

Microsoft™ .NET Framework:

The Common Language Runtime and Framework Class Library

January 2005



Copyright

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED, OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

© 2005 Microsoft Corporation. All rights reserved.

Microsoft, Active Directory, BizTalk, InfoPath, JScript, Outlook, SharePoint, Visio, Visual Basic, Visual Studio, and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

The Microsoft Enterprise Development Strategy Series

The Enterprise Development Strategy Series of briefing papers from Microsoft are designed to deliver overview and technical information of Microsoft application development technologies to developers, architects and IT management. The papers in this occasional series describe strategic goals Microsoft is addressing and summary technical information on these technologies and how they address the needs of the enterprise.

Feedback is important to us. If you have comments on the series, please email that information with a subject line of "Feedback" to entdevfb@microsoft.com.

Contents

Overview	2
Customer Problems.....	3
Security.....	3
Interoperability with Existing Systems	3
Versioning.....	3
Deployment	4
Different Techniques for Different Types of Applications.....	4
Reducing Software Defects.....	4
Supporting Multiple Languages.....	5
Localized Applications.....	5
Scalability	5
Design Goals for the .NET Framework.....	6
How Today's Technology Solves those Problems.....	7
Security.....	8
User vs. Code Security.....	9
Interoperability	9
Role-Based Security.....	9
Code Access Security	10
Versioning.....	10
Deployment	11
Similar Techniques for Different Types of Applications	12
Reducing Software Defects.....	12
Supporting Multiple Programming Languages	13
Localized Applications.....	14
Scalability	15
.NET Framework Class Library Namespaces	15
System.....	16
System.CodeDom	16
System.Collections.....	16
System.ComponentModel.....	16
System.Configuration	16
System.Data	17
System.Diagnostics.....	17
System.DirectoryServices	17
System.Drawing	17
System.EnterpriseServices	17
System.Globalizatiion.....	18
System.IO.....	18
System.Management	18
System.Messaging	18
System.Net.....	18
System.Reflection.....	19
System.Resources	19
System.Security.....	19

System.Text.....	19
System.Threading	19
System.Timers.....	19
System.Web	19
System.Windows.Forms.....	20
System.Xml	20
Future Evolution	21
64-bit Support.....	21
Generics	21
Nullable types	21
Lightweight transactions.....	22
Cache invalidation	22
Edit and Continue.....	22
ClickOnce	22
Appendix: Performance Evidence and Quotes	24
In the Industry.....	25
Agile Architecture	26
In the Industry.....	26
References.....	27

Overview

The .NET Framework from Microsoft provides services to application developers that are necessary to quickly create scalable solutions that meet stringent requirements for security, manageability and productivity. This whitepaper introduces the guiding principles and thoughts behind the .NET Framework, the core features of the Common Language Runtime and its supporting Framework Base Class Libraries and how it is evolving in the next major version.

The Microsoft Common Language Runtime (CLR) and the .NET Framework class libraries – collectively called the .NET Framework – were designed to enable developers to easily create scalable, secure, interoperable and manageable applications that can also leverage existing investments in other technologies and platforms. The .NET infrastructure works with non-Microsoft technologies through its built-in support for creating and consuming Web services, and it works with existing Microsoft technologies by providing native integration with COM components. These features help to extract further value from an organization's existing investments in prior technologies and ensure interoperability with other platforms.

Since the earliest days of software development, organizations have sought to build applications quickly and with higher quality – that is fewer bugs. Applications should also interoperate with different environments and technologies and should be built with widely-accepted languages that are easy to learn and maintain, simultaneously boosting productivity. Many approaches have appeared to try to improve the development process, but often these efforts have been focused on the analysis and design phase. It is not until more recently that the underlying architecture of the development platform itself has achieved a prominent role in laying the groundwork for building enterprise applications.

Whether an enterprise is concerned with interoperability with Web services or COM applications, building solutions more quickly and with fewer bugs, or using a common programming model across application types and languages, the .NET Framework provides organizations the tools to meet all these goals and more.

Customer Problems

There are a number of commonly recurring challenges that appear during the creation of enterprise applications. The .NET Framework addresses these issues by providing built-in support to addresses these issues.

Security

Security has moved to the forefront of the challenges faced by IT, and it takes on many forms. Companies seek to develop applications free of buffer overruns, the number one cause of security holes and other errors that can be maliciously exploited. Corporations are also seeking to eliminate the problems caused by downloaded code being able to run in the security context of the person using the machine; instead, they want that downloaded code to be far more restricted in the privileges it has on a computer. Companies seek a way to easily encrypt and decrypt data not only when it flows across a network, but when it is stored locally or in a database. Finally, companies seek to ensure that security flows from the Web site or the desktop through all the layers of business logic all the way to the database, and that this is handled efficiently and is easy to develop and track once it is in place. These are themes that are addressed by the CLR, which has a strong security model built directly into its core design.

Interoperability with Existing Systems

When evaluating any technology, one of the key points is how easily it integrates with existing systems. No organization wants to abandon systems that have evolved over the years, nor do they want to implement a platform that cannot work with other platforms. Even when companies attempt to narrow all their systems to the platform of a single vendor, working with external partners can lead to the need to interoperate with a wide variety of technologies. Any platform for enterprise systems should offer not just a rich set of tools for its own technology, but should provide a wealth of options for interoperating with existing technologies from various vendors. This interoperability must be based on industry standards in order to ensure the highest level of compatibility. As is shown, .NET provides a wide array of options for interoperating with other platforms.

Versioning

Businesses requirements and the solutions they drive change faster than ever, and this means that applications can be updated frequently. However, changing application components in the past often meant breaking numerous dependant applications, even when those applications did not need the changed functionality. This meant that every change, no matter how small, had to be tested against numerous client applications, slowing down deployment and often requiring updates in numerous places.

Having many different versions of particular components often led to a state known as “DLL Hell”, where client applications needed a component but would only work with a specific

version. Newer versions were automatically called even if older versions still physically existed, and client applications could break. .NET eliminates DLL Hell by providing advanced techniques for locating assemblies and allowing side-by-side instancing of multiple versions.

Deployment

Deploying enterprise applications can be a difficult process, as applications are typically written in multiple layers that can be independently distributed over a variety of physical hardware devices. Deployment may be complicated by the versioning issues previously discussed. Additionally, some technologies, including COM-based Web applications, lock components when they are used and keep them locked until the Web server is stopped. Add to this that most software requires some sort of registration or configuration, and the deployment process a complicated phase of the application lifecycle.

