



Microsoft Home

Search for

All MSDN

Advanced Search

.NET Architecture Center Home

Enterprise Architecture

Application Architecture

Systems Architecture

Patterns ▶

Architecture Community ▶

Read Me First: Microsoft Architecture Overview

What's Next: Upcoming Architectural Content

Developer

IT Pro



Products | Downloads | Support | Search | Worldwide | Microsoft

[Microsoft Home](#) > [.NET Architecture Center Home](#) > [Application Architecture](#)
[See This in MSDN Library](#)

A Guide to Building Enterprise Applications on the .NET Framework

Building the Next Generation of Service-based Software Systems

Microsoft Corporation

September 2003

Applies to:

Microsoft® .NET Framework

Microsoft Windows Server™ 2003

Summary: Learn about the technical basis for building enterprise-scale Web-based applications using the Microsoft .NET Framework and Microsoft Windows Server 2003. (40 printed pages)

Contents

[Introduction](#)
[Service Framework](#)
[The Unified Class Libraries](#)
[Service Presentation](#)
[.NET Development](#)
[Summary](#)
[Glossary](#)
[For More Information](#)

Introduction

This document describes the technical basis for building enterprise-scale Web-based applications using the Microsoft® .NET Framework and the Microsoft Windows Server™ 2003 platform. This new generation of applications is needed to meet the demands of enterprise computing over an Internet standard distributed network infrastructure.

This software model integrates the power of Web-based solutions with the distributed application model of traditional multi-tier client/server design. While client/server design has traditionally relied on proprietary technologies to control information flow between the tiers, current solutions take advantage of industry-standard communications protocols to harness the power of the Internet. By creating solutions based on a combination of supplied and created services, more powerful and flexible applications can be built in a fraction of the time required using previous development methodologies.

History of Web Development

The Web started as basically a read-only file system, enabling easy content access using industry standards and protocols.

Page Options

Average rating:
5 out of 9
 [Rate this page](#)
[Print this page](#)
[E-mail this page](#)
[Discuss this page](#)
[Add to Favorites](#)

The first interactive Web applications were typically outward extensions of existing two-tier applications that exposed and rendered data in a Web browser. Early Web development was typically based on the C programming language and the Common Gateway Interface (CGI), with which very few programmers had experience. As a result, development costs for dynamic Web applications were high.

In addition, most of these Web applications were built on two-tier architectures, creating challenges around scalability and application integration. Developers typically did not design Web applications to be used by anything other than the Web page they were hosting. In other words, the user interface and the application logic were the same thing. Consequently, it was difficult to link Web applications together to form more useful aggregations.

As a result of advances in the Microsoft Component Object Model (COM) and the release of technologies such as Microsoft's Active Server Pages (ASP) in 1996, Web sites offered a more interactive user experience. ASP made it easy to call the business logic and platform services that developers need through simple script languages. COM support made it easy to write applications through its ability to package this business logic into modular units that can be written in a wide range of popular programming languages, particularly Microsoft Visual Basic® and Microsoft Visual C++®.

The Web could now offer richer user experiences and it was taking basic steps to overcome some of the challenges of application integration with tricks such as using HTML frames to embed one company's Web site within another, and HTML "screen scraping" to extract data from Web pages.

These strategies for application integration had shortcomings. Simply put, they were (and still are) brittle: What happened if the other Web site changed its content, leaving the page with a broken link?

Advancements in Web development are rapidly moving from this two-tier architecture to an n -tier design, which enables a richer integration strategy by exposing business objects and middle-tier logic to Web and partner integration. The challenge with trying to use encapsulated business logic in this way is that most of these applications are designed on tightly coupled, proprietary protocols.

