of the technology had in mind during their development process. We start, then, with a discussion of the design goals for Windows CE. We have found that each design goal has a real impact on what an application programmer can do, and that understanding these design goals help to understand a wide range of choices made by the Windows CE development teams. [\[Comment 1.14\]](#)

We learned about these design goals through discussions with the original Windows CE development team. These design goals still play an important role, because they describe the requirements which subsequent development teams have followed as they enhanced and fine-tuned this operating system: [\[Comment 1.15\]](#)

- Small
- Modular
- Portable
- Compatible
- Connected
- Real-time

Small

The first design goal is also the most important: Windows CE was built to be small. The smallest Windows CE image is less than 500K – the tiny kernel that has no display screen and no device drivers. By itself, the tiny kernel image supports a file system, can run processes, start threads, load dynamic link libraries, and access memory. The tiny kernel does not support enough of the Win32 API to support the Compact Framework, but is certainly enough for simple devices like a dumb printer, or a portable music player. [\[Comment 1.16\]](#)

A more typical device image might occupy 5 or 10 megabytes of RAM – enough to support a display screen and enough network protocols to run a web browser. Devices such as the Pocket PC might have a 32 megabyte (or larger) ROM image, consisting largely of application programs and optional device drivers. [\[Comment 1.17\]](#)

This is not small by the standards of other embedded operating systems, which might have a minimum footprint even smaller than the tiny kernel. What Windows CE does provide, however is a feature-rich, configurable operating system. By way of comparison, consider the various desktop versions of Windows. Windows Me, for example, requires 100 megabytes of disk space; Windows XP requires 500 megabytes of disk space. In the Microsoft Windows family of operating systems, Windows CE is definitely the very small – and very flexible – sibling. [\[Comment 1.18\]](#)

The emphasis on making Windows CE small was to reduce the required hardware, thereby making Windows CE a good fit for high-volume, low-cost consumer electronic devices. In that highly competitive market, development teams work hard to reduce costs by scaling back on the required hardware – the RAM, the ROM, the CPU, and the myriad components – because lower costs create a competitive advantage in the price conscious world of consumer electronics. [\[Comment 1.19\]](#)

The 'small-is-good' mindset for Windows CE mostly affects developers who come to Windows CE with Win32 or .NET Framework experience from desktop versions of Windows. Such programmers often have an experience that we call 'stubbing your toe,' which occurs when a programmer new to Windows CE starts to get comfortable with whatever API they are using. Just when they think they have device-side programming figured out, they find they want to use an old familiar friend from the desktop that is not implemented under Windows CE. It might be a Win32 function call. It could be a .NET Framework namespace, class, property, method, or event. At first, these can be frustrating experiences – making one wonder what global conspiracy has deprived you of your favorite way to do some common task. In most cases, there is another way to accomplish what you need done. To make sense of the omission, it helps to remember that small is good, and that Windows CE was built to be a small operating system. [\[Comment 1.20\]](#)

Microsoft's Compact Framework team followed this design goal. The first version of the Compact Framework occupies less than 2 megabytes⁵. This compares to the 30+ megabytes which are occupied by the .NET Framework on desktop versions of Windows. [\[Comment 1.21\]](#)

Modular

Windows CE is modular, a quality that is necessary because Windows CE is configurable. Unlike desktop versions of Windows, which are developed and which ship as a monolithic set of files, Windows CE itself is made up of *modules* (programs as .exe files, and libraries as .dll files), and some modules are made up of two or more *components* (each of which consists of Win32 functions or operating system features). [\[Comment 1.22\]](#)

When a development team designs a new device, they use a tool from Microsoft called *Platform Builder*. Platform Builder allows a third-party to customize the operating system image by adding or removing various modules as needed for their custom smart device. Building a platform which needs a file-open and file-save dialog box? There is a module for that. Need this display driver or that network driver? Platform Builder lets you fine-tune the elements of an operating system image so that you have just the set that you need. Among available Platform Builder components is the Compact Framework itself, which means that custom devices can include the Compact Framework in its ROM⁶ image. The Pocket PC 2003 platform, for example, includes the Compact Framework as a built-in component. [\[Comment 1.23\]](#)

Pocket PC and the Platform Builder

The subject of Pocket PC and the Platform Builder often raises questions for smart device developers. Can the Platform Builder be used to create a custom image for a Pocket PC? Generally speaking, the answer is no. Only developers with a special Pocket PC-specific version of the Platform Builder can create a custom Pocket PC operating system image. The security of Pocket PC devices is further insured because the ability to install a new operating system image on a given platform normally requires special knowledge of that platform. [\[Comment 1.24\]](#)

In our training classes, we often get asked about the difference between a Pocket PC and a custom embedded device built with the Platform Builder. The short answer is that the core operating system is the same, and so porting software between a Pocket PC and other Windows CE devices is very doable. For Win32 programs, the key issue is the available set of functions that are supported on each device. Assuming that two Windows CE-powered devices support the

⁵ The actual size depends on the CPU and whether files are compressed or not. ROM-based files have the option of being compressed, and files in the RAM-based file system are always compressed with a fast-but-light compression algorithm that yields a 2:1 compression ratio.

⁶ ROM = Read-Only-Memory. A device manufacturer might use ROM, although or flash memory is often used. The benefit of flash memory is that field upgrades can often be accomplished without opening a device's physical case.

same set of Win32 functions, porting of Win32 (or MFC) programs should require a minimal of effort. Compatibility problems are often the result of platform-specific assumptions which found their way into the source code: assuming a specific screen size, a specific CPU, directory name for installable file systems, or other system state which should be queried at run-time, and not hard-coded. [\[Comment 1.25\]](#)

For developers targeting a specific smart device, the configurability of Windows CE means that the operating system image can be fine-tuned to include an exact set of desired features and functions. This quality is especially useful for developers of closed platforms, meaning platforms that can only be extended by the original development team. [\[Comment 1.26\]](#)

For other software developers, the configurability of Windows CE introduces some potential problems and risks. In particular, developers creating software for a wider range of Windows CE platforms need to think about what specific Win32 functions might be available on various platforms. This is important, for example, when developing a Win32 program, library, or development tool. Will the file-open dialog be present? What fonts will be available? What display screen will various devices have? How to handle headless platforms (meaning platforms with no display screen)? [\[Comment 1.27\]](#)

To help address some of these issues, Microsoft defines a 'Standard SDK' in the Windows CE .NET 4.2 Platform Builder. This SDK provides a reference point as an aid for multi-platform development efforts. This standard should help promote the creation of tools that can run on a wide range of platforms. The configurability of Windows CE is something of a mixed blessing for Win32 programmers. With configurability comes the possibility that some platform might not support a feature or function that a given program needs. [\[Comment 1.28\]](#)

For Compact Framework programs, the potential problems caused by operating system configurability are much less of an issue. The problem does not entirely disappear, because there is a minimum set of Win32 features that must be present to support the Compact Framework libraries. Once that threshold has been crossed, however, this becomes a non-issue for Compact Framework programmers. It is an all-or-nothing proposition. [\[Comment 1.29\]](#)

And while Windows CE itself is highly configurable, the current version of Compact Framework is not. This means that any platform which supports the Compact Framework should be able to run any Compact Framework program. For the present, then, programmers targeting the Compact Framework can design and develop their software with a good idea of what the Compact Framework itself can provide. [\[Comment 1.30\]](#)

Portable

The major challenges to the success of any operating system include availability of two critical categories of software: application programs and device drivers. Without a wide variety of software of each type, an operating system has limited appeal. To help address this, Microsoft has worked hard to enhance the portability of existing software to Windows CE. The basic idea is to leverage off the success of existing software – and software developers – in creating a successful Windows CE ecosystem. [\[Comment 1.31\]](#)

The first step was to establish the Win32 API as the primary programming interface for both applications and device drivers. As we discuss earlier in this chapter, Win32 was positioned as the strategic API for all Windows operating systems since the API first shipped in 1992. That focus continues with Windows CE. [\[Comment 1.32\]](#)

Other desktop programming interfaces have also been made available in Windows CE. This includes the Microsoft Foundation Class (MFC) library, which provides to C++ programmers an object-oriented layer on top of the Win32 API. It also includes the ActiveX Template Library (ATL), first created in support of lightweight ActiveX controls for the Internet Explorer. It also included Embedded Visual Basic, a lightweight version of the desktop Visual Basic environment. (Support for Embedded Visual Basic ends with the Pocket PC 2003; for the future, Embedded

Visual Basic programmers will need to switch to the Compact Framework, or adopt a third-party tool such as that provided for NS-BASIC⁷. [\[Comment 1.33\]](#)

To support the creation of new device drivers, Windows CE device drivers follow the desktop Windows NT/2K/XP driver model wherever possible. This is true for network (NDIS) drivers, as well as for display, printer, keyboard, and USB device drivers. The underlying implementation for all of these drivers is a simplified, lighter weight version of what is found on the desktop. The inspiration and architecture are in desktop drivers, the better to encourage desktop driver writers to support a Windows CE-based device driver. [\[Comment 1.34\]](#)

The Win32 API was designed as a portable programming interface, intended to provide *source-level-portability* between different CPU platforms⁸. The goal, which has largely been achieved in Windows CE, is that a single body of source code can be rebuilt to run on different CPUs. Just about every Windows CE developer takes advantage of this design, sometimes without thinking about it. Consider a developer who builds and tests software on the desktop-based emulator and a StrongARM-based Pocket PC; such a developer builds x86 executables to run in the emulator, and StrongARM executables to run on the Pocket PC. [\[Comment 1.35\]](#)

The Compact Framework takes the notion of portability one step further by supporting *binary-portability* between different processor platforms. Once built, a single Compact Framework (.exe) program runs unchanged under the Compact Framework on any supported processor, including StrongARM, MIPS, SH3, SH4, and on x86 CPUs. In addition, that same executable also runs on desktop versions of Windows with version 1.1 of the .NET Framework⁹. [\[Comment 1.36\]](#)

Several factors contribute to allowing this to happen. A .NET program gets compiled just like a C or C++ program, except that the resulting executable file is not the machine code of any specific CPU. Instead, the executable files uses a CPU-neutral machine-level language, the *Microsoft Intermediate Language* (MSIL) – sometimes known as the *Common Intermediate Language*, or CIL. We discuss this language later in this chapter, in our discussion of .NET. [\[Comment 1.37\]](#)

A second factor allows a Compact Framework program to run under the desktop .NET Framework. That factor, which we discuss next, is the compatibility between Windows CE programming interfaces and desktop programming interfaces. [\[Comment 1.38\]](#)

Compatible

Portability makes it possible to move code from the desktop to Windows CE, and between Windows CE devices built with different CPUs. Making the programming interfaces compatible means keeping the device-side interfaces as consistent as possible with the desktop interfaces. In doing so, the Windows CE team take the concept of portability one step further, making it as easy as possible to share code between desktop and smart devices. [\[Comment 1.39\]](#)

In some cases, this involved matching features on the desktop with a comparable feature in Windows CE. Consider the file system: both Windows CE and the desktop support a hierarchical file system with long names; the maximum file path in both environments is 260 characters. And both desktop and Windows CE use a hierarchical registry for system and application settings. [\[Comment 1.40\]](#)

Compatibility is further enhanced by choices made for each of the various programming interfaces. While Windows CE supports fewer Win32 functions than the desktop, those functions which are supported match the desktop equivalent as closely as possible. For example, the

⁷ For details, see www.nsbasic.com.

⁸ The first operating system to host the Win32 API was Windows NT version 3.1, which was simultaneously developed for the MIPS R4000 processor and the Intel-x86 family of processors. Subsequent versions ran on the DEC Alpha architecture and the PowerPC processor families.

⁹ Compact Framework compatibility is not supported for version 1.0 of the .NET Framework.