Easing deployment issues is a critical requirement for IT departments as they wrestle with a growing number of components and an increasing number of hardware devices. In addition, the ability to update components of applications without having to stop those applications or interrupt service is needed in today's high-availability scenarios. The .NET Framework addresses these deployment scenarios and allows companies to simply copy files without the need for registration, apply specific application policies, in addition to providing new techniques for deploying applications and components.

Different Techniques for Different Types of Applications

One of the challenges when building applications over the past ten years has been that applications requiring a rich user interface were coded using an event-driven model while Web applications were coded to account for the stateless nature of HTTP. This led to different programming paradigms and tools for interactive Windows clients versus Web or business tier applications. Couple this disconnect with yet another model needed for writing applications for smart devices such as PDAs and cell phones and there existed a mix of tools, development approaches, and languages in order to create applications across a businesses' spectrum of users. .NET allows companies to use the same event-driven programming model on both rich-client and Web applications and to use the same tools and languages to target Windows, Web, and device clients.

Reducing Software Defects

Defects, more affectionately known as bugs, cost companies money, both in an effort to catch them before products are released and in patching applications after deployment. One common defect is a memory leak caused by the failure to remove object references and memory after they are no longer needed. Other defects are caused by components written in multiple languages handling exceptions differently, or passing similar data types differently. The CLR manages all of a programs' memory for the developer, immediately eliminating a major cause of defects. The CLR also ensures that data types are the same across all .NET languages.

Supporting Multiple Languages

Most organizations support a variety of computer languages today. Many organizations have COBOL and RPG programmers working on mainframe and midrange computers, while developers on desktop and Web applications work in languages such as C, C++, Visual Basic, ASP, Java and more. Most languages use completely different tools in order to develop software. The CLR is language-neutral, which means that any language can compile code targeted to the CLR, and a development organization can use a unified toolset to help create, test and debug applications.

Localized Applications

Business is global in reach. Building applications that support multiple human cultures and languages has typically been a substantial challenge, and it is not unusual for departments to maintain separate source code for each localized version of an application. Maintaining a separate code base for each country can lead to versioning and update inconsistencies, as well as defects in implementation. .NET greatly simplifies the creation of localized applications using a single code base.

Scalability

Due to the global adoption of the Web, the number of customers simultaneously accessing a Web application can number in the hundreds of thousands or even millions. Building applications that can scale properly to a large number of users has always been challenging; throwing hardware at the problem is only part of the solution. Applications must be designed to use resources efficiently and sparingly, such as minimizing the load on database servers. .NET has been architected with scalability in mind and contains both technologies and detailed architecture guides that assist organizations in creating applications that will meet the demands of the largest workloads.

Design Goals for the .NET Framework

The .NET Framework is a Microsoft Windows component that supports building and running the next generation of software applications—both client- and server-side—and Web services. It is based on industry standards and manages provides services involved in developing and executing a piece of software, enabling developers to focus on writing the core business logic code. The .NET Framework is composed of two key parts: the Common Language Runtime (CLR) and the associate base class libraries. The .NET Framework provides:

- A consistent, language-neutral, object-oriented programming environment.
- A code-execution environment that minimizes software deployment and versioning conflicts, security-policy based execution of code, eliminates the performance problems of scripted or interpreted environments.
- A consistent developer experience across widely varying types of applications—including smart client applications and Web-based applications and services, all running on Windows.
- A consistent model for accessing system functionality through a well factored and powerful collection of utility classes, organized into the Framework Class Libraries.
- Built-in support for a wide variety of standards, ensuring interoperability with a wide range of technologies and platforms.

In .NET, application services are offered via a common object-oriented programming model, including where OS facilities are accessed via DLL functions or via COM objects. .NET also seeks to greatly simplify the plumbing required for both Windows and Web applications. Many of the more arcane details about protocols and internal structures are hidden (although they can be accessed if necessary.) The extensibility features of the .NET Framework allow developers to use existing functionality in DLLs and COM objects as necessary. They appear to the developer as if they were native code through code wrapping.

Once written and built, a managed .NET application can execute on any platform that supports the CLR, such as Windows 98, Windows 2000, Windows XP, Windows 2003, and may even execute on non-Microsoft operating systems that support their own implementation of the ISO/IEC 23271 standards.

The runtime automatically handles object layout and manages references to objects, releasing them when they are no longer being used. Objects whose lifetimes are managed in this way are considered “managed” by the runtime. Garbage collection eliminates memory leaks when using managed code and objects as well as some other common programming errors. If code is managed, developers can use managed data, unmanaged data, or both managed and unmanaged data in a .NET Framework application.

The CLR makes it easy to design components and applications whose objects interact across languages. Objects written in different languages can communicate with each other, and their behaviors can be tightly integrated. .NET supports over 20 languages; Microsoft provides

Visual C#, Visual Basic .NET, Visual C++ with Managed Extensions, and Visual J#. Third party languages include COBOL, RPG, Python, Perl, and more.

From a security standpoint, the CLR verifies that the executable file is valid and that addresses do not point outside of the file. This helps provide assembly isolation. The code is verified for type safety at compile time. This is a major plus from a security perspective because the verification process can prevent bad pointer manipulation, validate type conversions, check array bounds, and so on. This virtually eliminates buffer overflow vulnerabilities in managed code. Finally, the virtual execution environment provided by the Common Language Runtime allows additional security checks to be performed at runtime. Specifically, code access security can make various run-time security decisions based on evidence such as the identity of the calling code.

Scalability is provided by a variety of mechanisms, including enhancements to the handling of state in Web applications to the native, default support for disconnected caches of data in a relational format. Applications written on the .NET Framework can automatically benefit from multiple processors, and writing multithreaded applications is greatly simplified. Applications can also interact with Microsoft Message Queue in order to offload functionality that can be performed offline.

How Today's Technology Solves those Problems

The CLR manages code execution for the developer, reducing many of the problems that led to bugs in unmanaged applications: memory management is handled automatically; buffer overruns are eliminated; and cross-language integration is greatly simplified. The CLR allows developers to use a wide variety of languages, with over twenty currently available. The CLR ensures that the data types are consistent from one .NET language to another, and exceptions in one .NET language are identical to exceptions in any other .NET language. Support for multiple languages is key for a number of important reasons. For instance it allows for component developers and consumers to interoperate without the need for tightly specifying the consuming language.

By building in support for Web services, Microsoft has provided a clear path for interoperating with most other platforms. Microsoft implements the industry standards surrounding Web services: SOAP, XML, WSDL and UDDI, .NET can call Web services on other platforms as well as publish its Web services to applications on other platforms. Data, even structured as a relational schema, can be passed to Web services running on any platform with ease. This type of interoperability is made simple thanks to the compliance with industry standards implemented in .NET.

