Microsoft's Compact Framework Targets Smart Devices

Alexander Wolfe, Science Writer

Welcome to my first installment of *ACM Queue's* Toolkit column. Each issue I'll dig beneath the market-friendly, feature-rich exterior of some of the best-known (and some of the least-known) development tools in an attempt to separate the core app from the product spec sheet.

This month we're embedding ourselves inside Microsoft's .NET Compact Framework (CF), billed by Microsoft as the perfect platform to create software applications that target mobile devices. That's accurate—and it's not.

Released in March 2003, .NET CF has quickly become the de facto platform for Pocket PCs, and by year's end will be for Smartphones. (Here we're using *Pocket PC* and *Smartphone* in the Microsoft-specific sense—branded platforms generally powered by the code base surrounding Windows CE—and not in the generic personal digital assistant and “two word” smart phone sense.)

That's pretty straightforward. But the putative CF developer soon finds there's a confusing collection of Microsoft acronyms and product names that must be absorbed before you can proceed to actual software development with anything approaching confidence.

For example, take the CF moniker itself. In Microsoft circles, it's often tossed around as a synonym for an integrated development environment (IDE), to distinguish it from the full .NET Framework, which maps to desktop apps. Precisely speaking, the .NET CF isn't even a tool. Rather, it's a common language runtime and a set of class libraries tuned for smart devices. It's indeed a platform since it resides on the target device in a memory footprint of 1.5 MB of ROM.

The *tool* used to develop applications that run on .NET CF-equipped devices is Microsoft's Visual Studio .NET IDE. (The very same Visual Studio IDE is used to build apps for both the full framework and for the CF, which is one reason it's hard to tell Microsoft's tools from its platforms—and from each other—without a software scorecard.)

To help me out, I called Jonathan Wells, Microsoft's man in charge of CF. “It is confusing, to start out with, picking the right tools,” admits Wells. “But once developers enter this space, it becomes less of an issue. What I'm

Queue welcomes

TOOKKIT COLUMNIST

ALEXANDER WOLFE.

UP THIS MONTH:

MICROSOFT'S .NET

COMPACT FRAMEWORK.

finding out in the market is, people will gravitate toward the CF and Visual Studio .NET, which you can think of as one thing.” He proceeded to clarify the framework.

Among other data that developers must grapple with are numerous qualifications regarding target platforms supported by CF. For example, embedded applications targeting the latest releases of Windows CE—versions 4.1 and 4.2—are supported. And CF itself is built into the latest Pocket PCs equipped with Windows Mobile Software for Pocket PC 2003 (a new naming convention Microsoft began using with the Pocket PC systems-software package beginning with the 2003 release). Since CF hit the streets only in 2003, earlier Pocket PC 2000 and Pocket PC 2002 devices don't have the CF runtime. It must be added to RAM to enable CF-based apps to run. Moreover, Smartphone 2002 doesn't support the .NET CF at all, though Smartphone 2003 will.

LONG TIME COMING

For perspective, it's a good idea to understand the position of Microsoft's tools within the overall embedded arena. A big reason why .NET CF has become so high profile is because it, along with the CE 4.1 and 4.2 operating systems, marks the culmination of a decade-long effort by Microsoft to be taken seriously in the embedded arena.

Starting with the release of Modular Windows in 1991, Microsoft made several false starts toward entering the embedded market, each time slinking back to Redmond to lick its wounds after being rejected by the majority of experienced embedded developers.

But after Windows CE 3.0 hit the streets in 1996, Microsoft began to gain traction. Momentum grew with the introduction of the vastly improved Windows CE .NET (aka 4.1) in early 2001.

Not to be underestimated in this success is the graduation from college around that time of the first generation of software developers brought up pretty much exclusively on Microsoft products. As newly minted professionals,

these developers took their predilection for Windows into the business world, further driving adoption.

Managers in our new millennium are receptive for another reason: Microsoft has redefined the price that embedded vendors can charge for their tools. As one of the first to offer a full-blown embedded development environment for under \$1,000, Microsoft in one fell swoop pulled the rug from under traditional embedded houses that were accustomed to charging \$20,000 per seat.

Those old-line vendors—companies such as Wind River, QNX, Lynx (now Lynuxworks), Microtech Research (since acquired by Mentor), and Integrated Systems (bought in 2000 by Wind River)—needed these high price lines to fund the creation of their tools. In contrast, Microsoft's embedded-tools efforts were essentially subsidized outgrowths of the desktop work that's the main focus of the company. There's an interesting history here: The subsidization analogy is particularly true when it comes to the Visual Studio .NET IDE, whose prime target is desktop developers. When it comes to the embedded Windows CE itself, it was only when Microsoft finally cut its ties to desktop Windows and developed a completely new code base for CE 4.1—a process that took nearly five years—that the OS gained real credibility.