CreateWindow function on the desktop has the same number and type of parameters as that of the same function in Windows CE. [\[Comment 1.41\]](#)

It seems obvious that new operating systems maintain compatibility with the predecessors, but it has not always happened that way. By way of example, consider the OS/2 operating system – developed jointly by Microsoft and IBM as a successor to Windows. Presentation Manager, the GUI programming API, was 'fixed' and 'improved' compared to the Win16 API on which it had been based. But the resulting API was so different that the new API had lost all ties to the Win16 API which it was meant to replace. [\[Comment 1.42\]](#)

This was a problem because there was a large base of existing Win16 code, and relatively little code for the Presentation Manager. Many developers found that porting Win16 code to the Presentation Manager required the same level of effort as porting the same program to other GUI systems such as the Macintosh. To address this issue, Microsoft created a new programming interface, Win32, which had as its primary goal consistency and compatibility with the Win16 API. From this hard lesson came a greater understanding of the effort required to create an API upgrade path. Windows CE developers are a beneficiary of that experience, and gain the advantage of the ease-of-porting Win32 software from the desktop. [\[Comment 1.43\]](#)

A similar type of compatibility is found with the Compact Framework. The Compact Framework team worked hard to maintain consistency between the Compact Framework and the .NET Framework. As we discuss later in this chapter, the Compact Framework has many elements in common with the desktop – a common set of namespaces, classes, properties, methods, and events. [\[Comment 1.44\]](#)

Connected

Windows CE enables smart devices to be well-connected – to other Windows CE devices, to local area networks (both wired and wireless), and to the Internet. The Windows CE team has consistently added support for new connectivity options with each new release of the operating system. Windows CE-powered devices can connect to a personal area network (PAN), a local area network (LAN), and to wide-area networks (WAN). [\[Comment 1.45\]](#)

A personal area network describes the point-to-point connections which a Windows CE-powered smart device establishes in its direct proximity. Included as PAN devices are infrared (IrDA) ports, and blue-tooth controllers. Using these technologies, two Windows CE devices can share data: from tiny bits of data – such as calendar appointments and contact information – on up to include entire files. [\[Comment 1.46\]](#)

Most readers are familiar with local area networks, which typically connect a group of client systems and server systems within the same building or the same floor of a building. Windows CE supports several types of LAN adapters, including Ethernet (802.3), Token Ring (802.5), and wireless Ethernet (802.11). [\[Comment 1.47\]](#)

Windows CE also supports connections to broader wide-area networks, through a variety of devices and using a wide range of protocols. Windows CE supports the telephony API, which broadly speaking provides management of both incoming and outgoing telephone calls. Using the telephony API and a supported serial and/or modem driver, a Windows CE device can connect to the Internet using a dial-up connection through the plain-old telephone system (POTS). Dial-up networking supports both the Serial Line Internet Protocol (SLIP), as well as the Point-to-Point protocol (PPP). [\[Comment 1.48\]](#)

Security considerations are always paramount when communicating over the Internet. A Windows CE-powered smart device can establish a secure, private connection over the Internet to a distant corporate LAN using the Point-to-Point Tunneling Protocol (PPTP) to establish a secure Virtual Private Network (VPN). Among the other features that Windows CE provides for secure network communications are the Secure Socket Layer (SSL), support for the Cryptography API, Kerberos and NTLM authentication, and support for an IP firewall. [\[Comment 1.49\]](#)

In general, when there is a client/server relationship, Windows CE and the Compact Framework support the client-side of the connection. For example, the Compact Framework supports the creation of XML Web services clients, but not the creation of Web service servers¹⁰. [\[Comment 1.50\]](#)

The Compact Framework supports many of the high level network-oriented classes, just like the .NET Framework, including both TCP socket clients (the `TcpClient` class¹¹) and UDP socket clients (the `UdpClient` class¹²). An interesting omission – in both the Compact Framework and the .NET Framework – is support for the RS232 serial port. Neither framework supports a direct connection to an RS232-driven device. Programs that need this support must bypass the frameworks, and drill through to the underlying Win32 protocols. Compact Framework programs can access Win32 libraries through P/Invoke (a subject we discuss in chapter 4, *Platform Invoke*). [\[Comment 1.51\]](#)

Real-time

The final design goal for Windows CE is support for the development of real-time systems. In the world of embedded programming, 'real-time' refers to the ability to perform some specific task – whether calculating a value, recording some input, or sending a command to some external device – within a specified time. Many systems require that a task – or set of tasks – be consistently and reliably performed, no matter how heavily loaded a system. [\[Comment 1.52\]](#)

Starting with Windows CE 3.0, the Windows CE team put in place a set of important features to support the development of real-time systems. Included among these features are support for 256 thread priorities (Windows CE has always supported multi-threaded programming), and support for nested interrupt requests. The Windows CE real-time support is defined in the documentation for the Windows CE Platform Builder as follows: [\[Comment 1.53\]](#)

- Guaranteed upper bounds on high-priority thread scheduling — only for the highest-priority thread among all the scheduled threads. [\[Comment 1.54\]](#)
- Guaranteed upper bound on delay in executing high-priority interrupt service routines (ISRs). The kernel has a few places where interrupts are turned off for a short, bounded time. [\[Comment 1.55\]](#)
- Fine control over the scheduler and how it schedules threads. [\[Comment 1.56\]](#)

The world of real-time systems can be divided into two kinds: those with hard real-time requirements, and those with soft real-time requirements. Soft real-time requirements allow for some of the deadlines to be missed some of the time with no serious consequences. Hard real-time requirements are those which cannot tolerate even a single late reply. [\[Comment 1.57\]](#)

Consider a system that controls the movement of a robot arm on a factory floor. Perhaps the arm must be moved out of the way so that the assembly line can move forward, or so that other machinery can access the assembly line. If the failure to move the arm in a timely fashion results in damage to the arm, to the product, or to any part of an assembly line, this would be called a hard real-time requirement. (The assumption is that such a breakdown could stop an assembly line, with the resulting down-time and costs.) [\[Comment 1.58\]](#)

While Windows CE provides good real-time support, the Compact Framework is not the ideal programming interface for building real-time components. The reason is that various delays can occur at irregular intervals because of the requirements of the runtime engine. There is a delay, for example, when code is loaded and converted into native machine code by the execution-time just-in-time (JIT) compiler (which we discuss later in this chapter). Another delay

¹⁰ Windows CE does, however, support the creation of Win32-based XML Web service servers with version 2.0 of the SOAP toolkit.

¹¹ Fully-qualified name: `System.Net.Sockets.TcpClient`.

¹² Fully-qualified name: `System.Net.Sockets.UdpClient`.

occurs when the garbage collector is operating, because all threads running in managed code are frozen. [\[Comment 1.59\]](#)

While managed code threads are frozen by the garbage collector, unmanaged code threads are not. In this way, a Compact Framework managed-code program can support a real-time thread. But the real-time thread will most likely do its work in unmanaged code by calling into a Win32 DLL. These are, of course, only general guidelines. Given a fast CPU and a lightly loaded system, a thread running in managed code could provide sufficient response to provide the real-time support needed in a given system. For issues related to timing and performance, the approach taken by carpenters – 'measure twice and cut once' – applies. The number of times you measure the required performance is likely to rise in direct proportion to the cost of missing a real-time deadline. [\[Comment 1.60\]](#)

Platforms and Platform Builder

The Pocket PC is one of many possible types of Windows CE platforms. Microsoft provides *Platform Builder*, a tool for creating custom platforms. Platform Builder allows the developer of a new platform to select from the available modules and components when building a new operating system image. The Platform Builder provides a wide range of elements needed to build a complete, working version of Windows CE, including application programs, dynamic link libraries, device drivers, fonts, and icons. [\[Comment 1.61\]](#)

Microsoft includes the Compact Framework runtime with the Platform Builder. So a custom Windows CE smart device can include the Compact Framework runtime. It is an optional component, however, that is included at the discretion of the platform developer. [\[Comment 1.62\]](#)

Some developers of Compact Framework applications might need to develop device drivers to support their application. The Platform Builder also provides support for device driver developers, in the form of sample device drivers and documentation. As of this writing, device drivers must be written in C or C++ and use the Win32 API, as there is no support for building device drivers in managed code. [\[Comment 1.63\]](#)

Another reason why the Platform Builder might be interesting to Compact Framework developers is that Microsoft provides a significant portion of the Windows CE source code with the Platform Builder. The source code, known as Shared Source, includes a significant portion of the Windows CE kernel, various server components, and various network and communications protocols. A small investment of time could yield a lot of understanding of various Windows CE components, including the web server, Bluetooth support, display driver support, the scheduler, and the low-level memory manager. (This is only a small sample of the 1400 source files – and millions of lines of source code – which you can use to learn about the internals of Windows CE.) [\[Comment 1.64\]](#)

You can develop application programs using the Platform Builder. If you *only* want to develop applications, then the Platform Builder is overkill and you might consider other, more specialized tools. Where it can make sense to use the Platform Builder for application programs is for programs that are part of a platform. For example, it might make sense to use the Platform Builder to develop a configuration tool that every user of a given platform needs to use. [\[Comment 1.65\]](#)

Generally, however, other tools are used to develop applications. We next discuss those tools, which ship with a product from Microsoft called *Embedded Visual C++*.

Embedded Visual C++

Many Compact Framework applications need both managed and unmanaged components. To this end, Microsoft provides two products with tools for developing Windows CE applications:

- Embedded Visual C++ (eVC++) – for native Win32 programs and DLLs
- Visual Studio .NET 2003 (VS.NET) – for managed managed-code Compact Framework programs and shared libraries

The focus of the rest of this book is on the second product, *Visual Studio .NET 2003* and the Compact Framework. There are some useful features of the first product, though, so it is worth a brief note. Even if you never build Win32 programs or DLLs, there are several reasons to install *Embedded Visual C++*. [\[Comment 1.67\]](#)

Embedded Visual C++ ships with an Integrated Development Environment (IDE) which was built from the same source code as the Visual C++ 6.0 IDE. So it lacks some of the refinements that you might notice in Visual Studio .NET. What it does provide is the ability to create Win32, MFC, or ATL programs for Windows CE. You can build, download, and debug such programs to a remote Windows CE-powered device, or to an emulator. [\[Comment 1.68\]](#)

Embedded Visual C++ supports a wide range of smart device configurations. To configure the environment for a specific configuration, you install a Software Development Kit, or 'SDK,' for the device in question. For example, there is an SDK for Pocket PC 2002, and another one for the Pocket PC 2003 (the two are similar, though not exactly identical). Microsoft's tool for configuring custom Windows CE configurations, the *Platform Builder*, generates the SDK to match whatever configuration has been selected for the operating system image. An SDK contains, among other things, a set of include files and linker libraries that are custom tailored for a specific platform. Application developers install a device-specific SDK after installing *Embedded Visual C++*, which adds new options to the new project wizard, and to the *WCE Configuration* toolbar in the eVC++ IDE. [\[Comment 1.69\]](#)

Tips on using the eVC++ Emulator

The emulator lets you run Windows CE programs on a development workstation running Windows 2000 (service pack 2 required), or Windows XP. You cannot run the emulator on 16-bit versions of Windows, owing to the lack of Unicode support which the emulator requires (because Windows CE is a Unicode-only system). [\[Comment 1.70\]](#)

When using the emulators with Embedded Visual C++, be careful of the emulator version. Older emulators rely on the underlying operating system to simulate the behavior of Windows CE, and in many cases the behavior is quite different. Avoid using the emulator for the following devices: Pocket PC, the Handheld PC, the HPC Professional, or the Palm-sized PC. These emulators ship with SDKs that are compatible with Embedded Visual C++ 3.0. [\[Comment 1.71\]](#)

Microsoft invested a lot of effort in creating an emulator that has a high degree of fidelity to Windows CE running on a device. Starting with the Pocket PC 2002, emulators are based on the newer technology. The emulators for the SmartPhone and the Pocket PC 2003 use the newer technology, as does any emulator that is compatible with Embedded Visual C++ 4.0. [\[Comment 1.72\]](#)

Sometimes developers encounter problems getting the emulator to work properly. Here are some tips to help get the emulator going:

(1) Make sure to install the latest service pack from Embedded Visual C++ 4.0; (2) If your development system is not connected to a network, install the Microsoft loopback adapter; (3) With Embedded Visual C++, you must login to an account with Administrator privileges. [\[Comment 1.73\]](#)

The problem might be caused by incompatibilities with the emulator which ships with Visual Studio .NET 2003, which has emulators to support Compact Framework development. In some cases, the incompatibility only causes an inconvenience: for Win32 DLLs that are called from a Compact Framework program, add the DLL to the Visual Studio .NET project, and run both on the Visual Studio .NET emulator. [\[Comment 1.74\]](#)

In other cases, incompatibilities can cause one of emulators to stop working altogether. Sometimes an emulator cannot start, or it can start but cannot receive a downloaded program. In other cases, a debug session to an emulator cannot start. To address this issue, we have had success by shutting down one (or both) emulators completely. To do this, shut down an emulator and select *Turn off Emulator* on the *Shut Down* dialog. [\[Comment 1.75\]](#)

Remote Tools

Embedded Visual C++ includes a set of tools that run on a desktop system and which display details about a connected Windows CE-powered device. Many of these tools are device-enabled versions of tools which ship with current or past versions of Visual Studio. *Remote SPY++* and *Remote Zoomin*, for example, are adaptations of desktop tools. [\[Comment 1.76\]](#)

Table 1-1 summarizes the available remote tools that ship with eVC++ 3.0 and eVC++ 4.0. Most tools are straightforward to use, but some require a bit more work to understand how they work. The easiest way to start these programs is on the *Tools* menu in the IDE. The programs themselves can run by themselves, but they are not placed on the *Start* menu by the eVC++ installation program. We include the name of the executable file in the table, to help you create a shortcut for tools you use often. [\[Comment 1.77\]](#)

Table 1-1 Remote Tools in eVC++ 3.0 and eVC++ 4.x

Name in eVC++ Tools Menu	Executable File Name	Description	eVC++ 3	eVC++ 4
Remote SPY++	cespy.exe	Displays available windows, listens to window message traffic	Yes	Yes
Remote Registry Editor	ceredgt.exe	View and edit device and desktop registry	Yes	Yes
Remote Heap Walker	ceheapwk.exe	View available Win32 (HeapAlloc) heaps	Yes	Yes
Remote Process Viewer	cepview.exe	View current system processes, and details of each process including the running threads and the modules (dynamic link libraries) running in each process. You can also terminate a process using this tool.	Yes	Yes
Remote Zoomin	cezoom.exe	Displays a screen image for a target device, useful for writing documentation.	Yes	Yes
Remote File Viewer	cefilevw.exe	Browse a remote device file system. Supports copying files from a desktop system to a device or from a device to a desktop system.	Yes	Yes
Remote Call Profiler	msic.exe	Tool for displaying and analyzing program execution time.		Yes
Remote Kernel Tracker	kerneltracker.exe	Tool for observing the actions of the scheduler in passing control between threads. Useful for analyzing deadlocks and other inter-thread synchronization problems.		Yes
Remote Performance	ceperfmom.exe	Windows CE version of the desktop		Yes

Name in eVC++ Tools Menu	Executable File Name	Description	eVC++ 3	eVC++ 4
Monitor		Performance Monitor, which collects statistics and displays graphical state for a wide range of system-level resources including memory usage and network traffic.		
Remote System Information	cesysinfo.exe	Displays static snapshot of system information for a device, such as model of CPU, available system memory and system power state.		Yes

Documentation

Embedded Visual C++ provides documentation on Win32 functions as implemented in Windows CE. In some cases, this same information is available in the MSDN Library (the documentation set installed with Visual Studio .NET). In other cases, the eVC++ documentation provides details about device-specific functions that are not otherwise available. For example, you can find details about functions specific to the Pocket PC and also functions specific to the SmartPhone which are not otherwise discussed in the general documentation set. [\[Comment 1.78\]](#)

Where this is particularly useful is when you need to access a feature that is supported by the underlying operating system, but which is not supported by the Compact Framework. Making such calls requires your using the Platform Invoke support of the Compact Framework, which we discuss in detail in chapter 4 (*Platform Invoke*). [\[Comment 1.79\]](#)

Obtaining Embedded Visual C++

Embedded Visual C++ is available at no cost. This is a bit unusual, since Microsoft normally does not give away their development tools. To help promote smart device development, Embedded Visual C++ is free. Before you go and download anything, make sure to download the correct version. For Windows CE 3.0 devices, be sure to download Embedded Visual C++ 3.0. And for Windows CE .NET 4.x devices, you need Embedded Visual C++ 4.x. (Also - be sure to download and install any service packs which might be available.) [\[Comment 1.80\]](#)

What is .NET?