While .NET allows developers to create Windows applications, Web applications, Web services, Windows services, and other project types, the programming model is the same for all types of applications thanks to the .NET Framework Class Library. This library provides a base set of classes available to all types of applications regardless of the language used. This means that developers can learn one set of classes and use them across all applications and languages, greatly reducing the time to learn new languages or new types of applications.

The CLR and the .NET Framework support a rich set of APIs supporting development tools such as debuggers, tracing tools and editors. Thanks to the design of the CLR, developers

can actually create any application type that can be built using .NET with a simple text editor such as Notepad. Microsoft used this infrastructure to allow tools like Visual Studio .NET, an IDE designed to provide unparalleled power and productivity to developers. In addition a number of third party tools are available from a variety of sources such as Borland targeting .NET.

The CLR is language-neutral, which means that it can run code written in C#, Visual Basic .NET, Visual C++, Visual J#, Net COBOL, RPG.NET, or any other language, as long as that language is first compiled to the bytestream format the CLR expects. When compiling to managed code, the language-specific compiler translates the source code into Microsoft intermediate language (MSIL), which is a CPU-independent set of instructions that can be efficiently converted to native code. MSIL includes instructions for loading, storing, initializing, and calling methods on objects, as well as instructions for arithmetic and logical operations, control flow, direct memory access, exception handling, and other operations. The compiled MSIL is placed into an EXE or DLL file, called a Portable Executable (PE) file. Before code can be run, the MSIL in the PE must be converted to CPU-specific code, usually by a just-in-time (JIT) compiler. Because the Common Language Runtime supplies one or more JIT compilers for each computer-processor instruction architecture it supports, the same set of MSIL can be JIT-compiled and run on any supported architecture. During execution, managed code receives services such as garbage collection, security, interoperability with unmanaged code, cross-language debugging support, and enhanced deployment and versioning support from the CLR.

Security

The use of an intermediate language coupled with the run-time environment provided by the Common Language Runtime offers assembly developers immediate security advantages.

- File format and metadata validation. The Common Language Runtime verifies that the PE file format is valid and that addresses do not point outside of the PE file. This helps provide assembly isolation. The Common Language Runtime also validates the integrity of the metadata that is contained in the assembly.
- Code verification. The MSIL code is verified for type safety at JIT compile time. This is a major plus from a security perspective because the verification process can prevent bad pointer manipulation, validate type conversions, check array bounds, and so on. This virtually eliminates buffer overflow vulnerabilities in managed code, although developers still need to carefully inspect any code that calls unmanaged application programming interfaces (APIs) for the possibility of buffer overflow.
- Integrity checking. The integrity of strong named assemblies is verified using a digital signature to ensure that the assembly has not been altered in any way since it was built and signed. This means that attackers cannot alter source code in any way by directly manipulating the MSIL instructions.
- Code access security. The virtual execution environment provided by the Common Language Runtime allows additional security checks to be performed at runtime. Specifically, code access security can make various run-time security decisions based on the identity of the calling code.

User vs. Code Security

User security and code security are two complementary forms of security that are available to .NET Framework applications. User security answers the questions, "Who is the user and what can the user do?" while code security answers the questions "Where is the code from, who wrote the code, and what can the code do?" Code security involves authorizing the application's (not the user's) access to system-level resources, including the file system, registry, network, directory services, and databases. In this case, it does not matter who the end user is, or which user account runs the code, but it does matter what the code is and is not allowed to do. The .NET Framework user security implementation is called role-based security. The code security implementation is called code access security.

Interoperability

When evaluating any new technology, one of the key points is how easily it integrates with existing systems. Microsoft's .NET Framework provides a wide range of interoperability features built in. First, developers have support for creating and consuming Web services. With access to Web services written on any platform, they can continue to use those existing applications and leverage a company's investment in those technologies. A business can operate a mixed platform environment on easily share data across platforms, thanks to .NET's adherence to open standards for Web services, including XML, SOAP, WSDL, and UDDI.

Beyond calling Web services, .NET can continue to use existing COM components. The CLR includes COM interoperability functionality; .NET applications can call COM components as if they were written in .NET, and COM applications can call .NET assemblies as if they were COM components. This dual functionality allows development groups to migrate from COM to .NET gradually, in a staged fashion; they do not have to rewrite an entire application, but can replace components over time – the user interface first and business logic later, for example.

Web services and COM interoperability are not the only ways to interoperate with existing systems. If a company has existing CICS transactions, Microsoft's Host Integration Server allows applications to call CICS transactions as if they were local .NET components. This means that .NET can extend the life of an organization's existing tested and debugged CICS transactions into the future, making them as easy to use as assemblies developed natively in .NET while using them in new styles of applications – Web services for instance.

Role-Based Security

.NET Framework role-based security allows a Web application to make security decisions based on the identity or role membership of the user that interacts with the application. If an application uses Windows authentication, then a role is a Windows group. If an application uses other forms of authentication, then a role is application-defined and user and role details are usually maintained in Microsoft SQL Server or user stores based on Active Directory.

The identity of the authenticated user and its associated role membership is made available to Web applications through Principal objects, which are attached to the current Web request.

Code Access Security

Code access security authorizes code when it attempts to access secured resources, such as the file system, registry, network, other .NET assemblies and so on, or when it attempts to perform other privileged operations, such as calling unmanaged code or using reflection.

Code access security is an important additional defense mechanism that developers can use to provide constraints on a piece of code. An administrator can configure code access security policy to restrict the resource types that code can access and the other privileged operations it can perform. From a Web application standpoint, this means that in the event of a compromised process where an attacker takes control of a Web application process or injects code to run from inside the process, the additional constraints that code access security provides can limit the damage that can be done.

The authentication (identification) of code is based on evidence about the code, for example, its strong name, publisher, or installation directory. Authorization is based on the code access permissions granted to code by security policy.

Versioning

In .NET, applications are compiled into one or more files called assemblies. Each assembly has a version number as part of its identity. Assemblies may also be *strongly named*, which is a mechanism by which each company that produces assemblies have a way to uniquely identify that assembly. It was Microsoft's responsibility to devise some mechanism that allows this, and Microsoft chose to use standard public-key cryptographic techniques.

The version number is stored in the assembly manifest along with other identity information like relationships and identities of other assemblies connected with the application.