With regard to programming languages, implicit in Microsoft's tools story is its strategic decision to encourage developers to move toward its homegrown C# language. Microsoft is positioning C# as both easier to use and better tuned for embedded devices than the more established C++ language.

C# IMPERATIVE

Developers using Visual Studio .NET as their IDE for building CF applications have a choice of C# or Visual Basic. Those who wish to stick with C++, however, must use Microsoft's embedded Visual C++ tool. The latter, though powerful, requires more developer intervention to create a fully formed CF application than does Visual Studio .NET.

A developer could also take a hybrid approach, basing the bulk of one's work in C#, but bouncing over to C++ when higher-performance code is required.

What does C++ have to do with high performance? That's probably a question that only an old-timer would ask, since in decades past assembly language was used when fast code was needed, while C++ was considered the slow way to go. Well, Visual Studio .NET automatically handles memory allocation and deallocation issues, which is done manually in C++. This adds a lot of overhead to the C# code.

Thus, when near-realtime considerations rear their head and a developer needs to get into the guts of the application and optimize some of its code, Visual Studio .NET allows a developer to do that by inserting hand-tuned C++ code (developed with embedded Visual C++) and then bounce back to C# once things slow back beyond the millisecond-response realm.

Of course, as Wells reminded me, a Pocket PC doesn't often require realtime execution. But if you're dealing with embedded avionics equipment or machinery control, the realtime requirement comes into play. Windows CE doesn't typically see service in avionics, though it is used on the factory floor. In the PDA space, my own informal survey indicates that developers of business apps will go with the straight CF approach, while games developers will stick with C++.

WORKADAY ENVIRONMENT

In operation, Visual Studio .NET and CF work together in fairly standard fashion. Visual Studio.NET, used to create the CF applications, operates in a typical drag-and-drop fashion. The IDE is also fitted with network APIs and support for Web services.

CF provides the managed execution environment, which guards against memory leakage. Also included is a just-in-time compiler, which converts the compiled CF application into native code for the target device.

When an application is ready for testing, the IDE can either run an emulator or deploy the executable to the target device. In the former case, an x86 virtual machine runs under Windows CE. In the latter, code is downloaded to a Pocket PC or other hardware device via an ActiveSync cradle connected to the computer.

CF SAMPLE

To get an idea of the code that can be created for CF, I hooked up with Peter Foot, a Microsoft "Windows Embedded Most Valuable Professional" (a distinction the company bestows on cutting-edge developers) and owner of InTheHand, a software consultancy in the United Kingdom.

Foot has put together a range of data-oriented CF components in his work developing a multimedia "field guide" software (birdwatchers, take note!) for Pocket PCs (see figure 1).

OPEN SOURCE CODE FOR MICROSOFT?

You can sample many CF components at Open NETCF (<http://www.opennetcf.org/>), an open-source project that has been billed by its founders as "an independent

source for Compact Framework development information working under the spirit of the open-source movement.”

Source and binaries posted at the Open NETCF site include a Registry library module, which provides read/write access to the system registry from .NET CF code. Another chunk of code, dubbed the desktop communication library, contains classes used for communicating between a PC and a Windows CE device via Microsoft’s Remote API.

A Windows.Forms library available on Open NETCF contains replacement controls (buttons, clipboards, and HTML viewer) for existing Compact Framework controls, as well as new controls.

As this issue of *Queue* went to press, Microsoft released its software development kit for Windows Mobile 2003 Smartphone. The SDK enables developers to build applications for Smartphones leveraging the .NET Compact Framework. The free download is available from the

```
[C#]
using System;
using System.Runtime.InteropServices;

namespace InTheHand
{
    public class Sound
    {
        public static void Play(String FileName)
        {
            int hResult = PlaySoundW(FileName,0,0);

            //playsound
            [DllImport("coredll.dll", EntryPoint="PlaySoundW")]
            private extern static int PlaySoundW(String lpszName, int hModule, int dwFlags);
        }
    }
}
```

This snippet plays a .wav sound file.

Microsoft Web site (<http://www.msdn.microsoft.com/library/default.asp?url=/downloads/list/mobilesmart.asp>)

ALEXANDER WOLFE received his electrical engineering degree from Cooper Union in New York City. A science writer based in Forest Hills, New York, he has contributed to *IEEE Spectrum*, *EE Times*, *Embedded Systems Programming*, and *Byte.com*.

© 2003 ACM 1542-7730/03/1000 \$5.00.

RESOURCES

Microsoft Smart Devices Developer Community
<http://smartdevices.microsoftdev.com>

.NET Compact Framework 1.0 SP1 redistributable download
<http://microsoft.com/downloads/>

Fergus, D., and Roof, L. *The Definitive Guide to the .NET Compact Framework*. Apress, California, 2003.
<http://www.apress.com/>

.NET CF code samples from Peter Foot’s InTheHand support forums
<http://www.inthehand.com>

OpenNETCF resource site
<http://www.opennetcf.org/>