From the moment Microsoft made its .NET initiative public, the question was: "What is .NET?" The short answer is that it is a branding that covers all the different parts of a larger strategy. Another answer, from the Microsoft web site, offers the following, pithy explanation: [\[Comment 1.81\]](#)

Microsoft® .NET is a set of software technologies for connecting information, people, systems, and devices. This new generation of technology is based on Web services—small building-block applications that can connect to each other as well as to other, larger applications over the Internet. [\[Comment 1.82\]](#)

This explanation focuses on the new features that .NET introduces: the network focus for .NET. Towards that end, it mentions one element of .NET: Web services. If you interpret this too literally, you risk missing the larger picture that .NET addresses. At this point, this much should be clear: .NET is a large undertaking by a large company. [\[Comment 1.83\]](#)

To put the .NET Compact Framework into perspective of the larger .NET picture, a more detailed description of .NET is required. Whether you spend your whole career development Compact Framework programs, or whether you move on to desktop .NET development, an

understanding of .NET is important because the effects of .NET will last for a decade or more. To get a good understanding of .NET, it helps to look at the following: [\[Comment 1.84\]](#)

- Available .NET Implementations
- The Scale and Scope of .NET
- Three .NET Application Classes
- Common Programming Elements

Available .NET Implementations

At its heart, .NET is about a runtime that supports a certain set of features. If you want to know what .NET is, then, here is a list of implementations that provide something tangible for you to work with. Microsoft has created three public implementations of .NET, and there are at least two other efforts underway to create .NET runtimes. [\[Comment 1.85\]](#)

- .NET Framework – Microsoft's runtime for desktop and server versions of Microsoft Windows. It is included with two shipping Microsoft operating systems: Windows XP Tablet Edition, and Windows 2003 Server; and is offered as a downloadable installation file, dotnetfx.exe for other versions of Windows (Windows 98 and later.) through a downloadable installation file, dotnetfx.exe. [\[Comment 1.86\]](#)
- Shared Source CLI – Microsoft has released the source code to Rotor, an implementation of the Common Language Infrastructure (CLI), in support of its submission to ECMA of this .NET technology as an ECMA approved standard. This can be downloaded from the Microsoft web site, and used as a resource to learn about the architecture and operation of the .NET runtime. [\[Comment 1.87\]](#)
- .NET Compact Framework – Microsoft's runtime for Windows CE-powered platforms, and of course the subject of this book. [\[Comment 1.88\]](#)
- DotGNU – The GNU project for building an open source implementation of the .NET Framework. [\[Comment 1.89\]](#)
- Mono Project – Another open source project to create an implementation of the .NET Framework from a company called Ximian. [\[Comment 1.90\]](#)

The Scale and Scope of .NET

A complete answer to the question "What is .NET?" has proven elusive in part because the scale and scope of the .NET initiative contains so many elements. Like the story about the group of blind men encountering an elephant, each experiences something different from the others: the broad flat surface is an ear; the skinny wavy object is the tail, and so forth. Each can discuss what he finds with the others, but none agree on what exactly they find because each encounters something different. [\[Comment 1.91\]](#)

The Scale of .NET

The .NET initiative is a major technology initiative for Microsoft. Steve Ballmer, Microsoft's CEO, refers to .NET as a "bet-the-company" investment. To put this into the perspective of Microsoft's (almost) thirty year history, .NET is the third major initiative of this magnitude. The two previous efforts that had anywhere near the kind of impact that .NET promises were major direction changes for Microsoft. We refer here to the two operating systems which reshaped Microsoft, and which also had a major impact on the rest of the computer industry: MS-DOS and Microsoft Windows. A brief summary of Microsoft's three major initiatives are as follows: [\[Comment 1.92\]](#)

- 1981 – Beginning of MS-DOS era – character-based computing
- 1990 – Beginning of Windows era – GUI-based computing
- 2000 – Beginning of .NET era – Network-centric computing

Microsoft was founded in 1975 to provide a BASIC interpreter for a new computer system, the MITS Altair. Since the company was founded to build a programmer tool, Microsoft has never lost sight of the importance of the programmer in their long-term, strategic plans.¹³

[\[Comment 1.93\]](#)

In 1981, Microsoft acquired all rights to a character-based operating system from Seattle Computer Products, written by an employee named Tim Patterson. Microsoft licensed the operating system to IBM, and in August of 1981 IBM started shipping the IBM-PC. Microsoft subsequently licensed MS-DOS to the broader market for IBM-compatible computers. In doing this, Microsoft established itself as an operating system company and made its first major direction change: from a developer tool company to an operating system company. [\[Comment 1.94\]](#)

In May 1990, Microsoft released Microsoft Windows 3.0, the version which achieved widespread success and established Microsoft Windows as the standard Graphical User Interface for personal computers. This did not happen overnight; Microsoft started work on the first version of Windows in early 1983. Almost two years later, in November of 1985, Microsoft shipped the first version of Windows. [\[Comment 1.95\]](#)

There was a third operating system. In 1984, Microsoft embarked on a joint development agreement with IBM to develop OS/2. In those days, Microsoft was positioning OS/2 as the successor to Windows¹⁴. With the success of Windows 3.0, Microsoft dropped OS/2 – an operating system in which Microsoft had made a sizeable investment. [\[Comment 1.96\]](#)

With the success of Windows 3.0, Microsoft succeeded in making its second major direction change: from a character-based operating system company to a Graphical User Interface (GUI) operating-system company. This change required many years of dedicated effort, in part because GUI systems are inherently more complex than character based systems. But also the competition was more intense for this second change, with operating system alternatives that include Apple's Lisa and Macintosh systems, GEM from Digital Research, DeskView from Quarterdeck, and TopView¹⁵ from IBM. [\[Comment 1.97\]](#)

In the grand scheme of things, .NET is significant because it represents only the third major direction change in Microsoft's history. Microsoft announced the .NET initiative in July 2000 at the Professional Developers Conference (PDC) held in Orlando, Florida. .NET amounts to Microsoft's complete re-invention of its core software technologies, its software development tools, and ultimately of all of its products. By contrast, the adoption to MS-DOS was simple: a group of engineers supported Microsoft's OEM customers. The total number of OEM customers for MS-DOS was under 100, a population that could be supported by the MS-DOS team itself. Windows required a much larger development team than MS-DOS, and also a larger support organization to assist the tens of thousands of software companies which built software for the operating system. [\[Comment 1.98\]](#)

The .NET initiative involved¹⁶ an even greater effort than what went into creating Windows, in large part because it involved re-creating in three or four years a larger number of older technologies, some of which had been developed over a decade or more. The driving force this time is not just a new user-interface, one of the key features that Windows 3.0 provided. Instead,

¹³ We omit details because this story is reported more completely elsewhere. Our purpose is to position the .NET initiative in the context of Microsoft's other major direction changes.

¹⁴ What eventually became known as Windows NT 3.1 had another, earlier name: OS/2 NT.

¹⁵ TopView added multi-tasking support to MS-DOS, which by itself is a single-tasking operating system. TopView did not, however, support GUI programming.

¹⁶ The effort is continuing as of this writing.

it involved redeploying these older technologies in a unified way that is ready to support the future of high-bandwidth, network-centric computing. You can get a sense for this by studying the three basic classes of .NET applications. [\[Comment 1.99\]](#)

The Scope of .NET

.NET covers a very wide range of software technologies. Just how broad is the scope of .NET? Consider the following scenarios involving programmers working with a different aspect of .NET technology. [\[Comment 1.100\]](#)

Web site developers are interested in ASP.NET, a set of services which support web site creation. Like other web server systems, ASP.NET runs on a web server system and interacts with web browser using HTML and some scripting language (JavaScript or VBScript). Older web server systems (including Microsoft's own *Active Server Pages - ASP*) freely mix HTML and script to create hard-to-read and hard-to-manage source files. ASP.NET separates the HTML from the code, and provides a host of improvements to the web application development process. [\[Comment 1.101\]](#)

Consider another developer, this one building desktop database applications. Such a developer might want to create a GUI front-end that interacts with a user and manages the interactions with a back-end SQL database. .NET offers this developer many things, but each is different from what the web site developer works with. Such a developer uses the Windows Forms portion of the .NET Framework to build the windows, menus, and dialog boxes that make up the user-interface for a traditional Windows GUI application. And for the database work, the same developer would rely on a different portion of .NET, the ADO.NET support which provides access to the SQL database. [\[Comment 1.102\]](#)

A developer creating mobile database programs for a Pocket PC sees some of the same things as a desktop database programmer. But that developer also sees a programming environment that bridges the gap between desktop programming and device-side programming. Code written for the Compact Framework, for one thing, can run largely unchanged on the desktop¹⁷. As a mobile device, a Pocket PC is often disconnected from any desktop system and also from a wired or wireless network. Pocket PC database programming will likely need to address two separate scenarios: (1) managing data when the device is disconnected, and (2) the merging of that data with an external database when the device is connected¹⁸. [\[Comment 1.103\]](#)

Those two perspectives are different from that of an SQL programmer. While .NET has something for each of these developers, the specific features which are interesting to one group hold little interest to another group of developers. [\[Comment 1.104\]](#)

Three Application Classes

When you scratch the surface of .NET, you find three application classes: Windows Forms, Web Forms, and XML Web services. The first two application classes differ from each other in the user-interface and in the delivery mechanism for that user-interface. A Web service, on the other hand, has no user-interface¹⁹. It describes, instead, a headless application which supports the other two types of application classes. [\[Comment 1.105\]](#)

¹⁷ The reverse is not true – that is, a desktop .NET program typically does not run on a device. Also, the device-to-desktop scenario requires version 1.1 of the .NET Framework.

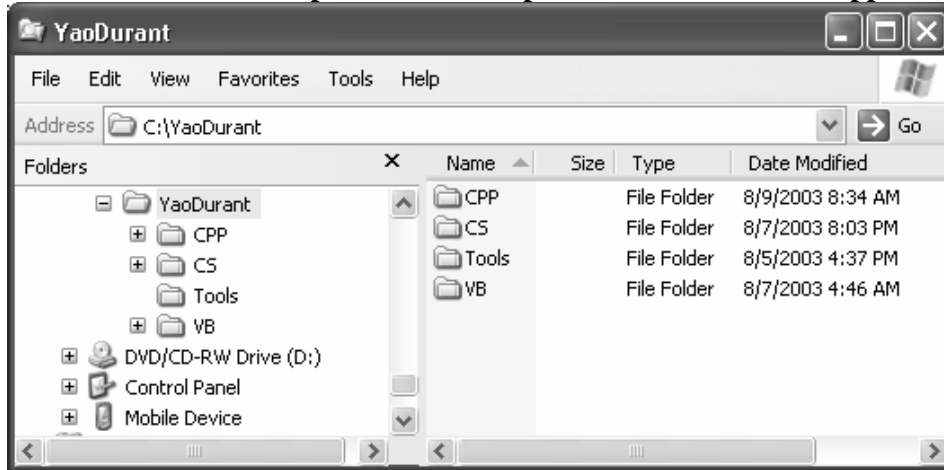
¹⁸ Of course, a Pocket PC database can stand-alone and not require synchronization with a central database, but the interesting real-world scenarios are the ones that push the limit – which in this case means having a distributed database & disconnected devices.

¹⁹ We do not count the WSDL web page as a user-interface; it is at best a development and testing interface, and not a user-interface that interacts with non-technical end-users.