- The version number is physically represented as a four part number containing <major version>.<minor version>.<build number>.<revision>.
- Two assemblies that differ only by version number are considered by the runtime to be completely different assemblies.
- The run time distinguishes between regular and strongly named assemblies for the purpose of versioning. Version checking only occurs with strongly named assemblies.
- When an assembly is built, the development tool records dependency information for each assembly that is referenced in the assembly manifest. The run time uses these version numbers in conjunction with configuration information set by the administrator, an application, or a publisher, to load the proper version of a referenced assembly.
- When an application uses strong-named assemblies, it will always use the version number the application was built with unless this is explicitly overridden. This allows side-

by-side execution of different versions of the same applications, thereby preventing the traditional problems encountered when a DLL is replaced with a different version.

The run time uses the following steps to resolve an assembly reference:

1. Determines the correct assembly version by examining applicable configuration files, including the application, publisher policy, and machine policy configuration files. In a Microsoft Internet Explorer Web scenario where the configuration file is located on a remote machine, the run time must locate and download the application configuration file first. This step only occurs for strong-named assemblies. Since simple named assemblies don't undergo version checking, policy files are not checked.
2. Checks whether the assembly name has been bound-to before and, if so, use the previously loaded assembly.
3. Checks the global assembly cache. If the assembly is found there, the run time uses this assembly. This step only occurs for strong-named assemblies.
4. Probes for the assembly using the following steps:
 - a. If configuration and publisher policy do not affect the original reference and if the bind request was created using the `Assembly.LoadFrom` method, the run time checks for location hints.
 - b. If a code base is found in the configuration files, the run time checks only this location. If this probe fails, the run time determines that the binding request failed and no other probing occurs.
 - c. Probes for the assembly using the heuristics described in the .NET CLR documentation. If the assembly is not found after probing, the run time requests the Windows Installer to provide the assembly. This acts as an install-on-demand feature.
 - d. Finally the directory the caller loaded from is used as the last probing location.

Deployment

The .NET Framework introduces several new features aimed at simplifying application deployment and solving DLL Hell. Both end users and developers are familiar with the versioning and deployment issues that can arise with today's component-based systems. For example, virtually every end user has installed a new application on their machine, only to find that an existing application mysteriously stops working. Most developers have also spent time with `regedit`, trying to keep all the necessary registry entries consistent in order to activate a COM class.

Installing an application today is a multi-step process. Typically, installing an application involves copying a number of software components to the disk and making a series of registry entries that describe those components to the system.

The separation between the entries in the registry and the files on disk makes it very difficult to replicate applications and to uninstall them. Also, the relationship between the various

entries required to fully describe a COM class in the registry is very loose. These entries often include entries for `coclasses`, `interfaces`, `typelibs`, and DCOM app IDs, not to mention any entries made to register document extensions or component categories. Oftentimes developers end up keeping these in sync manually.

Finally, this registry footprint is required to activate any COM class. This drastically complicates the process of deploying distributed applications because each client machine must be touched to make the appropriate registry entries.

These problems are primarily caused by the description of a component being kept separate from the component itself. In other words, applications are neither self-describing nor self-contained. Assemblies are the building blocks used by the .NET Framework to solve the versioning and deployment issues. Assemblies are the deployment unit for types and resources. In many ways an assembly equates to a DLL in today's world; in essence, assemblies are a "logical DLLs." Assemblies are self-describing through metadata called a manifest. Just as .NET uses metadata to describe types, it also uses metadata to describe the assemblies that contain the types.

.NET applications can be distributed in a variety of ways, including xcopy, code download, and through the Windows Installer. For many applications, including Web applications and Web Services, deployment is as simple as copying a set of files to disk and running them. Uninstall and replication are just as easy—just delete the files or copy them.

The ease with which applications may be distributed is not limited to desktop applications. Server deployments are greatly simplified, through the ability to upload assemblies and Web components, even while an application is running. Because no resources are locked, files may be uploaded at any time through simple mechanisms, including FTP.

Similar Techniques for Different Types of Applications

.NET removes the need to program different applications using different techniques. For example, development groups can develop Web and Windows applications the same way, thanks to the consistent programming model of .NET: all application services are offered via a common object-oriented programming model, unlike previous environments like VB6 where some OS facilities are accessed via DLL functions and other facilities are accessed via COM objects.

Reducing Software Defects

Automatic memory management is one of the services that the Common Language Runtime provides during managed execution. The Common Language Runtime garbage collector manages the allocation and release of memory for an application. For developers, this means not having to write error prone code to perform memory management tasks when developing managed applications. Automatic memory management can eliminate common problems, such as forgetting to free an object and causing a memory leak, or attempting to access memory for an object that has already been freed.

.NET seeks to greatly simplify the plumbing and arcane constructs required by Win32® and COM. Specifically, developers no longer need to gain an understanding of the registry, GUIDs, IUnknown, AddRef, Release, HRESULTS, and so on. It is important to note that .NET doesn't just abstract these concepts away from the developer; in the new .NET platform, these concepts simply do not exist at all.

Both Web applications and Web services can be debugged as easily as Windows client applications using Visual Studio .NET. The environment is designed to make the Web development experience the same as developing client forms-based applications. For example, starting a Web services project will display a simple test page, through which testers and developers can call the Web services that have been developed. Debugging this code is simple. Developers can set breakpoints in the Web service code and invoke the method through the test page. The breakpoints will be hit as the Web method is invoked.

As mentioned earlier, .NET virtually eliminates buffer overruns in managed code. Additionally, ASP.NET 1.1 eliminated the ability for script code entered into a Web form to then be executed, preventing certain forms of cross-site scripting attacks.

Supporting Multiple Programming Languages

The CLR itself is language-neutral, because it only understands one language: MSIL (Microsoft Intermediate Language.) This means that any language that compiles to MSIL can be used for .NET development. The .NET Platform programming languages — including Visual Basic .NET, Visual C#, Visual J#, Managed Extensions for C++, JScript, and many other programming languages from various vendors — use .NET Framework services and features through a common set of unified classes. The .NET unified classes provide a consistent method of accessing the platform's functionality. If developers learn to use the class library, they will find that all tasks follow the same uniform architecture. They no longer need to learn and master different API architectures to write applications.

The Common Type System (CTS) is a piece of the CLR that defines how types are declared, used, and managed in the runtime, and is also an important part of the runtime's support for cross-language integration. The CTS establishes a framework that helps enable cross-language integration, type safety, and high performance code execution. It provides an object-oriented model that supports the complete implementation of many programming languages and defines rules that languages must follow, which helps ensure that objects written in different languages can interact with each other.

Traditionally, a language's error handling model relied on either the language's unique way of detecting errors and locating handlers for them, or on the error handling mechanism provided by the operating system. The Common Language Runtime greatly assists the design of fault-tolerant software by providing a model for notifying programs of errors in a uniform way. All .NET Framework operations indicate failure by throwing exceptions. The runtime handles exceptions without regard for the language that generates the exception or the language that handles the exception and does not require any particular language syntax for handling exceptions, but allows each language to define its own syntax. Exceptions can be thrown across process and even machine boundaries.