Current challenges

In the past, the companies that have tried to offer solutions for enabling a Web site to expose application integration information and functionality in a modular, scalable, and Internet-friendly way have encountered significant challenges. Chief among these challenges are the following:

- **Time to market.** The length of development time for getting an application or Web site to market may render the offering no longer viable.
- **Scaling to the Web.** Existing object models and component designs simply do not work over Internet protocols. Stateless application development that can be rerouted and served by any server is a new concept for many developers. Yet such a design pattern is vitally important to achieve global scalability.
- **Lack of end-to-end development tools.** Tool sets available today don't empower organizations with the flexibility necessary to stay ahead of their competitors. In the rapidly changing world of the Internet, organizations must exhibit the agility to integrate with new partners, using development tools that solve the challenges of today's heterogeneous computing environments.

Service-oriented architecture

Perhaps the biggest challenge facing technology professionals today is how to leverage and manage the diverse IT assets within an organization. Software developers need to

be able to reuse organizational assets to reduce time-to-market and cut code maintenance costs. IT managers need to understand and permit appropriate access to network resources, regardless of the platform or programming language used to deliver those resources. Everyone needs insight into the problems that arise as the application portfolio is used to drive the organization's business.



Figure 1. Web services

To address these challenges, many organizations are realizing the benefits of exposing key software components as network-addressable services. In a service-oriented architecture (SOA), data, logic, and infrastructure assets are accessed by routing messages between network interfaces. Services encapsulate and componentize complex processes and systems, permitting controlled change and continuous improvement of the underlying implementations.

An SOA realizes its full potential when it permits cross-platform integration, so that, for example, applications written in a variety of languages can all share the same exception management services. To this end, Microsoft is working with a consortium of system vendors to develop a platform-independent framework for services around XML and its related technologies.

In an SOA, applications composed of Web services expose their functionality programmatically over the Internet or intranet using Internet protocols and standards, such as HTTP and XML.

Web services are Microsoft's implementation of an SOA. Web services solve the challenges facing Web developers by combining the tightly coupled, highly productive aspects of *n*-tier computing with the loosely coupled, message-oriented concepts of the Web. Think of Web services as component programming over the Web.

While a single application may be designed, developed, and deployed following the principles of service orientation, the benefits of an SOA are best realized when an organization commits to structuring its application portfolio and technology architecture around reusable, composable services.

Distributed Application Design

Designing a distributed application involves numerous decisions—typically made by multiple people—about its logical and physical design and the technologies and infrastructure used to implement it. To help guide this process, it is necessary to have a conceptual model for the partitioning of functionality in the application. The following diagram—taken from the Microsoft guide [Application Architecture for .NET: Designing Applications and Services](#)—shows a typical conceptual view, which is appropriate to a wide range of specific application scenarios:

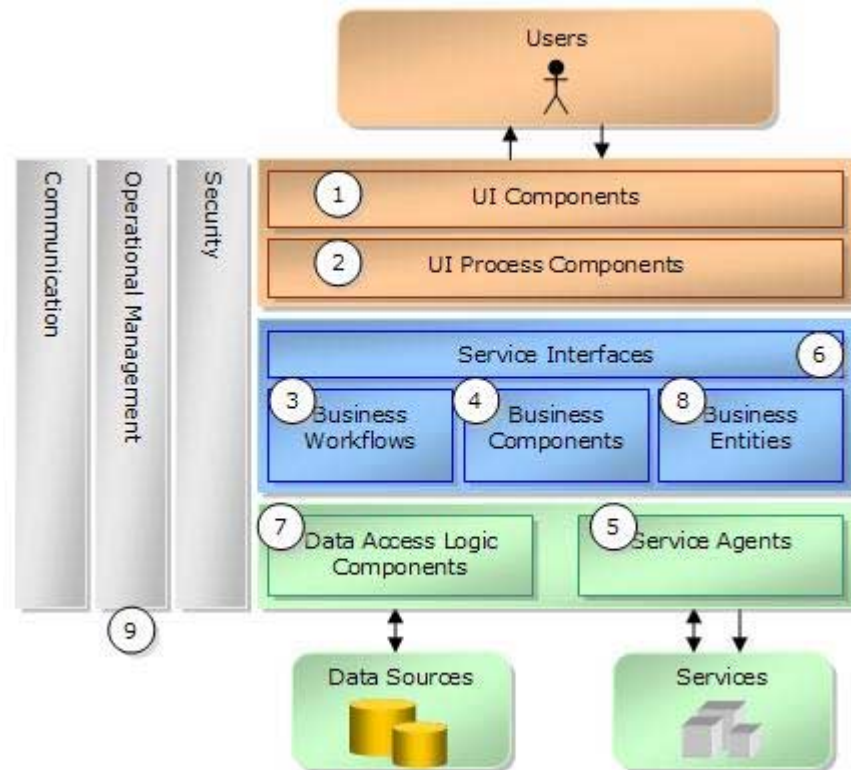


Figure 2. Layered application: conceptual view

The components in this conceptual view include:

- **User Interface (UI) components.** Most solutions provide some way for users to interact with the application. User interfaces are implemented using smart clients or Web pages to acquire, validate, render and format user data.
- **User process components.** Many solutions synchronize and orchestrate these user interactions using separate user process components. This way the process flow and state management logic is not hard-coded in the user interface elements themselves, and the same basic user interaction "engine" can be re-used by multiple user interfaces.
- **Business workflows.** Many business processes involve long running multi-step processes that must be performed in the correct order and coordinated with other services.
- **Business components.** Virtually all applications and services require components that implement business rules and perform business tasks. Business components implement the business logic of the application.
- **Service agents.** Business components that require functionality provided in external services must communicate with those services. Service agents isolate the application from the idiosyncrasies of calling services and can re-map data as appropriate for the application.
- **Service interfaces.** In order to expose business logic as a service, service interfaces must be created that support the communication contract (message-based communication, formats, protocols, security, exceptions, and so on) that are needed by its different consumers.
- **Data access logic components.** Almost all applications and services will need to access a data store at some point during a business process. Abstracting the data access logic in a separate layer centralizes data access functionality and makes it easier to configure and maintain.
- **Business entity components:** Most applications require data to be passed between components. For example, in the retail application a list of products must be passed from the data access logic components to the user interface

components, so that the product list can be displayed to the users. The data is used to represent real world business entities, such as products or orders. The business entities that are used internally in the application are usually data structures provided by the platform, but they could also be implemented using custom object-oriented classes that represent the real world entities that applications have to deal with, such as a product or an order.

- **Components for Security Operational Management and Communication:** Applications take advantage of platform features for security, exception management, and communications. Applications must also support a broad spectrum of operational technologies including administration, management, and deployment. For more information on these topics, visit the [.NET Architecture Center](#).

Technical Architecture

Technical architecture is the technology basis for achieving an enterprise-scale Web service-oriented architecture in an organization. The high-level service technology diagram shown below illustrates a set of generic layers that provide enterprise-based services for Web service generation. These levels contain the common elements that are required by any Web service application or system.

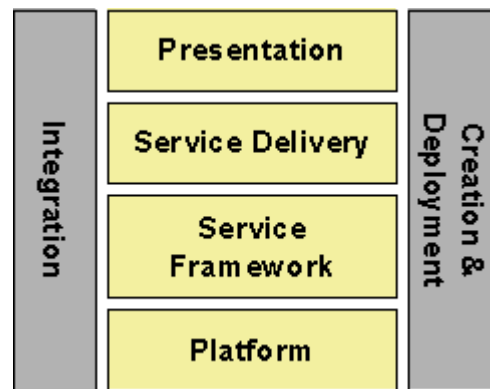


Figure 3. A conceptual view of the technology architecture

- **Platform**—Forms the base of the conceptual view, providing operating system, hardware, storage, networking, and the trust and management services for the whole system. Many of these capabilities may be provided as part of the operating system.
- **Framework**—Provides the process and state management, language interoperability, and common libraries required by a service-based application. The libraries allow developers to use a higher-level, consistent abstraction of the underlying platform.
- **Delivery**—Contains the communications and service interfacing required to support the location and consumption of Web services. These services can be consumed by the presentation layer as well as external systems.