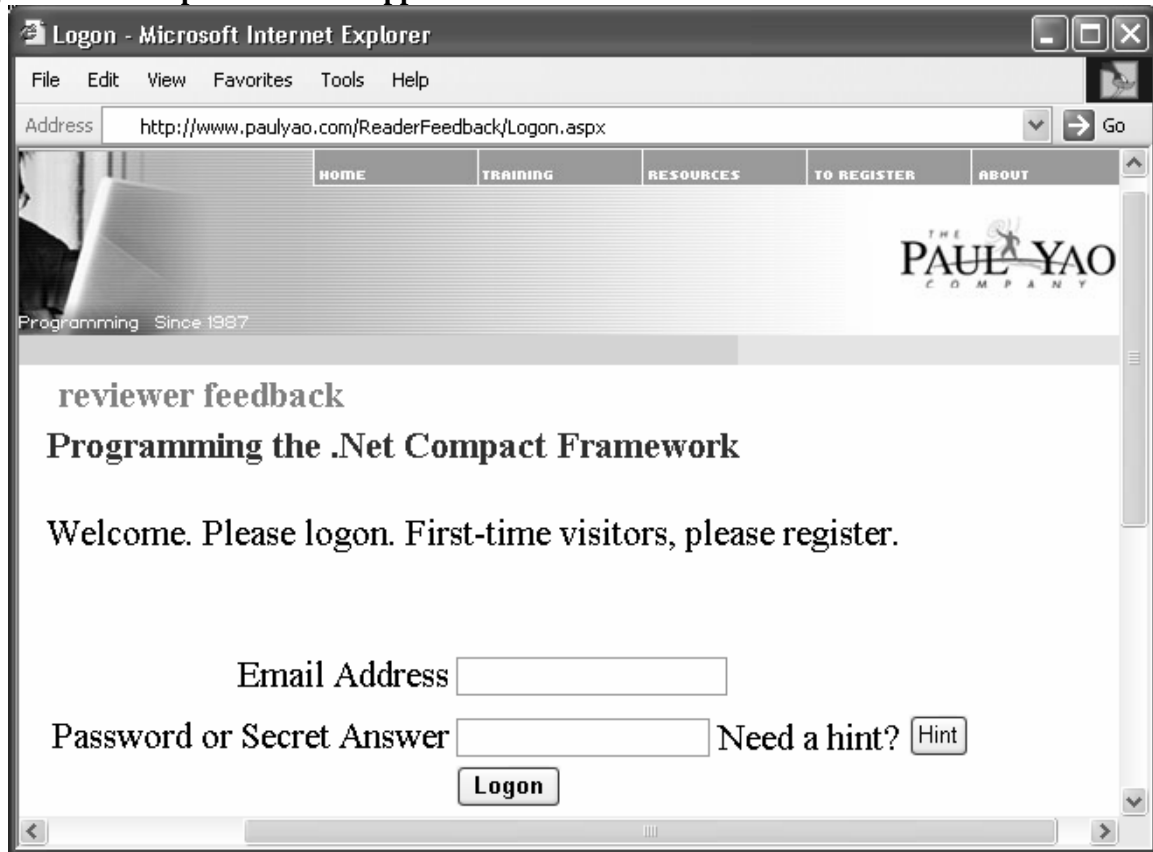
Windows Forms and Web Forms

A Windows Forms application is a traditional GUI application. Such an application might run as a stand-alone application, like a word processor; or it might run as the client portion of a networked client/server application. Windows Forms encompass the rich graphical and interactive elements traditionally associated with GUI systems. Figure 1-1 shows a desktop Windows Forms application which every Windows user is familiar with, the Windows Explorer. [\[Comment 1.106\]](#)

Figure 1-1 The Windows Explorer – an example of a Windows Forms application



A Web Form application is an HTML/script application that appears in a web browser. By definition, these are client/server applications, which require a live connection to a web server to operate. Most of the computing work gets handled on the server side, and the client side involves displaying HTML, navigating between web pages, and handling data entry forms. An example of a Web Form application running in the Internet Explorer appears in figure 1-2, the login page for a web site we created for feedback from book reviewers. [\[Comment 1.107\]](#)

Figure 1-2 A Sample Web Form application

To an end-user, and to a programmer first looking at .NET, there does not seem to be much that is new. Nothing in the Windows Forms application in figure 1-1 has anything that seems to be different from what was available prior to .NET. And, in fact, the Windows Explorer under Windows XP does not use the .NET Framework. All the same, it certainly qualifies as a 'traditional GUI application,' which is another term often used to describe Windows Forms applications. [\[Comment 1.108\]](#)

The same can be said about the Web Forms application in figure 1-2. To an end-user it looks just like any other web page²⁰. While this is, in fact, a .NET application, it presents nothing different or new to the end-user. [\[Comment 1.109\]](#)

Based on what we have discussed so far, it would be easy to dismiss .NET as just a new name for the same old thing. To a software developer, however, there are a huge number of very important differences. One is the third application class, XML Web services, which we discuss next. [\[Comment 1.110\]](#)

XML Web services

The third class of .NET applications is XML Web services. XML Web services have no user-interface, but instead serve to support the other two application classes by providing a standard mechanism for two computers to communicate with each other over a network. [\[Comment 1.111\]](#)

This might not sound like anything new. There have, after all, been network protocols as long as there have been networks. You transfer files with the FTP protocol, log into a command

²⁰ Astute readers may notice that the name of the page ends with '.aspx,' the only outward sign that a web page is hosted by ASP.NET.

prompt with the TELNET protocol, attach to network printers and file servers with the SMB protocol, and browse web pages with the HTTP protocol. [\[Comment 1.112\]](#)

XML Web services are built on a set of standard protocols, notably the SOAP²¹ protocol. The purpose of this protocol is to support remote procedure calls. XML Web services provide the next generation for network-based client/server communications. Previous generation technologies include the Distributed COM (DCOM) from Microsoft, the industry standard DCE RPC protocol, and CORBA. [\[Comment 1.113\]](#)

An important aspect of XML Web services is that they are being adopted as a standard by a wide range of large computer hardware and software companies. So while previous generations of RPC got bogged down in compatibility issues (COM vs. CORBA comes to mind), XML Web services is largely a platform-neutral RPC mechanism. [\[Comment 1.114\]](#)

Why should an application programmer care about web services? It is certainly possible for someone to spend their whole programming career building traditional GUI applications or building web applications. You might find yourself in that situation – in which you do not build a distributed, network-oriented application. In the short term, that may be the case. But looking out more long-term, web services are going to grow in importance. [\[Comment 1.115\]](#)

One reason has to do with the growth in speed, reliability, and security of networks. The incredible advances in computer hardware have made the PC revolution possible. The rise of the Internet was made possible by these advances in computer hardware, but also by advances in computer communications technology. As digital communication continues to evolve, added speed and lower costs will contribute to making XML Web services a key part of many new kinds of application software. [\[Comment 1.116\]](#)

Another reason why XML Web services will become important has to do with the growth and success of the World Wide Web. The World Wide Web is the most successful – and most visible – artifact of the computer industry. If you doubt this, start counting the number of URLs on consumer products; has any computer artifact ever achieved this level of public awareness? [\[Comment 1.117\]](#)

All large companies, government agencies, and non-governmental organizations (NGOs) have made – and will continue to make for the foreseeable future – large investments in their web sites. Web sites aggregate a huge amount of information, which gain value by being made public. All of this data is available in a human-readable form, HTML. [\[Comment 1.118\]](#)

While HTML makes the data easy for people to read, HTML is difficult to access programmatically. Some have built 'HTML scraping' tools to strip the HTML from web-server-based data, but this is a fragile approach to accessing the data. For one thing, changes to a web site break an HTML scraper. There are also legal issues to consider, because often web site owners also own the copyright to the contents of the web site. [\[Comment 1.119\]](#)

One use for XML Web services is to provide programmatic access to the huge collection of web site data. Think back to the last two or three web sites that you browsed. Were you reading the New York Times online? Perhaps looking up a sports score or the weather in a city you are about to visit. If you live near a big city like Seattle, you might go online to check the traffic. In each of these cases, the data you searched for could have been made available programmatically. While finding a piece of information a search engine is sometimes necessary (and, in fact, sometimes fun), anything that you browse for with any regularity is a candidate for accessing from a program through XML Web services. [\[Comment 1.120\]](#)

While all web-browser HTML data is not yet available through an XML Web service, as of this writing web content from quite a few large web sites is available. Here is a partial list of web sites and services which are externally accessible through XML Web services: [\[Comment 1.121\]](#)

- Amazon – Link third-party product inventory database to Amazon product database

²¹ An acronym for *Simple Object Access Protocol*.

- eBay – Access auction information programmatically
- Google – Submit web site search
- MapPoint .NET – Microsoft service for programmatic access to geographic data

We provide more details on XML Web services in chapter 14 (*Building XML Web Service Clients*). [\[Comment 1.122\]](#)

Common Programming Elements

The essence of .NET can be found in the new programming model that .NET introduces for Microsoft Windows. .NET provides a programming model that extends from small-screen mobile and embedded devices, through the personal computer – whether desktop, laptop, or tablet – on up to server-scale systems. .NET provides a unified way to develop many different kinds of software applications, including traditional GUI applications, web-browser accessible applications, or back-end XML Web service applications. The .NET initiative unifies software development across a wide range of device classes, and incorporates a broad set of application types through the following core elements: [\[Comment 1.123\]](#)

- Well designed programming interface [\[Comment 1.124\]](#)
- Common Intermediate Language [\[Comment 1.125\]](#)
- Common Language Runtime [\[Comment 1.126\]](#)
- Common Language Specification [\[Comment 1.127\]](#)
- Common Type System [\[Comment 1.128\]](#)
- Common Language Infrastructure [\[Comment 1.129\]](#)

Well Designed Programming Interface

Back in the early 1980's, Microsoft's goal in creating Windows was simple: to make an easy-to-use graphical user-interface system. The goals never included making it easy to write code for the operating system. Quite to the contrary, it was just and proper that professional programmers bore the burden of taming the complexities of computers. The result was the various APIs behind Windows, first the sixteen-bit Win16 API followed by the thirty-two bit Win32 API. Because Win32 is the API for Windows CE, that is our focus here (although aside from some refinements and extensions, Win32 carries the same flaws as its 16-bit predecessor). [\[Comment 1.130\]](#)

The Win32 API, while serviceable and necessary in many cases, has a number of design flaws. The importance of these flaws is that they serve to highlight the types of problems that Microsoft has addressed with the .NET initiative, which include: [\[Comment 1.131\]](#)

- Quirky API – Win32 has many inconsistencies, many poorly-named functions, quite a few functions with too many parameters, and many large unwieldy data structures. (In many cases, some of the extra parameters are 'reserved' and must be set to zero, and often the large data structures contain unused members.) [\[Comment 1.132\]](#)
- Memory management – Application programs required to cleanup system objects, making the API prone to memory leaks. [\[Comment 1.133\]](#)
- Low-level API – Sometimes called "the assembly-language of Windows," the Win32 API often requires multiple steps to accomplish seemingly simple tasks. [\[Comment 1.134\]](#)
- Procedure oriented API (no benefits from object-oriented languages²²) [\[Comment](#)

²² The first version of Windows pre-dates the first PC-based C++ compiler by at least five years, so there were no object-oriented tools to support object-oriented development.

[1.135\]](#)

In fairness to the development team which created Windows 1, they accomplished a lot in a short time and faced many obstacles. While creating a new GUI, they also were creating a new compiler, a new debugger, a new program loader, and a new memory manager. They worked on very slow computers (80286 and 80386 CPUs), with small storage devices (20 megabytes was large in those days), and limited networking support. Given the obstacles they faced and what they accomplished, one wonders that they were able to deliver at all. [\[Comment 1.136\]](#)

Once Windows started shipping, the Windows API acquired a momentum all its own. At the same time, there was recognition that a better API was needed, and several efforts were put forth. One such effort was OS/2, a project largely staffed by former Windows systems developers. With the success of Windows 3.0, however, Microsoft abandoned OS/2 with its more refined programming interface. Other efforts to improve Windows programming include Visual Basic (which first shipped in 1992), and the Microsoft Foundation Class (MFC) library for C++ programmers. Both provided improvements with fewer quirks than the native Windows API; neither completely replaced the Win32 API. [\[Comment 1.137\]](#)

To programmers building software for Microsoft Windows, .NET represents a new programming interface that is a well-designed, feature-rich candidate for replacing Win32 programming. This interface is called the .NET Framework on the desktop, and the .NET Compact Framework for mobile and embedded devices. These frameworks follow a wonderfully consistent approach to exposing operating system services to programmers. [\[Comment 1.138\]](#)

These frameworks are object-oriented. For example, everything in .NET is an object, derived from the `Object` base class²³. In this object-oriented API, all support functions are contained in classes. Classes, in turn, are grouped by namespace. Namespaces are not new: C++, for example, has namespaces. But with .NET, namespaces are used to organize the system service classes, thereby providing a very visible and useful organization to operating system functions. [\[Comment 1.139\]](#)

The structure and organization of the namespaces and classes help make the programming interface very *discoverable*, a term often applied to a well-designed user-interface. If, for example, you know about one method or one class, it is easy to find other, related methods or classes. For example, when trying to figure out how to create graphical output, the `System.Drawing` namespace has everything you need. For example, when you find the `DrawString` method for drawing text, then you soon find that other text drawing elements – fonts and brushes – are organized together in the same namespace. By contrast, a Win32 programmer sometimes searches long and hard to organize the functions needed for even the simplest of operations. [\[Comment 1.140\]](#)

While namespaces organize classes and code, container classes help programmers to organize their data. The various .NET frameworks make extensive use of the container classes in a wide range of settings, and this adds to the consistency and ease-of-use when writing code. For example, a window can contain other windows. A container of type `ControlCollection` exposes this containment relationship to the .NET code, which can access the collection using methods with names like `Add`, `Remove`, `Contains`, and `Clear`. The items in a listbox or a combobox are similarly available through a collection class, with a set of methods that have the same names: `Add`, `Remove`, `Contains`, and `Clear`. This consistency adds further in making .NET a very well-designed, easy to program API. [\[Comment 1.141\]](#)

Common Intermediate Language (CIL)

When you build a .NET program (.exe) file or a shared library (.dll) file, you create an executable file that outwardly is the same as Win32, MFC, or compiled VB program. This is

²³ We ignore for now the distinction between the lightweight value types and reference types.

known technically as a Win32 portable executable, or 'PE' file²⁴. Each PE file targets a specific instruction set. Some instruction sets are tied to a specific CPU family, which is true for the Intel x86, MIPS R4000, and Toshiba SH4 instruction sets. [\[Comment 1.142\]](#)

One unique aspect of .NET executable files is that the executable files are written for an instruction set not tied to a specific CPU family. This is the *IL* instruction set, which stands for 'Intermediate Language.' It has other names, including *Microsoft Intermediate Language* (MSIL), and *Common Intermediate Language* (CIL), the name used in the submission as an ECMA standard. [\[Comment 1.143\]](#)