A powerful feature of .NET is its ability to debug across languages that target the Common Language Runtime, and across execution environments. For example, if a development team writes a Visual Basic .NET component that is called by a C# component that is in turn called by COBOL code (that targets the runtime), a debugger could seamlessly step between languages when debugging. The developer is presented with a single call stack that shows the different functions called in the languages just stepped through.

Developers can also step in the debugger between native C++ code to any .NET Framework language and vice versa. Code written for .NET-based applications can work with native C++ code through Platform Invoke, COM Interoperability, or through Managed Extensions for C++. They can seamlessly step into and debug from one language to the other, and again, the debugger will have a single call stack showing all the components. To enable debugging between the Common Language Runtime code and C++ code running natively, developers need to change an option in the launching project settings. In Solution Explorer, locate the startup project (this project's name will be bolded). This is the project that gets launched when a developer presses F5. To debug both runtime code and native code, the startup project has to be configured to support both types of debugging. To do this, open the project properties and select the Debugging folder under Configuration Properties.

Localized Applications

.NET was designed from the start to make developing for an international audience easy by taking advantage of services built into the .NET Framework.

In .NET, there are two parts to creating a world-ready application: globalization, the process of designing applications that can adapt to different cultures, and localization, the process of translating resources for a specific culture. The .NET localization model consists of a main assembly that contains both the application code and the fallback resources — strings, images, and other objects for the language in which the application is originally developed. Each localized application will have satellite assemblies, or assemblies which contain only the localized resources. Because the main assembly always contains the fallback resources, if a resource is not found in the localized satellite assembly, the Resource Manager will attempt to load it in a hierarchical manner, eventually falling back to the resource in the main assembly. This means that a development staff can localize applications for a variety of cultures but if they encounter a culture for which they have not created a localized version, the application will gracefully fall back to the default culture.

In the past, a common way to localize a Web application was to create a copy of each page and localize it. While this works well for static content, a more robust approach is needed for dynamic content. .NET makes this easy by allowing applications to load localized content at runtime while maintaining just a single page. When a localized application executes, its appearance is determined by two culture values. (A culture is a set of user preference information related to the user's language, environment, and cultural conventions.) The UI culture setting determines which resources will be loaded. The UI culture is set as `UICulture` in Web.config files and page directives, and `CurrentUICulture` in Visual Basic or Visual C# code. The culture setting determines formatting of values such as dates, numbers, currency, and so on. The culture is set as `Culture` in Web.config files and page directives, `CurrentCulture` in Visual Basic or Visual C# code.

Scalability

Tools to enhance scalability have been built into the .NET Framework from the beginning. Many of these enhancements center around Web applications, which tend to have the greatest number of users. For example, ASP.NET applications include sophisticated support for tracking user information between multiple requests – session state management. To enable applications to scale to thousands of users, sessions can be stored in different flexible ways. The default method is to store session information in-process. This stores user information in the memory of a single server. The in-process method offers the fastest performance. Two other methods offer slower performance but greater scalability: storing session information on a state server, or storing session information on a computer running SQL Server.

Data access is also an area of intense interest when it comes to performance and scalability. To this end, the most commonly-used object with which to work with data, the DataSet, uses a disconnected paradigm. This allows applications to make the connection to the database as late as possible and release it as soon as possible. In addition, data is cached in memory, meaning that working with data does not require round trips to the database server. This can greatly increase the scalability of all types of .NET applications.

The Middleware Company, founders of the leading J2EE developer forum TheServerSide.com, have conducted a benchmark of the .NET Framework and J2EE, finding the .NET Framework to significantly outperform J2EE for Web application hosting, Web services, and distributed transactions. The .NET Framework also offers significant performance and scalability benefits over the previous Active Server Pages (ASP) technology, thanks to its just-in-time (JIT) compilation and caching technologies.

Visual Studio .NET also ships with Application Center Test (ACT), a tool used to stress test Web servers and analyze performance and scalability problems with Web applications. ACT simulates a large group of users by opening multiple connections to the server and rapidly sending HTTP requests. It also supports several different authentication schemes and the SSL protocol, making it ideal for testing personalized and secure sites.

When applications appear to misbehave, developers can rely on such tools as the CLR Profiler. CLR Profiler enables developers to look at the managed heap of a process and investigate the behavior of the garbage collector. Using the various views in the tool, developers can obtain useful information about the execution, allocation, and memory consumption of an application. CLR Profiler is not a starting point for analyzing problems. Rather, it helps developers identify and isolate problematic code and track down memory leaks. Using CLR Profiler, developers can identify code that allocates too much memory, causes too many garbage collections, and holds on to memory for too long.

.NET Framework Class Library Namespaces

The .NET Framework includes classes, interfaces, and value types that provide access to system functionality and expedite and optimize the development process. To facilitate interoperability between languages, the .NET Framework types are CLS-compliant and can therefore be used from any programming language whose compiler conforms to the common language specification (CLS). .NET Framework types use a dot syntax naming scheme that connotes a hierarchy. This technique groups related types into namespaces so they can be searched and referenced more easily. The first part of the full name — up to the rightmost dot — is the namespace name. The last part of the name is the type name. For example, `System.Collections.ArrayList` represents the `ArrayList` type, which belongs to the `System.Collections` namespace. The types in `System.Collections` can be used to manipulate collections of objects. The top-level namespaces are listed here.

System

The `System` namespace contains fundamental classes and base classes that define commonly-used value and reference data types, events and event handlers, interfaces, attributes, and processing exceptions. Other classes provide services supporting data type conversion, method parameter manipulation, mathematics, remote and local program invocation, application environment management, and supervision of managed and unmanaged applications.

System.CodeDom

The `System.CodeDom` namespace contains classes that can be used to represent the elements and structure of a source code document. The classes in this namespace can be used to model the structure of a source code document that can be output as source code in a supported language using the functionality provided by the `System.CodeDom.Compiler` namespace. The strength here comes from the fact that developers can generate code at runtime, compile it in memory, and execute it. This allows applications to respond to a changing environment in real time.

System.Collections

The `System.Collections` namespace contains interfaces and classes that define various collections of objects, such as lists, queues, bit arrays, hashtables and dictionaries.

System.ComponentModel

The `System.ComponentModel` namespace provides classes that are used to implement the run-time and design-time behavior of components and controls. This namespace includes the base classes and interfaces for implementing attributes and type converters, binding to data sources, and licensing components.

