Foreword

WINDOWS APPLICATION DEVELOPMENT has changed substantially since Windows 1.0 was introduced in 1983. Both the way Windows programmers write software and the architecture of the software they write have changed dramatically. The most recent step in this continuous evolution involves the Microsoft .NET Framework. This new platform influences both the developer's tools and their very definition of "application."

The .NET Framework, its compliant languages, and the tools that support them, let developers trade a bit of performance and some control for improvements in developer productivity, code safety, and overall robustness of the completed application. For many developers, this tradeoff is extremely exciting, as they'll be able to get their work done faster without compromising quality. In fact, some developers find that the quality of their code increases substantially when using languages like C# or VB.NET because the languages themselves are inherently an improvement over older offerings.

In the wake of the release of these new managed toolsets, a common misconception in the software development community is that applications written with the .NET Framework are designed only to be written to be "Web apps"—applications which really live on a central Web server, but show their user interface through a Web browser like Internet Explorer. Web-based applications are very appealing for some solutions. Because some Web browser is probably installed on every machine in the world, no distribution of the Web application is necessary. As long as users know

how to reach the application's server and interact with it, they can successfully use it. Updates to the application are done on the server where the application is actually running and don't need to involve updating every client which has a local copy of the software. No local copy exists!

But not all applications written with the .NET Framework must be Web applications. The .NET Framework provides a set of classes (known collectively as "Windows Forms") that are designed to implement "smart client" applications. Smart clients are applications that provide a local user interface directly using the Windows user interface features, rather than using HTML as a presentation layer. In fact, it's very common for developers to use Windows Forms to write stand-alone client applications which provide very rich user interfaces, work offline, and still let developers reap the benefits of the .NET Framework.

Smart client applications have several benefits over Web-oriented applications. Being able to get something done while offline is a very important feature, and will remain so until high-bandwidth connections are available everywhere people want to get work done, from airplanes to living rooms. Because their code executes locally, no round-trip must go over the network to the server before the smart client application can respond to the user. That missing round-trip renders smart client applications impervious to network latency as well.

Because of its intimate relationship with the machine where it is running, a smart client application is generally able to provide a much richer user experience. Custom drawing, interesting font settings, and convenient controls are some of the visual features which help set apart the smart client application from a Web-based application. Smart client applications also, by their nature, are able to use all of the resources available on the client computer. Local disk and memory storage is easy to access.

Problems which can benefit from those strengths are great candidates for solutions involving a smart client. Opportunities for those solutions have always been around: For nearly a decade, MFC made it possible for C++ programmers to write client-side applications with very rich user interfaces. MFC's goal was to provide an object-oriented wrapper that added value to C++ programmers. Sometimes, that value was only to make particular APIs more convenient from C++, but the bulk of MFC was

aimed at offering a framework that made rich client applications easy to write by integrating features most commonly found in such applications.

Windows Forms developers enjoy a more complete set of lightweight wrappers for the system APIs than MFC developers did. The extensive coverage of the .NET Framework Class Library is born out of necessity as, unlike MFC developers, managed programmers have a few challenges in making direct calls to the raw system APIs.

In this book, Chris Sells discusses how the Windows Forms classes and their supporting infrastructure can be used to write robust and rich smart client applications. If you're an experienced Windows programmer who has previously used MFC or directly utilized the Win32 API to write such applications, you will find Chris's direct delivery very appropriate for transferring your knowledge to the managed classes. If you're a developer with less Windows development experience, you'll find the treatment of core concepts in application UI programming indispensable.

A new language, and a new framework for writing smart client applications, offers a new set of compromises to software engineers. What is software engineering besides choosing a solution which brings an acceptable set of compromises to its users? The more tools a developer has and the better he or she is at applying them appropriately, the more problems that developer will be able to solve. (Best of all, a new set of tools for client applications will give Chris new focus, and he'll quit barraging me with complaints about MFC.) Read on to add Windows Forms to your toolbox.

Mike Blaszczak
Priary MFC Developer
Microsoft Corporation
mikeblas@msn.com