Managed Code and Unmanaged Code

The code within an IL executable file is called *managed code*. Non-IL executables, by contrast, contain *unmanaged code*. Managed code is code that relies on the .NET runtime for automatic memory management, verification of type-safety of code, and array-boundary checking among other features. Unmanaged code, sometimes called 'native code', does not get the benefit of these features. [\[Comment 1.144\]](#)

Programmers sometimes express confusion about whether IL is interpreted or compiled. The concern is performance because interpreted languages – like SmallTalk and some versions of BASIC – run slower than compiled languages. The sluggishness of an interpreted language occurs because most of the processing of source level instructions occurs at runtime. An interpreted language is often easy to develop for, with a very interactive development environment that lets a programmer modify the code and continue running. The performance concerns, however, have largely made interpreted languages useful for prototyping but not for production code. IL provides is the portability of an interpreted language with the speed benefits of a compiled language. [\[Comment 1.145\]](#)

While an IL interpreter is possible, all current implementation of IL run as native instructions. The conversion from IL to native CPU instructions is done at runtime, on a per-function basis. Desktop versions of .NET allow for an entire IL executable to be converted to native instructions and added to the native image cache (using a tool called `ngen.exe`). This feature is not supported for Windows CE-based IL executables, however, because a typical mobile device lacks the storage capacity for a native image cache (and so the Compact Framework does not support an IL cache). [\[Comment 1.146\]](#)

IL is not the world's first portable instruction set, incidentally. Java programs are often compiled to create Java byte code executable files. At runtime, a Java Virtual Machine (JVM) converts the portable instruction set to native machine instructions. [\[Comment 1.147\]](#)

The conversion of portable instructions to native instructions is called *JITting* (the term used for both Java and .NET programs). The 'JIT' stands for "just-in-time," a term that originates in manufacturing to refer to last minute delivery of required parts. With IL, JITting is done on a per-method basis on both the desktop and the Compact Framework. By having this fine level of granularity, only the specific methods which are called get converted; methods that never get called are never JITted, saving both processing time and storage space. [\[Comment 1.148\]](#)

Once a method has been converted from IL to native instructions, the native code version of the method stays in RAM. The collection of JITted native code is referred to as the *native image cache*. This means that the cost incurred for the JITting can get amortized for methods which are called often. Native methods are not locked into memory, but rather the memory they occupy is subject to reclamation by the garbage collector just like unreachable data objects. [\[Comment 1.149\]](#)

Windows CE 3.0 was the first IL-Compatible Platform

²⁴ The ASCII letters 'PE' are the signature bytes in the Win32 header, which you can see using a hex dump of a .EXE or .DLL binary file.

Most programmers are unaware that Microsoft's first product with IL was Windows CE 3.0, under the name Common Executable Format (CEF). The Pocket PC supports converting IL to native CPU instructions. The Windows CE team adopted IL because it provided a portable, CPU-neutral language that allowed a single set of installation files to support a wide range of target CPUs. [\[Comment 1.150\]](#)

A key difference between IL and CEF is that CEF is an install-time translation, not a JITting translation. The .NET runtime, on the other hand, does run-time JITting. [\[Comment 1.151\]](#)

Common Language Runtime (CLR)

The *Common Language Runtime* (CLR) refers to a set of services which support .NET executables. Among the services it provides are the loading of an executable into memory, the JITting of its IL to native instructions, and the allocating and freeing of objects. [\[Comment 1.152\]](#)

The most commonly discussed task that the CLR handles is memory management, and in particular the reclaiming of unreachable objects known as *garbage collection*. The various .NET frameworks are not the first to implement this service: SmallTalk, Visual Basic, and Java all support garbage collection in one form or another. [\[Comment 1.153\]](#)

.NET garbage collection is important because it represents the first automatic garbage collection that is compatible with many different programming languages. As we mention elsewhere, a program written in one language can call a component written in another language, and the two can freely share high-level objects. This is possible because the two share a common memory allocator, as provided by the CLR. And that memory is properly reclaimed as part of garbage collection, again provided by the CLR. [\[Comment 1.154\]](#)

How does automatic memory management (i.e. 'garbage collection') work? When the available memory drops below a predefined threshold, the memory manager freezes all threads in a given process. At that point, it walks the stack of all threads looking for references to objects in the heap. Some objects refer to other objects, and this continues until every stack frame for every thread in the process has been taken into account. [\[Comment 1.155\]](#)

Each object that can be reached, through either a direct or an indirect reference on the stack, is live data. The live data gets compacted in the heap, and everything else is reclaimed. A benefit of this approach is that a pure memory object can be allocated and then abandoned and it gets reclaimed. No explicit operation is needed to free such objects. [\[Comment 1.156\]](#)

In some cases, however, you may want to explicitly free an object because it contains one or more references to a resource which does, in fact, require some cleanup. This might be a file that needs to be closed, a network connection that needs to be logged out, or an unmanaged library that needs to free some Win32 objects. In such cases, you implement a pair of cleanup methods to handle automatic (`Finalize`) cleanup, or manual (`Dispose`) cleanup. We discuss these two in more detail in chapter 3, *Fundamental .NET Data Types*. [\[Comment 1.157\]](#)

Common Language Specification (CLS)

The word 'common,' which appears often in .NET terms, has many meanings including 'plain' and 'ordinary.' That is not what 'common' means in .NET. A better word might be 'standard,' because these elements define a set of standards that allow a wide range of programming languages to interoperate with each other. [\[Comment 1.158\]](#)

The *Common Language Specification* (CLS), in particular, refers to a set of standards that were created to promote interoperability between different programming languages. This is a very important architectural feature of the .NET initiative. It opens up the possibility that existing code libraries can more easily be made available as managed code libraries. It also allows programmers a wider range of programming languages instead of restricting them to just a few. In the world

before .NET, the Win32 API was primarily available to C and C++ programs. Visual Basic programs were second-class citizens, because aspects of Win32 and COM were difficult if not impossible to access from Visual Basic. As Microsoft shipped new versions of VB, the gap got narrower and narrower between what a C programmer and a VB programmer could do, but it never really went away entirely in unmanaged code. In the world of managed code programming, however, that gap essentially disappears. [\[Comment 1.159\]](#)

In .NET, because it supports the Common Language Specification, Visual Basic .NET is a first-class citizen. It is fully the equal of other languages such as C#. Compact Framework programmers, for example, can write programs and libraries in one of two languages: C# or VB .NET. Both languages use the same runtime (the *Common Language Runtime*), and both support a common set of types (as defined in the *Common Type System*). A library written in VB .NET can be called from a C# program, and vice-versa. The common standards used by .NET-compatible languages are what enable this interoperability. [\[Comment 1.160\]](#)

An even larger set of programming languages is available on the desktop .NET Framework. The desktop supports C# and VB .NET, just like the Compact Framework. There is, in addition, a .NET-compatible version of the C++ compiler that supports a .NET-compatible mode known as "Managed C++." At the initial .NET announcement, there was a demonstration of a .NET-compatible web site that was written in COBOL. Other .NET languages that we know of include: Dyalog APL, Eiffel, Fortran (Salford FTN95), J++, JScript .NET, Mercury, Mondrian, Oberon, Pascal, Perl, Python, SmallTalk, and Standard ML. [\[Comment 1.161\]](#)

Common Type System (CTS)

The *Common Type System* (CTS) defines a standard set of types which are supported by all .NET-compatible compilers. The ability for .NET executables to freely interoperate between executables built from different languages relies on the common set of types defined by the common type system. The common type system is a fundamental building block for .NET programming. We provide a more complete discussion of the common type system in chapter 3 (*Fundamental .NET Data Types*). [\[Comment 1.162\]](#)

Common Language Infrastructure (CLI)

The *Common Language Infrastructure* (CLI) describes a subset of the .NET Framework which Microsoft submitted as a standard to ECMA. It does not include the Windows Forms classes, the Web Forms classes, or the Web Services classes. What it does include is enough of the low-level details to allow third-parties to create .NET-compatible compilers, debuggers, and other tools. The CLI does include the two topics we just discussed, the *Common Language Specification* and the *Common Type System*. [\[Comment 1.163\]](#)

Programmers who are eager to learn more about the *Common Language Infrastructure* can find a set of Word files which spell out in detail the various elements. Those files are included with Visual Studio .NET 2003, and can be found in the program file directory in the `..\SDK\v1.1\Tool Developers Guide\docs` directory. [\[Comment 1.164\]](#)

The .NET Compact Framework

The two previous sections of this chapter focus on two key ingredients: the Windows CE operating system, and Microsoft's .NET initiative. For the rest of this chapter, we see what happens when these two are mixed together. The results are an important improvement in support for smart device programming. [\[Comment 1.165\]](#)

Sidebar: Why '.NET' in Windows CE .NET?

The full name for the latest version of Windows CE is 'Windows CE .NET.' One question that we encounter often has to do with why the '.NET' is included in this name. The reason is simple: Microsoft sees Windows CE-powered devices – including, of course, the Pocket PC – as part of its larger .NET strategy. Using the Compact Framework, programmers can build traditional GUI applications that consume the services of XML Web service servers. [\[Comment 1.166\]](#)

With two different browsers – the Pocket Internet Explorer and the Generic Internet Explorer (GENIE) – Windows CE-powered devices can also run Web Forms applications. An add-on to the ASP.NET web server, ASP.NET Mobile Controls, allows for server-side customizations to web-based applications for small-screen devices. [\[Comment 1.167\]](#)

The '.NET' in the operating system name suggests to some people that perhaps Windows CE is going to be rewritten to run entirely in managed code, but we do not expect that to ever happen. The reason is simple: a huge investment has been made in creating the Win32 infrastructure, including the core OS and existing device drivers. There is no business case for rebuilding something that already works well. For the future, we expect that Win32 will always be the core of Windows CE; and Compact Framework will always rely on Win32 for its low-level support. [\[Comment 1.168\]](#)

Design Goals

When the Microsoft team started to develop the Compact Framework, various tools were already being used for applications on smart devices. Embedded Visual C++ was the development tool of choice for creating Win32 and MFC-based software. Another development tool, Embedded Visual Basic, was also being used to create smart-device applications. On the desktop, the development of the .NET Framework – and its companion, Visual Studio .NET – was well underway and was going to be released well in advance of the Compact Framework. The existing and soon-to-be-released tools were important because each would play a role in setting expectations on the software development population that the Compact Framework team was targeting with its new framework. The Compact Framework team made plans to address the perspectives of each group of developers. [\[Comment 1.169\]](#)

Developers who had adopted Embedded Visual Basic were expected to make up the largest group of Compact Framework developers. With its support for Visual Basic, easy to use development environment, and high level of portability, Microsoft decided to discontinue support for Embedded Visual Basic. As of this writing, the Pocket PC 2003 is the last device for which the Embedded Visual Basic runtime will ship. For this group, the Compact Framework team set out to provide a development experience that was as easy to use as Embedded Visual Basic. [\[Comment 1.170\]](#)

To address the needs of developers with experience with the desktop .NET Framework, the team worked to maintain a high level of consistency with the existing framework. They knew that a Windows CE-based framework would not be as extensive as the .NET Framework. But they could make sure that the namespaces, classes, and other types found in the Compact Framework had a high level of consistency with corresponding elements of the desktop framework. [\[Comment 1.171\]](#)

The third group of developers consisted of developers using C and C++ to develop Win32 and MFC applications with Embedded Visual C++. The Compact Framework team expected this to be the smallest of the groups to adopt the Compact Framework. For those who did switch, the

team wanted to allow the development of applications that had the power and speed approaching native Win32 applications. Towards that end, a significant amount of performance tuning was done to help make the Compact Framework attractive to this third, and perhaps most demanding, developers. [\[Comment 1.172\]](#)

The Compact Framework team adopted these design goals for the Compact Framework:

- Build on the Benefits of the .NET Framework [\[Comment 1.173\]](#)
- Maintain consistency with the desktop [\[Comment 1.174\]](#)
- Run well on mobile and embedded devices [\[Comment 1.175\]](#)
- Expose the richness of target platforms [\[Comment 1.176\]](#)
- Preserve the look & feel of platforms [\[Comment 1.177\]](#)

Build on the Benefits of the .NET Framework

A relatively recent operating system, Windows CE has always borrowed heavily from existing desktop tools and technologies. Doing so has allowed Windows CE to benefit from existing code, existing tools, and existing programmer skills. The Win32 API, for example, was created for the desktop, but became the centerpiece of Windows CE. And the various Pocket applications on the Pocket PC, such as Pocket Word, Pocket Excel, and Pocket Outlook, were created by porting from existing desktop applications. [\[Comment 1.178\]](#)

All of the standard 'common' elements of the desktop .NET Framework are also found in the Compact Framework. Executable files use the CIL instruction set, which gets brought into memory and JITted into native machine instructions by the Compact Framework's Common Language Runtime. Memory gets reclaimed by automatic garbage collection, and all the standard which are spelled out in the Common Language Infrastructure (CLI) are supported in the Compact Framework. This includes the Common Type System as well as the Common Language Specification. While the framework is not as extensive as that on the desktop, the operation of the underlying runtime is identical. [\[Comment 1.179\]](#)

Maintain Consistency with the desktop

The second design goal of the Compact Framework was to maintain consistency with the experience found in the desktop framework. Programmers experienced with desktop .NET Windows Forms applications will find many old friends in the .NET Compact Framework. Both share all of the same basic value types, most of the same namespaces, and many of the same classes. [\[Comment 1.180\]](#)

The Compact Framework development team worked hard to make the Compact Framework consistent with its desktop counterpart. For one thing, two frameworks share an identical structure. The names for Compact Framework namespaces, for example, mirror existing desktop namespaces. In scaling back the framework to fit within a small, 2MB footprint, the team eliminated many of the desktop framework classes. Within the remaining classes, the team trimmed back available class members. Most Compact Framework classes have fewer properties, methods, and events than their desktop counterparts. [\[Comment 1.181\]](#)

You can get a sense for how consistent the two frameworks are by studying the online documentation. The high level of consistency allows a single documentation set to support both frameworks. As figure 1-3 illustrates, class members with Compact Framework support are labeled with 'Supported by the .NET Compact Framework.' [\[Comment 1.182\]](#)

Figure 1-3 Online documentation highlights Compact Framework compatibility

The screenshot shows a window titled ".NET Framework Class Library" with a sub-header "Marshal Members". It contains a table with three rows, each representing a method:

<u>Copy</u> Supported by the .NET Compact Framework.	Overloaded. Copies data from a managed array to an unmanaged memory pointer.
<u>CreateWrapperOfType</u>	Wraps the specified COM object in an object of the specified type.
<u>DestroyStructure</u>	Frees all substructures pointed to by the specified unmanaged memory block.