System.Configuration

The System.Configuration namespace provides classes and interfaces that allow applications to programmatically access .NET Framework configuration settings and handle errors in configuration files (.config files). Configuration files are incredibly powerful because they allow administrators to change values in applications without having to recompile those applications.

System.Data

The System.Data namespace consists mostly of the classes that constitute the ADO.NET architecture. The ADO.NET architecture enables developers to build components that efficiently manage data from multiple data sources. In a disconnected scenario (such as the Internet), ADO.NET provides the tools to request, update, and reconcile data in multiple tier systems. The ADO.NET architecture is also implemented in client applications, such as Windows Forms, or HTML pages created by ASP.NET.

System.Diagnostics

The System.Diagnostics namespace provides classes that allow applications to interact with system processes, event logs, and performance counters.

System.DirectoryServices

The System.DirectoryServices namespace provides easy access to Active Directory from managed code. The namespace contains two component classes, DirectoryEntry and DirectorySearcher, which use the Active Directory Services Interfaces (ADSI) technology. ADSI is the set of interfaces that Microsoft provides as a flexible tool for working with a variety of network providers. ADSI gives the administrator the ability to locate and manage resources on a network with relative ease, regardless of the network's size.

The classes in this namespace can be used with any of the Active Directory service providers. The current providers are: Internet Information Services (IIS), Lightweight Directory Access Protocol (LDAP), Novell NetWare Directory Service (NDS), and WinNT.

System.Drawing

The System.Drawing namespace provides access to GDI+ basic graphics functionality. GDI+ is a subsystem of Windows that allows drawing to the screen or to printers, and provides developers with advanced functionality for rendering user interfaces and data.

System.EnterpriseServices

The `System.EnterpriseServices` namespace provides an important infrastructure for enterprise applications. `System.EnterpriseServices` provides access to the COM+ services, which provide a services architecture for component programming models deployed in an enterprise environment. These services include transactions distributed across components and across databases, as well as management of object pools for scalability. This namespace provides .NET objects with access to COM+ services making the .NET Framework objects more practical for enterprise applications.

System.Globalization

The `System.Globalization` namespace contains classes that define culture-related information, including the language, the country/region, the calendars in use, the format patterns for dates, currency, and numbers, and the sort order for strings. These classes are useful for writing globalized (internationalized) applications. Classes like `StringInfo` and `TextInfo` provide advanced globalization functionalities, such as surrogate support and text element processing.

System.IO

The `System.IO` namespace contains types that allow reading and writing to files and data streams, and types that provide basic file and directory support.

System.Management

Provides access to a rich set of management information and management events about the system, devices, and applications instrumented to the Windows Management Instrumentation (WMI) infrastructure. Applications and services can query for interesting management information (such as how much free space is left on the disk, what is the current CPU utilization, which database a certain application is connected to, and much more.)

System.Messaging

The `System.Messaging` namespace provides classes that allow applications to connect to, monitor, and administer message queues on the network and send, receive, or peek messages.

System.Net

The `System.Net` namespace provides a simple programming interface for many of the protocols used on networks today. The `WebRequest` and `WebResponse` classes form the basis of what are called pluggable protocols, an implementation of network services that enables developers to create applications that use Internet resources without worrying about the specific details of the individual protocols.

System.Reflection

The System.Reflection namespace contains classes and interfaces that provide a managed view of loaded types, methods, and fields, with the ability to dynamically create and invoke types.

System.Resources

The System.Resources namespace provides classes and interfaces that allow developers to create, store, and manage various culture-specific resources used in an application.

System.Security

The System.Security namespace provides the underlying structure of the Common Language Runtime security system, including base classes for permissions.

System.Text

The System.Text namespace contains classes representing ASCII, Unicode, UTF-7, and UTF-8 character encodings; abstract base classes for converting blocks of characters to and from blocks of bytes; and a helper class that manipulates and formats String objects without creating intermediate instances of String.

System.Threading

The System.Threading namespace provides classes and interfaces that enable multithreaded programming. In addition to classes for synchronizing thread activities and access to data (Mutex, Monitor, Interlocked, AutoResetEvent, and so on), this namespace includes a ThreadPool class that allows applications to use a pool of system-supplied threads, and a Timer class that executes callback methods on thread pool threads.

System.Timers

The System.Timers namespace provides the Timer component, which allows developers to raise an event on a specified interval. The Timer component is a server-based timer, which is designed for use with worker threads in a multithreaded environment. Server timers can move among threads, resulting in more accuracy than Windows timers in raising the event on time.

System.Web

The System.Web namespace supplies classes and interfaces that enable browser-server communication.

System.Windows.Forms

The System.Windows.Forms namespace contains classes for creating Windows-based applications that take full advantage of the rich user interface features available in the Microsoft Windows operating system.

System.Xml

The System.Xml namespace provides standards-based support for processing XML. The supported standards are: XML 1.0, XML Namespaces, XSD Schemas, XPath expressions, XSLT transformations, DOM Level 1 Core, DOM Level 2 Core

Future Evolution

Today, .NET brings businesses a tremendous set of resources for realizing the benefits necessary to create secure, scalable, interoperable applications. From memory management to security to standards-based interoperability, .NET allows businesses to quickly build applications on a variety of platforms using a consistent programming model, with a variety of languages.

Microsoft continues to improve the development experience with version 2.0 of .NET – previously code-named “Whidbey”. The emphasis on productivity, security and interoperability continue. Support for 64-bit computing architectures, generics, lightweight transaction programming and developments in Web services standards over the last two years are a few of the technology changes in the core framework. More detail on a few of these features are called out below.

64-bit Support

.NET 2.0 will allow for the creation of native 64-bit managed applications to support the newer processors from Intel and AMD. Today, with non-managed 32-bit applications, they may continue to run on 64-bit platforms, but they will run through a thunking layer that will likely significantly degrade performance. With .NET, the CLR will compile the MSIL to native code for that processor, removing the need to use a thunking layer. More importantly, developers will continue to code as they do now, with no changes required to support 32-bit or 64-bit.

Generics

Generics permit classes, structs, interfaces, delegates, and methods to be parameterized by the types of data they store and manipulate. Without generics, general purpose data structures can use the generic object type to store data of any type. While the use of the generic object type makes the implementation very flexible, it is not without drawbacks. For example, it is possible to send a value of any type, such a Customer instance, into the class. However, when a value is retrieved, the result must explicitly be cast back to the appropriate type, which is tedious to write and carries a performance penalty for run-time type checking. Generics provide a facility for creating types that have type parameters, allowing applications to pass in the appropriate type at runtime and avoid the run-time type checking penalty.

Nullable types

Support for nullability across all types, including value types, is essential when interacting with databases, yet general purpose programming languages have historically provided little or no support in this area. Many approaches exist for handling nulls and value types without direct language support, but all have shortcomings. For example, one approach is to use a “special” value (such as -1 for integers) to indicate null, but this only works when an unused value can be identified. Another approach is to maintain boolean null indicators in separate fields or variables, but this doesn’t work well for parameters and return values. A third approach is to use a set of user-defined nullable types, but this only works for a closed set of types. Nullable

types solve this long standing problem by providing complete and integrated support for nullable forms of all value types.

Lightweight transactions

.NET 2.0 will allow developers to create transactions simply by declaring one in code. These transactions do not go through the overhead of using COM+ Component Services; instead, they run through managed code.

Cache invalidation

One of the biggest advantages that both data and output caching provide to the Web developer is the ability to avoid incurring the cost of processing or retrieving expensive resources on each request. Of course, one of the most expensive resources is often the trip to the database, so caching is especially useful for storing in-memory copies of datasets, or cached versions of pages that are data-driven.

The problem comes when applications have a page that needs to have the most up-to-date data possible. In ASP.NET v1.1, there is no built-in way to ensure that cached data does not become stale, beyond setting a very short cache duration, or using partial page caching to only cache the portion of the page that will not change.

ASP.NET Whidbey remedies this by adding support for invalidating cached items or output cached pages based on changes in Microsoft® SQL Server™ data. This feature is fairly simple to implement, and is supported on SQL Server 7, 2000, and 2005.

For more information, please refer to the ASP.NET 2.0 Whitepaper.

Edit and Continue

Visual Basic has always been about rapid application development (RAD). One key feature is the ability to fix runtime errors on the fly. This feature is now supported by the runtime, which means that any .NET language can support edit and continue. If applications run into an exception at runtime, they get an exception helper that provides tips for fixing common errors, but more importantly, developers can edit the code, select F5, and it continues right where they left off. This feature will be available in both Visual Basic .NET and C# in the next release of .NET.

ClickOnce

If a development staff has ever worked with href-exes in the .NET Framework 1.0 and .NET Framework 1.1, they've probably realized how many deployment issues href-exes solve. Href-exes are also known as no-touch deployment, or zero impact deployment. Essentially, developers can add a link in a Web page that points to an executable file. When the user

clicks the link, the application downloads to their Internet files cache and runs. To keep this from being a huge security hole, the application permissions are restricted based on the URL (Intranet applications get different permissions than Internet applications, for example), or other factors. This means that some applications no longer need to be deployed in the traditional sense; no more setup.exe or Windows Installer file.

But for all their glory, href-exes have some limitations. First, the .NET Framework must be pre-installed on the client machine. There's no good way to bootstrap the .NET Framework down if it's not there. Also, most non-trivial applications consist of the main .exe and a number of assembly files. With href-exes, the assembly files are downloaded on demand, which is great for corporate Intranet applications, but there's no way to download the application in one shot so that users know it can be safely used off-line. There are other issues as well, such as limited support for versioning, the application doesn't hook into Add/Remove Programs, and the application doesn't install Start menu shortcuts.

The Whidbey release of Visual Studio .NET takes href-exes a big leap forward with the advent of the ClickOnce deployment model. ClickOnce deployment differs from href-exes in some significant ways.

First, ClickOnce applications are self-updating. With href-exe's developers can only make the application self-updating through the use of custom code or through the use of a component, such as the .NET Application Updater component. With ClickOnce, updates can also be marked as mandatory or optional. ClickOnce works its magic through two XML manifest files. The first is an application manifest and the second is a deployment manifest. The application manifest describes the application itself, which includes information about the application assemblies, dependencies, and files that make up the application. The application manifest also states the required permissions, and the location where updates can be downloaded. The deployment manifest points to the location of the application manifest and files, and instructs the clients on which version of the application they should be running.

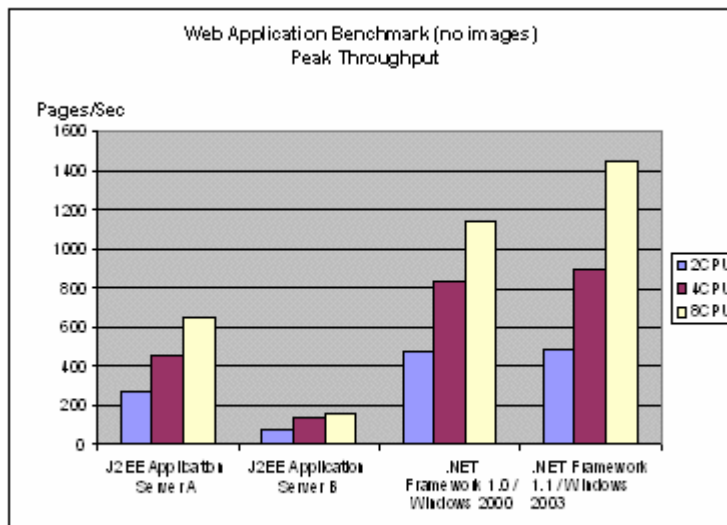
When applications are installed through ClickOnce, they show up in the start menu and they can be uninstalled through the Add/Remove Programs feature. Developers also control when the application scans for updates. Options include on startup or while the application is running. Developers also control whether the entire application is downloaded (which is a good choice if it is to be used off-line), or if the application trickles down on demand.