By maintaining a high level of consistency, the Compact Framework team hoped to leverage the knowledge and programming skills of existing .NET programmers. Some things are immediately familiar to desktop .NET programmers because both frameworks support C# and VB .NET, share the same set of common types, and share many namespaces and classes. Programmers who adopt the desktop .NET Framework are highly qualified candidates to get productive quickly with the Compact Framework. [\[Comment 1.183\]](#)

But because the Compact Framework supports a significantly reduced feature set from the desktop framework, desktop framework programmers need to allow time to get comfortable with the Compact Framework. It takes time because many favorite classes and class members are 'missing.' Desktop framework programmers are likely to 'stub their toes' quite a number of times before they get used to the reduced set of supported classes and supported class members which the Compact Framework provides them. [\[Comment 1.184\]](#)

While the move from the desktop to smart devices can be frustrating, the move going the other way is likely to be much less of a problem. Programmers who start their .NET career working with the Compact Framework will move to the desktop framework quite easily. Because of the consistency between the desktop and the Compact Framework, and the fact that the desktop is a true superset of the Compact Framework, a Compact Framework program can run unchanged on the desktop²⁵. [\[Comment 1.185\]](#)

Windows CE-specific extensions in the Compact Framework are defined in namespaces other than those used in the desktop framework. For example, the `Microsoft.WindowsCE.Forms` namespace contains Windows CE-specific controls and structures. This namespace ships in the `Microsoft.WindowsCE.Forms.dll` library, which contains three elements: a `MessageWindow` class (for sending Win32 messages to managed code), the `Message` structure (for use with the `MessageWindow` class), and `InputPanel` a wrapper class for the Pocket PC Software Input Panel (SIP). [\[Comment 1.186\]](#)

And the same tool used for desktop .NET development, Visual Studio .NET, is also used to build Compact Framework applications. The Visual Studio .NET features that desktop programmers have come to rely upon are available for Compact Framework development, including Intellisense, the Windows Forms Designer (with device-specific controls in a Compact Framework-specific toolbox), the object browser, and an integrated debugger. [\[Comment 1.187\]](#)

²⁵ This requires version 1.1 of the .NET Framework. A Compact Framework can run unchanged under the desktop Framework, provided that P/Invoke declarations referencing device-only libraries are not present.

Run well on mobile and embedded devices

Another important Compact Framework design goal was to run well on mobile and embedded devices. This meant addressing the two key challenges for any small device: size and speed. Size is an issue because at 25+ megabytes, the desktop Framework would have to be scaled down to fit in the 32MB to 64MB storage available on a typical mobile device like a Pocket PC or other Windows CE-powered device. Speed is an issue because even the fastest mobile and embedded CPUs are slower than the CPU in an average desktop system. To provide acceptable performance to users of fast desktop systems, the Compact Framework team spent time tweaking the performance of the Compact Framework. [\[Comment 1.188\]](#)

At 2MB, the Compact Framework has met the size goal. In working towards that goal, the Compact Framework team took a page from the playbook of the Windows CE team. Namely, if there were two or more ways to do the same operation, most get removed. On the desktop, the .NET Framework has the luxury of large hard drives and seemingly endless amounts of RAM. It can support, for example, several different ways to change the background color for most (but not all) controls. On the desktop, you could set the `BackColor` property, handle the `PaintBackground` event, or override the `OnPaintBackground` method. In the Compact Framework, to save space, there is one: override the `OnPaintBackground` method in a derived class. Some controls support the `BackColor` property – including `Form`, `Control`, `Panel`, `TabPage`, and `TextBox` – but most do not. [\[Comment 1.189\]](#)

On its way to achieving the size goal, the Compact Framework team left out desktop-centric features such as drag-and-drop, and large controls like the `RichTextBox`. For each supported controls, only the most basic functionality is supported. In particular, a focused subset of the Properties, Methods, and Events – the "PMEs" as the Compact Framework team calls them – are supported in Compact Framework controls. [\[Comment 1.190\]](#)

To meet the performance goals, the Compact Framework was benchmarked and tuned so that controls rely heavily on their Win32 counterparts. The Win32 controls reside in unmanaged code, and Compact Framework code runs as managed code. There is a cost in crossing the boundary between managed and unmanaged code. The Compact Framework team found that they could enhance the performance of the controls by allowing only a carefully controlled subset of Win32 messages to percolate as .NET events. [\[Comment 1.191\]](#)

Expose the richness of target platforms

The fourth design goal was to expose the richness of a target platform. In every case, the Compact Framework relies on native controls to do the majority of the work. This has obvious benefits in size and performance. What it also provides is a more authentic look and feel for Compact Framework application on each platform. [\[Comment 1.192\]](#)

For example, the Compact Framework `MainMenu` class provides the basic support for application menus. In a Pocket PC application, this menu appears – as in any other Pocket PC application – at the bottom of the window. On a (non-Pocket PC) Windows CE .NET device, that same Compact Framework program displays its main menu at the top of the window like standard Windows CE .NET applications do. While the underlying Win32 menu implementations are different, the Compact Framework conveniently hides this difference from developers through the Compact Framework `MainMenu` and `MenuItem` classes. [\[Comment 1.193\]](#)

Preserve the look & feel of platforms

The fifth design goal was to preserve the look & feel of individual platforms. The challenge comes from subtle differences in the way the native controls are implemented. Many of these changes involve tiny, almost imperceptible differences – an extra pixel here, a pixel left out there. Such differences are invisible when looking at two applications running on two different devices. But when two applications are running on the same device, such differences become readily apparent. The Compact Framework team worked hard to make sure that the look & feel of

different platforms was honored by the Framework. To a user, a Compact Framework application is visually indistinguishable from non Framework applications. [\[Comment 1.194\]](#)

Compact Framework Files

The Compact Framework itself consists of a set of dynamic link libraries. A few are CPU-specific native code libraries, but most of the supporting libraries are managed code libraries. Table 1-2 summarizes the Compact Framework modules, as installed by Visual Studio .NET 2003. Two of these modules, `ConManClient.exe` and `cmtnpt_tcpaccept.dll`, are used by Visual Studio .NET to communicate between a development workstation and a smart device. One of these modules, `System.SR.dll`, contains a set of string resources to display the meaning of unhandled exceptions. This is something that can help you when developing Compact Framework software to help identify unhandled exceptions in your code. The software you ship should catch all exceptions and handle them internally. So the error message library is not something that needs to be installed on end-user systems. [\[Comment 1.195\]](#)

Table 1-2 Compact Framework Modules

File Name	Description	Size (Kbytes)	Managed Code
<code>cgacutil.exe</code>	Compact Global Assembly Cache utility. Add a managed code library to global assembly cache to save device storage space. Base Compact Framework libraries are installed in this way. Requires a file with an extension <code>.GAC</code> and the name of the managed code library (must be signed with strong names).	8	No
<code>ConManClient.exe</code>	Connection manager client, a device-side program to assist in connecting a device to Visual Studio .NET to support the download and debug of managed code modules.	36	No
<code>cmtnpt_tcpaccept.dll</code>	Connection manager custom transport (helper for <code>ConManClient.exe</code>)	11	No
<code>Microsoft.VisualBasic.dll</code>	Visual Basic-specific helpers, including constants (<code>vbCr</code> , <code>vbCrLf</code>), built-in VB functions (<code>Rnd</code> , <code>Mid</code> , <code>UCase</code> , <code>NPV</code>), among others.	136	Yes
<code>Microsoft.WindowsCE.Forms.dll</code>	Windows CE-specific classes. Supports Software Input Panel (SIP) on a Pocket PC (a.k.a. 'InputPanel'). Supports sending Win32 messages, and creation of window procedure to receive messages sent from unmanaged code.	11	Yes
<code>mscorlib.dll</code>	Stub for <code>mscorlib_0.dll</code>	13	No
<code>mscorlib_0.dll</code>	Execution engine and Platform Adaptation Layer (PAL) for Compact Framework version 1.0	461	No
<code>mscorlib.dll</code>	Low-level .NET support classes: data collections, file I/O, threads, application domains, object activator, garbage collector, globalization support, platform invoke, text character set conversion (encoding and decoding), resources,	383	Yes

File Name	Description	Size (Kbytes)	Managed Code
	standard system exceptions, definitions for common types, and basic types including Object and ValueType.		
netcfag11_0.dll	Native code support for user-interface library, System.Windows.Forms.dll	107	No
System.Data.Common.dll	Generic set of classes for accessing a data source	106	Yes
System.Data.dll	Core support for ADO.NET programming interface for in-memory database operations. Primarily relational operations on in-memory tables, including support for view and joins.	394	Yes
System.Data.SqlClient.dll	(Optional) Data provider for remote, server-based database for Microsoft SQL Server™ 7.0 or later.	145	Yes
System.Data.SqlServerCe.dll	(Optional) Data provider for a local, device-based database for Microsoft SQL Server 2000 Windows CE Edition (SQL Server CE).	121	Yes
System.dll	Low level .NET support for specialized collection classes, diagnostic helpers, low-level network connectivity classes including sockets, and regular expressions.	249	Yes
System.Drawing.dll	Provides Graphics object, the core drawing object. Also provides drawing support objects including bitmaps, colors, fonts, pens, brushes, and icons.	38	Yes
System.Net.IrDA.dll	Infrared device support in the form of IrDA-compatible sockets and endpoint support.	11	Yes
System.SR.dll	(Optional) String resources for exceptions to help develop and debug applications.	91	Yes
System.Web.Services.dll	Support for creating SOAP-enabled web service clients.	94	Yes
System.Windows.Forms.DataGrid.dll	Data grid control and supporting classes.	38	Yes
System.Windows.Forms.dll	Core library for Windows Forms application, including Application object and user-interface controls.	137	Yes
System.Xml.dll	XML schema and serialization support, along with XML reader, and XML writer	197	Yes

The size of the standard Compact Framework library components is around 2,383 Kbytes. Adding in the data providers for SQL Server and SQL Server CE, the size grows to just about 2.5 Mbytes. The actual space required is probably smaller than this, because device storage adds a bit of compression that is fast but light, yielding a 2:1 compression on average. [\[Comment 1.196\]](#)

When you study the file system of a device with Compact Framework installed, you find all of the unmanaged modules in the \windows directory. No matter how hard you try, however, you do not find the managed modules listed in table 1-2 anywhere on a device. A search for

`System.dll`, for example, yields no result. What you *do* find is a file with the intriguing name of `GAC_System_v1_0_5000_0_cneutral_1.dll`. [\[Comment 1.197\]](#)

The 'GAC' stands for *Global Assembly Cache*, the name for the set of cached shared libraries. (In .NET, the term *Assembly* refers to one or more modules – EXEs or DLLs - that make up a logical unit²⁶.) As on the desktop, the GAC in the Compact Framework identifies a set of libraries which have been marked for sharing. When libraries are placed in the GAC, the loader does some verification of the file's contents. The primary benefit of putting shared files in the GAC is reduced storage required by keeping just one copy of a shared library. [\[Comment 1.198\]](#)

If you study the name of the GAC file, you see that the name contains the version number of the library (1.0.5000.0). It also contains details about what culture the library targets. This library has a *neutral* culture, which means the library itself is not localized for any particular language or country. What we do not see in the file name is the key used to sign the library. The inclusion of these values – a version number, culture, and signing key – in addition to the friendly name of the library allows the CLR to uniquely identify each library. [\[Comment 1.199\]](#)

Two additional, unseen elements are required for a library to be installed in the GAC: a public key and a digital signature. These last two items uniquely identify the developer of the library, and also help verify that the contents of the library have not been tampered with or accidentally damaged in some fashion. [\[Comment 1.200\]](#)