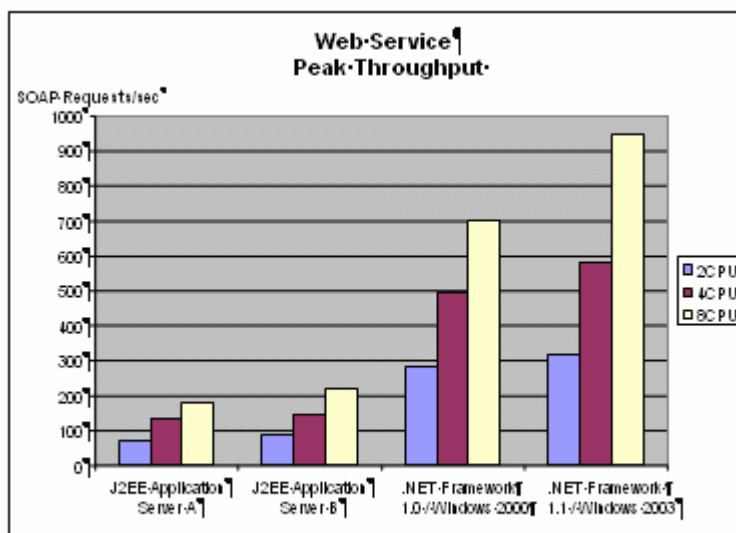
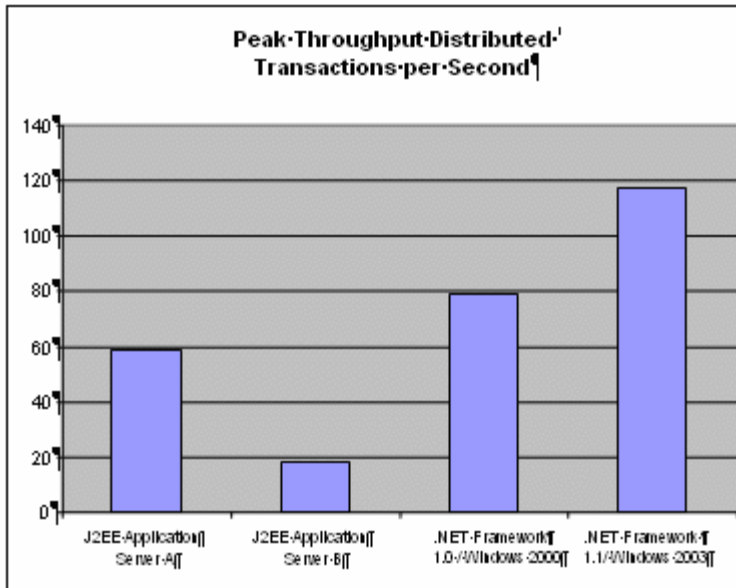
Appendix: Performance Evidence and Quotes

The .NET Framework improves the performance of typical Web applications.

- The Middleware Company, founders of the leading J2EE developer forum TheServerSide.com, have conducted a benchmark of the .NET Framework and J2EE, finding the .NET Framework to significantly outperform J2EE for Web application hosting, Web services, and distributed transactions, as shown in the graphs below.
- The .NET Framework also offers significant performance and scalability benefits over the previous Active Server Pages (ASP) technology, thanks to its just-in-time (JIT) compilation and caching technologies.

The results below were achieved with both Windows 2000 Advanced Server and Windows Server 2003 running versions 1.0 and 1.1 of the .NET Framework, respectively. Both setups used a SQL Server 2000 database.





In the Industry

"We get subsecond page loads while handling millions of page views a day. We deployed on December 23, 2000 and haven't had a minute of downtime as of October 3, 2001, and we saved \$1.3 million (US) over a Java 2 Enterprise Edition solution." Stephen Forte, chief technology officer, Zagat Survey

"Compared with similar projects in the past, we're measuring deployment time in hours instead of weeks." *Ferdy Khater, director of application development, Continental Airlines*

Agile Architecture

Companies worldwide are using the XML Web services communication mechanism that is native to the .NET Framework to integrate quickly and easily with suppliers and customers.

In the Industry

"From our partners' perspective, accessing our content via XML Web services will be far easier than what they've had to go through in the past. They will no longer need to build the infrastructure to import, store, and manage it. When combined with our new flexibility in licensing options, this means we'll have a far more attractive package to offer to prospective partners." *Stephen Forte, chief technology officer, Zagat Survey*

"This makes it easier for us to inform portals and enterprises about how our code handles user data, security concerns, and integration with existing databases. Particularly handy are the automatically generated documentation and test Web pages, which enable our partners to integrate their systems with ours using minimal assistance." *Tore Lode, senior developer, CyberWatcher*

References

Improving Web Application Security: Threats and Countermeasures by J.D. Meier, Alex Mackman, Michael Dunner, Srinath Vasireddy, Ray Escamilla and Anandha Murukan

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/THCMCh06.asp>

.NET Framework Developer's Guide

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconMicrosoftIntermediateLanguageMSIL.asp>

Side-By-Side and Versioning Considerations for .NET Remoting by Piet Obermeyer and Jonathan Hawkins

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/versremote.asp>

Simplifying Deployment and Solving DLL Hell with the .NET Framework by Steven Pratschner

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/dplywithnet.asp>

Introduction to ASP.NET

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconintroductiontoasp.asp>

Microsoft .NET Framework Delivers the Platform for an Integrated, Service-Oriented Web by Jeffrey Richter

<http://msdn.microsoft.com/msdnmag/issues/0900/Framework/>

Debugging in Visual Studio .NET by Shaykat Chaudhuri

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dv_vstechart/html/vstchDebuggingInVisualStudioNET.asp

Caching Improvements in ASP.NET Whidbey by G. Andrew Duthie

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnvs05/html/aspnetwhidbey_caching.asp

A Programmer's Introduction to Visual Studio .NET "Whidbey" by Sean Campbell and Scott Swigart

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dv_vstechart/html/vs2004_intro.asp

Technology Overview

<http://msdn.microsoft.com/netframework/technologyinfo/overview/>

Common Language Runtime Overview

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconcommonlanguageruntimeoverview.asp>)

Is COM Dead? by Don Box

<http://msdn.microsoft.com/msdnmag/issues/1200/com/default.aspx>

Microsoft .NET Framework Delivers the Platform for an Integrated, Service-Oriented Web by Jeffrey Richter

<http://msdn.microsoft.com/msdnmag/issues/0900/Framework/>
<http://msdn.microsoft.com/msdnmag/issues/1000/Framework2/>

Handling Language Interoperability with the Microsoft .NET Framework by Damien Watkins

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/interopdotnet.asp>

.NET Security Overview by J.D. Meier, Alex Mackman, Michael Dunner, Srinath Vasireddy, Ray Escamilla and Anandha Murukan

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/THCMCh06.asp>

Avoiding DLL Hell: Introducing Application Metadata in the Microsoft .NET Framework by Matt Pietrek

<http://msdn.microsoft.com/msdnmag/issues/1000/metadata/>

Class Library

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/cpref_start.asp

Introduction to the .NET Framework Class Library

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconTheNETFrameworkClassLibrary.asp>

Moving Java Applications to .NET

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/dotnet_movingjavaapps.asp

Simplifying Deployment and Solving DLL Hell with the .NET Framework by Steven Pratschner

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/dplywithnet.asp>

Side-by-Side and Versioning Considerations for .NET Remoting by Piet Obermeyer and Jonathan Hawkins

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/versremote.asp>

Automatic Memory Management

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconautomaticmemorymanagement.asp>

Debugging Managed Code

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vsdebug/html/vxoriDebuggingManagedCode.asp>

Manageability

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/vsent7/html/vxoriManageability.asp>