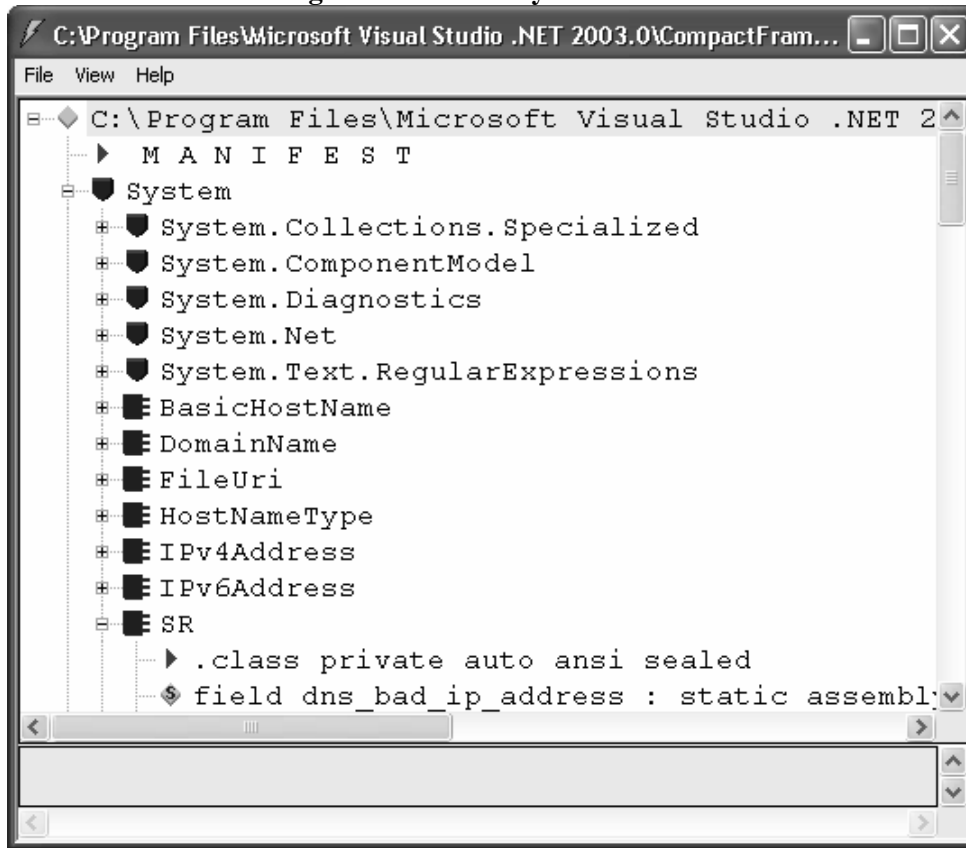
Installing Shared Libraries in the Global Assembly Cache

You can add your own shared libraries to the system's global assembly cache by creating a file with an extension of `.gac`. The file should be a text file with the full path of the library you want converted (for multiple libraries, put each library on a separate line). Copy that file, along with your libraries, to the `\windows` directory of a device. When the library is next loaded into memory, the loader creates the required `GAC_` file for your library. Only signed libraries with strong names can be added to the GAC. Two example files are created by the Compact Framework itself:

`Microsoft .NET Compact Framework 1.0.GAC` and
`System.SR.ENU.gac`. [\[Comment 1.201\]](#)

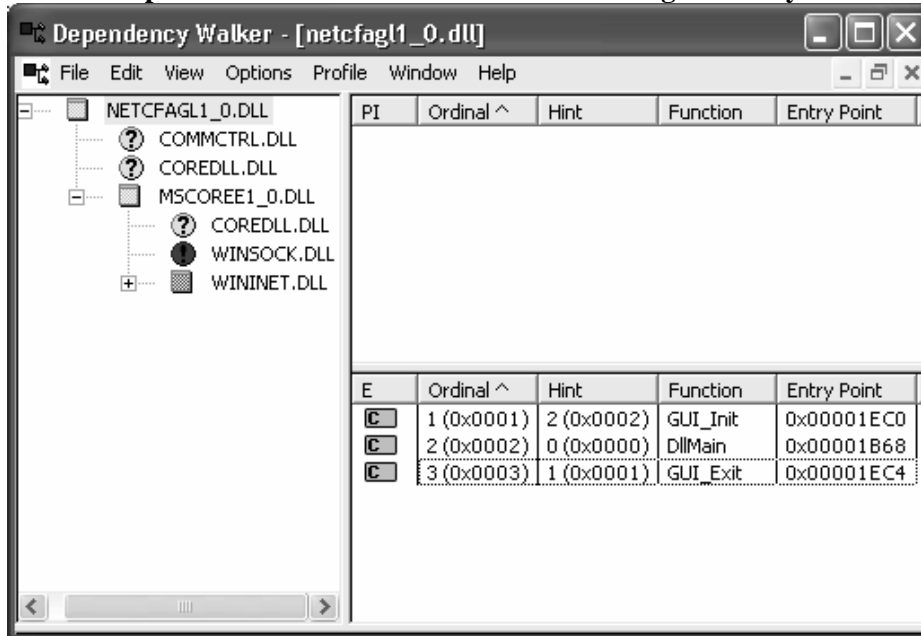
If you are just getting started with managed code development, you should know about a utility that Microsoft ships with Visual Studio .NET. The name of the utility is `ILDASM.EXE`, which stands of 'IL Disassembly'. This tool helps answer the question 'What is inside a given managed module?' Figure 1-4 shows this tool in action, displaying an outline of the contents of the Compact Framework's `system.dll` library. [\[Comment 1.202\]](#)

²⁶ We use the term library here because developers new to .NET are normally not familiar with the newer term.

Figure 1-4 ILDASM showing the contents of system.dll

ILDASM provides a lot of details about a managed code library, because a managed code module contains all available type information encoded as *metadata* within the module file itself. You can find out the namespaces and classes within a module, and within each class you can find out the type information for each of the class members, including the names of both code and data (methods and fields). For methods, ILDASM disassembles the method to show the associated IL for the method. (To make it harder for competitors to reverse-engineer your code, obfuscator utilities jumble the metadata names in a managed code module.) [\[Comment 1.203\]](#)

A similar tool for unmanaged libraries is DEPENDS.EXE, which also ships with Visual Studio .NET. This tool does not provide the same depth of information as ILDASM, because much of the type information in unmanaged modules gets lost during the compile and link process. What you can find out, however, is the names of exported (public) functions, the names of libraries that an unmanaged module depends on, and the functions that are imported from the libraries. Figure 1-5 shows the output from this tool for one of the Compact Framework's unmanaged libraries, netcfag11_0.dll. [\[Comment 1.204\]](#)

Figure 1-5 Output from DEPENDS.EXE for an unmanaged library

Programmers who have worked with the desktop .NET Framework can probably get a good idea about the Compact Framework capabilities and limitations just by poking around with these tools. In some cases, you have to do that so that you know exactly what is present. The online documentation also provides quite a bit of useful information, and as you browse through the various classes you may notice the 'Supported by the .NET Compact Framework' tag in quite a few places. [\[Comment 1.205\]](#)

While we recognize that some of our readers are going to have experience with the desktop .NET Framework, we do not assume that is going to be the case for most readers. To help the general reader start planning their smart-device development efforts, we turn our attention to a summary of what the Compact Framework can – and cannot – do. We start on a positive note, with the Compact Framework Capabilities. [\[Comment 1.206\]](#)

Compact Framework Capabilities

The Compact Framework was designed to be consistent with the desktop .NET Framework. Like it or not, the desktop framework always seems to be lurking in the background, like a worried parent or a homeless puppy. A crude way to compare the frameworks is by number of files and disk space occupied. The Compact Framework has 18 libraries (including optional ones) and occupies about 2.5 megabytes. The .NET Framework version 1.1, by contrast, has 86 libraries and occupies around 40 megabytes. [\[Comment 1.207\]](#)

Supported Runtime Elements

So what can the Compact Framework library do for us? First, the Compact Framework implements 100% of the Common Language Infrastructure (CLI). So the standard data types found on the desktop are also present for smart device programming. At present, Microsoft supports two programming languages for the Compact Framework, C# and Visual Basic .NET. This is quite a reduction from the twenty or more languages which support desktop .NET Framework. And yet, support for C# and VB .NET in the Compact Framework shows that interoperability between languages is more than a promise; in the Compact Framework, as on the desktop, we have direct evidence in the form of working, commercial quality compilers where

assemblies created in one language can freely interoperate with assemblies created in other languages. [\[Comment 1.208\]](#)

The Compact Framework supports an execution engine that provides JIT compilation of IL, and which verifies the type-safety of managed modules before loading them. The runtime for the Compact Framework provides the same memory management as on the desktop, meaning memory allocation, heap management, and garbage collection of unreachable objects. [\[Comment 1.209\]](#)

The Compact Framework supports *Application Domains* (a.k.a. 'AppDomains'), which is a kind of lightweight process. Multiple AppDomains can run in a single operating system process. On some operating systems, processes have high overhead and the cost of context switching from one process to another is high. That certainly describes processes on desktop versions of Windows, and the AppDomain provides a lightweight alternative. [\[Comment 1.210\]](#)

A process in Windows CE is already lightweight. As far as the CPU is concerned, all processes in Windows CE reside in a single process. So is there a benefit to using AppDomains in the Compact Framework? The answer is: yes. A side effect of having lightweight processes is that there is a fairly low limit to the number of processes that can be created on a Windows CE system. That limit is 32, and is so tightly ingrained in the memory and process architecture of Windows CE that we usually call this a *hard limit*, by which we mean that this limit is unlikely to change in future versions of the operating system. The operating system itself uses a few of these processes, which leaves 24 or 25 available for application programs. A Compact Framework programmer might use AppDomains to avoid using up all available operating system processes for large, complex applications. [\[Comment 1.211\]](#)

On the desktop, an unmanaged process can load the CLR into a process (using the `CorBindToRuntimeEx` function). This is how, for example, the managed code of ASP.NET Web Forms applications can be loaded into the unmanaged address space of the IIS web server process (`inetinfo.exe`). This feature is not supported in version 1.0 of the Compact Framework. [\[Comment 1.212\]](#)

The Compact Framework runtime does support Platform Invoke (P/Invoke), the ability for managed code to call through to unmanaged code. There are some limitations in comparison to the desktop. One involves the complexity of parameters which can be passed. Simple value types can be passed as parameters, as you might expect. Most reference types can also be passed as parameters, including strings, structures, and objects. But complex structures and complex objects – meaning those with embedded structures or embedded objects – cannot be marshaled automatically²⁷. We discuss this subject in more detail in chapter 4 (*Platform Invoke*). [\[Comment 1.213\]](#)

On the subject of interoperability, the Compact Framework does not support COM Interop. For the present, the recommended approach to accessing COM/ActiveX components is to create an unmanaged wrapper around the component, and then export a set of C-callable functions. This approach, though tedious, does allow use of existing components with a relatively small amount of effort in the form of this glue layer. Microsoft has publicly announced that additional support for COM will be provided in a future version of the Compact Framework, although details about that support have not yet been made public. [\[Comment 1.214\]](#)

Supported .NET Application Classes

Among the three .NET application classes, the Compact Framework supports Windows Forms – traditional GUI applications. Support on both the desktop and in the Compact Framework is provided by various classes in the `System.Windows.Forms` namespace. Of the

²⁷ Compact Framework does not support the `MarshalAs` attribute. Complex objects can still be marshaled, using `IntPtr` and `static` (for VB programmers, `Shared`) members of the `System.Runtime.InteropServices.Marshal` class.

35 controls which are supported on the desktop, 28 are supported in the Compact Framework. These controls are not supported in their entirety, however. To accommodate the size and performance constraints of mobile and embedded development, a subset of the desktop class members – the properties, methods, and events or 'PMEs' – is supported in the Compact Framework. [\[Comment 1.215\]](#)

The creation of Windows Forms applications is the subject of part 2 in this book, *Building the User-Interface*. We discuss the supported controls in chapter 7 (*Inside Controls*), and present a tool in chapter 9 (*Inside More Controls*) named `ControlPME` for testing each Compact Framework control for supported properties, methods, and events. This tool is necessary because quite a few Compact Framework controls inherit members which are not supported (this is also true of desktop controls, but to a lesser extent). [\[Comment 1.216\]](#)

Another .NET application class which the Compact Framework supports is XML Web services. A Compact Framework program can create an XML Web services client, a subject we discuss in chapter 14 (*Building XML Web service Clients*). For the present, the Compact Framework supports providing the client side of a web service, but the Compact Framework does not support implementing a device-based Web service server. (Windows CE does support this, however, in unmanaged code with the SOAP Toolkit, version 2.0.) [\[Comment 1.217\]](#)

The Compact Framework has no support for Web Forms applications. As of this writing, this third application class can only be implemented on ASP.NET-compatible web servers. The following operating systems can host a Web Forms-compatible web server: [\[Comment 1.218\]](#)

- Windows 2000 (Professional, Server, Advanced Server) [\[Comment 1.219\]](#)
- Windows XP Professional [\[Comment 1.220\]](#)
- Windows 2003 Server [\[Comment 1.221\]](#)

As deployed, a Web Forms application is deployed on a web server system, not on a web client system. [\[Comment 1.222\]](#)

While the Compact Framework does not support Web Forms applications, Windows CE devices – including Pocket PC – can certainly run as Web Form clients. All that is required is a suitable web browser. Windows CE supports two basic browsers: the Pocket Internet Explorer (PIE) is the smaller, lightweight browser that ships with Pocket PC. A more fully featured browser, the Generic IE browser, is a port of the desktop Internet Explorer. This larger, more fully-featured browser provides a web browsing experience more like that found on the desktop. [\[Comment 1.223\]](#)

Microsoft has created a set of ASP.NET controls to address the small-screen web browsers like Pocket PC. These controls are known as the ASP.NET Mobile Controls, and they support the creation of web pages for three separate types of web browsers: (1) HTML 3.2 (which includes the Pocket PC), (2) Compact Html (CHTML), a lightweight version of HTML found on some devices, and (3) WML/WAP – Wireless markup language/ Wireless application protocol, such as is supported on mobile phones. [\[Comment 1.224\]](#)

Using the ASP.NET Mobile Controls, an ASP.NET web server can support a set of web pages for a wide range of small-screen devices. Further discussion is beyond the scope of this book. An important point to keep in mind is that a web site which supports desktop browsers as well as mobile browsers will need two sets of web pages: a set for the desktop browsers and a set for the mobile browsers. The reason is that the ASP.NET Mobile Controls does not reformat existing web pages to accommodate smaller screen sizes; instead it supports the creation of an entirely new set of pages for the specific needs of those devices. [\[Comment 1.225\]](#)

Graphical Output

Windows CE has a pretty capable graphics engine, although it is nowhere near the extensive features provided by the GDI+ support of the .NET Framework. What it does provide, though, is

reasonably complete support for the three basic drawing families: text, raster, and vector.

[\[Comment 1.226\]](#)

TrueType fonts are supported for drawing text, although most smart devices are not going to have the same number of fonts that are available on the desktop. A typical Pocket PC, for example, ships with 4 fonts (one of which is a symbol font). And while Windows CE supports rotating TrueType fonts and also displaying the ClearType font technology, neither feature is exposed by the Compact Framework. In chapter 17 (*Text and Fonts*), we provide the basics of drawing text, and also show how to drill through to the underlying libraries for some of the extra features. [\[Comment 1.227\]](#)

For raster graphics – meaning bitmaps – the Compact Framework is able to work with the standard set of Device Independent Bitmap (DIB) formats. What the Compact Framework cannot do is rotate bitmaps, because that support is not present in the underlying Windows CE.

[\[Comment 1.228\]](#)

For vector graphics, the Compact Framework can draw most of the same basic set of outlines and filled figures as on the desktop. Vector objects cannot be rotated, however, because once again the underlying Windows CE graphics support does not provide this. [\[Comment 1.229\]](#)

The desktop framework supports a wide range of drawing coordinates. You can draw using pixel units, inches, millimeters, or in printer's points. The Compact Framework only supports drawing in pixels (except for font sizes, which are specified in printer's points). As with other limitations in graphics support, the Compact Framework limitation on drawing coordinates are a direct result of what the underlying Windows CE drawing functions support. [\[Comment 1.230\]](#)

We dedicate part 4 of this book (*Creating Graphical Output*) to the subject of drawing. Programmers need to do draw when displaying output directly in a form, and also when creating custom controls. [\[Comment 1.231\]](#)

Supported ADO.NET Elements

The largest portion of the Compact Framework is the libraries which support ADO.NET, Microsoft's architecture for managing data from multiple data sources. Bill Vaughn, an author well known for his VB and ADO expertise, suggested to us that XDO might have been a more accurate name (for *XML Data Objects*) because XML serves as its underlying storage format. You get a hint of that from this quote, taken from an MSDN article: [\[Comment 1.232\]](#)

"In effect, the DataSet is an in-memory relational structure with built-in Extensible Markup Language (XML) support."²⁸ [\[Comment 1.233\]](#)

The primary ADO.NET data management class is `DataSet`, which hold in-memory collections of one or more `DataTable` objects. A data table, in turn, is a collection of rows and columns. Tables in ADO.NET are structured according to a relational database model. Relationships between the `DataTables` are specified by creating `DataRelation` objects. ADO.NET does not have a built-in ad-hoc query language, such as SQL. Instead, data gets read and written using members of objects created from the various ADO.NET classes.. [\[Comment 1.234\]](#)

A set of ADO.NET objects can be created from scratch, using various classes from the `System.Data` namespace. While the idea of creating an in-memory data table might seem like a lot of work, it can simplify the handling of data arrays. The `DataGrid` control, for one thing, provides an easy way to display such data. The data grid in Compact Framework supports data display but not data editing, unlike the desktop data grid which supports both display and editing of data. We discuss the `DataGrid` control and binding a table to that control, in chapter 8 (*Data Binding*). [\[Comment 1.235\]](#)

²⁸ From ".NET Data Access Architecture Guide" by Alex Mackman, Chris Brooks, Steve Busby, and Ed Jezierski, Microsoft Corporation, October 2001.

The use of in-memory data tables forms the basis for a variety of .NET-based application software, including the ixio Smart Data Machine from ixio Corporation²⁹. The architecture for this client/server system provides a way to encapsulate business logic in a manner analogous to the way that DBMS software encapsulates the data management for applications. In the Smart Data Machine, the results returned by all transactions are of type `DataSet`. In the simplest of cases, a data set contains one data table, which in turn has one row and one column: a simple scalar value. As dictated by the needs of individual transactions, data tables can contain multiple rows and/or columns, and data sets can contain multiple data tables. This approach provides a flexible mechanism for packaging up client/server data in a simple, elegant manner. [\[Comment 1.236\]](#)

A more traditional approach to using ADO.NET is to access a permanent database residing in more permanent storage. For this, ADO.NET requires the support of a data provider. A data provider is a kind of logical device driver for a database. Data providers enable connections to databases, the execution of database commands, and access to the results. The Compact Framework ships with two data providers. [\[Comment 1.237\]](#)

One data provider, `System.Data.SqlClient.dll`, supports using databases managed by SQL Server (version 7.0 or later) as the data source. SQL Server does not run on Windows CE, but rather runs on desktop and server versions of Microsoft Windows. The use of SQL Server as a data source requires a live network connection between the Windows CE system and the SQL Server system. [\[Comment 1.238\]](#)

A second data provider, `System.Data.SqlServerCe.dll`, supports using local databases managed by SQL Server 2000, Windows CE Edition (a.k.a. 'SQL Server CE'). This database runs on the Windows CE system itself, with database files stored in the Windows CE object store or on an installable file system. The only requirement for using this data source is that SQL Server CE be installed on the local Windows CE system. (A side note: SQL Server CE databases are not the same as Windows CE property databases, sometimes referred to as CEDB databases.) [\[Comment 1.239\]](#)

The desktop .NET Framework supports two other data providers which are not, as of this writing, supported in the Compact Framework. The data providers that are not supported are the ODBC data provider and the OLE DB data provider. We discuss database programming in more detail in chapter 12 (*Local Data Access*) and chapter 13 (*Remote Data Access*). [\[Comment 1.240\]](#)

XML Support

An important data format for .NET programming is XML³⁰, the *eXtended Markup Language*. XML provides a standard way to create HTML-like custom tags. While HTML structures text and images for display in a web browser, XML structures data for programmatic access. HTML, for example, is the format for Windows CE help files. XML, on the other hand, provides the format for storing data that can be shared between computers in a platform-neutral way. [\[Comment 1.241\]](#)

There are a great many uses of XML in the computer industry today, including Microsoft's BizTalk Server, which provides an XML-based mechanism for exchanging documents (such as purchase orders and invoices) between companies. Microsoft SQL Server can use XML to send and receive data, which means any computer – whether running Microsoft Windows or not – can query and update a SQL Server-based database. [\[Comment 1.242\]](#)

There are myriad uses of XML within the .NET world. We mentioned one earlier: XML is used for ADO.NET data. Another well known use of XML is within Web services, commonly

²⁹ www.ixio.com.

³⁰ A very good summary of .NET support for XML appears in Aaron Skonnard's article *XML in .NET: .NET Framework XML Classes and C# Offer Simple, Scalable Data Manipulation*, MSDN Magazine, January 2001.

referred to as XML Web services. SOAP, the underlying protocol for Web services, is defined in terms of XML datagrams. An interesting feature of .NET support for Web services is that while XML is used as the underlying format, the use of XML itself is buried within the Web service support classes. We discuss Web services in detail in chapter 14 (*Building XML Web Service Clients*). [\[Comment 1.243\]](#)

The extensibility of XML means that programmers can create custom tag dialects to suit a specific set of needs, and structure the tags needed. The ability to create custom tags introduces the need to validate an XML dataset with a specific dialect. While the desktop .NET Framework supports both DTD and schema approach to validating XML – in the form of the `XMLValidatingReader` class – the Compact Framework supports neither type of validation. [\[Comment 1.244\]](#)

The Compact Framework supports the XML Document Object Model (DOM), which allows an entire XML document to be read into memory and manipulated. Through its support of the `XMLDocument` class, the Compact Framework supports traversing a RAM-resident XML hierarchy. Significantly, the Compact Framework does not support the `XMLDataDocument` class, which loads an XML document into an ADO.NET data set. This additional support requires the presence of an XML schema to convert the XML document into the relational structure of ADO.NET. XML schemas, however, are not supported in the Compact Framework. [\[Comment 1.245\]](#)

The desktop framework provides XML document query support through various classes in the `System.Xml.XPath` namespace. On the desktop, you use these classes to step through a subset of nodes which match specific query criteria. However, the Compact Framework supports none of the XPath features. [\[Comment 1.246\]](#)

Strictly speaking, the .NET Framework and Compact Framework do not support SAX, the *Simple API for XML*. SAX was created before the .NET Framework, as an alternative to the XML DOM. SAX provides a forward-reading, push-model 'firehose' approach to parsing an XML document. What .NET provides – both on the desktop and for the Compact Framework – is the pull-model `XmlTextReader` class. You use this class when searching an XML document for a specific value or attribute, when you do not want the memory overhead of reading an entire XML document into memory. [\[Comment 1.247\]](#)

When working with XML documents, it sometimes becomes useful to convert an entire XML document from one set of tags to another. That is one of several services which are available with the *eXtensible Stylesheet Language* (XSL), and the associated XSL transformations. This type of transformation is commonly done between a custom XML syntax and HTML (or another presentation markup language like WML). On the desktop, this is done with the various classes of the `System.Xml.Xsl` namespace. The Compact Framework does not support XSL transformations. [\[Comment 1.248\]](#)

The Compact Framework provides a basic set of XML services, which will meet the needs of most programmers. But not all the XML classes found on the desktop are present in the Compact Framework. As elsewhere, the effects of the 'small is good' design goal for Windows CE are felt everywhere, and omissions from the desktop framework are because the increased code size were deemed too much compared to the anticipated need for a given feature. [\[Comment 1.249\]](#)

The Windows CE XML parser provides services not supported by the Compact Framework. The XML parser under Windows CE is a componentized version Microsoft's MSXML parser, version 3.0 SP1. This parser exposes its services as a set of COM interfaces, which means you must write a Win32 wrapper library to access the unmanaged XML parser. The unmanaged XML parser supports the following features that the Compact Framework XML classes do not: [\[Comment 1.250\]](#)

- DTD Validation
- Schema Validation

- XPath queries
- XSL Transformations

Whether these feature are available to you depends on several factors. First, whether your platform has been built to include the native XML parser, and whether the specific configuration has the feature(s) that you need. Second, is whether you are willing to write the required C++ managed wrappers to work with your XML in native code. [\[Comment 1.251\]](#)

Compact Framework Limitations

As has been said elsewhere, the Compact Framework is a smaller, simpler subset of the desktop .NET Framework. This description of missing features is by no means exhaustive, but is instead meant to discuss some of the more commonly used features by desktop .NET programmers. In some cases, you can find support for missing features in the unmanaged Win32 libraries. When that is the case, a P/Invoke declaration may be all that you need to access one or another feature. In other cases, the lack of a feature may require you to rethink how to implement a given feature. [\[Comment 1.252\]](#)

The Compact Framework does not support pre-compiling IL modules, and all IL-to-native conversion occurs at runtime as JITted code. Programmers familiar with the desktop framework might be familiar with a utility called `ngen.exe`, used to pre-compile modules and to manage the set of existing pre-compiled native images. Since the Compact Framework does not support pre-compiling, there is no equivalent to `ngen.exe` in the Compact Framework. [\[Comment 1.253\]](#)

XML Web services, which the Compact Framework supports, are not the only kind of remote procedure call mechanism in .NET. Another mechanism, .NET Remoting, is also supported on the desktop. .NET Remoting provides a lighter-weight mechanism than Web services, as it is largely driven by a low-level binary protocol. .NET Remoting is also more flexible than Web services, since an object can run locally on the same computer or remotely on a networked computer. It is not an industry-standard protocol, however, and because small is good in Windows CE, .NET Remoting is not supported in the Compact Framework. [\[Comment 1.254\]](#)

The underlying Windows CE libraries primarily³¹ support the Unicode character set. The following table summarizes how various Microsoft operating systems support different character sets: [\[Comment 1.255\]](#)

Operating System	ANSI support (Single/Multi-Byte characters)	Wide support (Unicode characters)
Windows 98, Me	Yes	No *
Windows NT, Windows 2000, Windows XP, Windows Server 2003	Yes	Yes
Windows CE .NET	No	Yes
		* With the Microsoft Unicode Library (MSUL), Unicode support is available.

The computer world is slowly making a transition to using Unicode characters. Windows CE supports only Unicode, but this only means that Win32 functions on Windows CE only accept Unicode characters. This is probably only going to be an issue when reading files which are not Unicode, or communicating with other computers with non-Unicode characters. (Most web sites, for example, do not send Unicode characters.) In that case, you may need to convert between

³¹ We say "primarily" because a few of the network-oriented libraries, including the socket libraries and `wininet.dll`, are bimodal in supporting both ANSI and Unicode character sets.

Unicode and other character sets. For such cases, the encoder and decoder classes from the `System.Text` class can provide you with conversion support. [\[Comment 1.256\]](#)

The system registry is as important in Windows CE as on the desktop. And perhaps as part of trying to de-emphasize the registry, the Compact Framework does not support the framework classes for accessing the registry. The `Registry` and `RegistryKey` classes are not supported by the Compact Framework. But because there are probably some useful things that you can do with the registry, we cover the subject of creating P/Invoke wrappers for the registry in chapter 11 (*Storage*). [\[Comment 1.257\]](#)

Conclusion

Like other core technologies, like Win32 and COM, Microsoft has ported the .NET Framework to Windows CE in the form of the .NET Compact Framework. The core features of both frameworks are identical: a common set of types that support interoperability between different computer languages, executable files with IL instructions which are JITted to native instructions at run-time, and automatic memory reclamation by the runtime. [\[Comment 1.258\]](#)

On to that core, the Compact Framework supports a reduced subset of the namespaces found on the desktop; with an emphasis on building Windows Forms applications that store data using ADO.NET features, and which communicate with the outside world using XML Web services. The consistency between the two implementations is so strictly followed that Compact Framework programs can run unchanged on the desktop under the .NET Framework 1.1. [\[Comment 1.259\]](#)

In the next chapter, we begin looking at the code that goes into a Compact Framework program. [\[Comment 1.260\]](#)