The framework and the platform together form the full enterprise application server, which provides specific support for Web services.

- **Presentation**—Contains the client services that focus on user presentation issues and technologies, and provides support for all types of clients including devices. The functionality at this layer may reside on the client and devices, or may be provided by Web servers that generate HTML presentation. In Web applications, the presentation may also include processing done by the client browser.

At the same time, two additional conceptual areas are used to express functionality that encompasses all the other conceptual layers:

- **Integration**—Provides integration and interoperation between services and present-day operational systems: legacy applications, commercial applications, databases, and other Web services. This is commonly called enterprise application

integration (EAI).

- **Creation and Deployment**—Includes the tools, process, methodologies, and patterns required to support the entire application life cycle, including the design, development, testing, deployment, and management of enterprise solutions built on Web services.

Microsoft Service Framework

The Microsoft platform for service-oriented architecture includes the .NET Framework running on Windows Server 2003. The balance of this document describes how this platform can be used to build enterprise-scale Web applications.

Introduction to the .NET Framework

The .NET Framework is an integral 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 much of the plumbing involved in developing and running a piece of software, enabling developers to focus on writing the core business logic code. The .NET Framework provides:

- A consistent, language-neutral, object-oriented programming environment.
- A code-execution environment that minimizes software deployment and versioning conflicts, guarantees safe 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.

The .NET Framework is composed of two key parts: the common language runtime (CLR) and the class libraries. These will be covered in detail in the next section, "Service Framework."

Windows Server 2003

The Windows Server 2003 family forms the base platform for an enterprise service-oriented architecture. At the cornerstone is native-mode Microsoft .NET functionality through the .NET Framework and standards-based technologies, which will enable businesses to easily and seamlessly connect information, people, systems, and devices.

As the platform for a service-oriented architecture, Windows Server 2003 provides the operating system, storage, networking, security, and management services on which solutions are built. Windows Server 2003 is the foundation enabling an unprecedented level of software integration through the use of XML-based Web services.

Service Framework

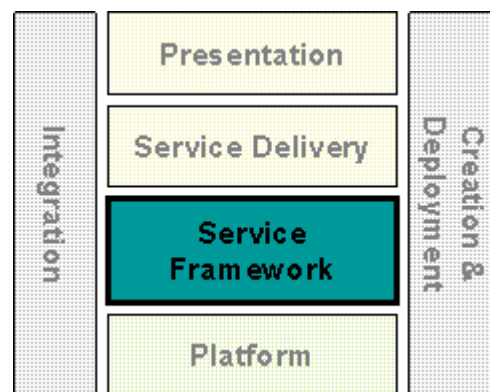


Figure 4. Service framework

A great deal of the functionality of any software application system is commonly provided by a framework that supports building solutions. The framework provides infrastructure functionality—such as reliability, scalability, communications, and so on—as well as an application model that enables developers to easily construct and assemble applications using a variety of programming tools and languages. The .NET Framework provides exactly this functionality for building software solutions based on the Windows platform.

.NET Framework Applications

The .NET Framework can be used to build many different types of Windows applications, including those that run on servers—such as Web applications and Web services—as well as those that run on client side.

In particular, the application server solution—composed of the .NET Framework, Windows Server 2003, and Microsoft Visual Studio® .NET—can provide the following .NET Framework applications:

- **ASP.NET**—Used for building server-based Web applications, ASP.NET is an evolution of the widely-used Active Server Pages (ASP) programming model. Using the ASP.NET mobile controls, these applications can target small devices—such as PDAs and phones—that support a Web browser.
- **Web Services**—Web services are components which facilitate integration by sharing data and functionality over the network through standard, platform-agnostic protocols such as XML, SOAP, and HTTP.
- **Server Components**—Communication between different objects and services running on the same or remote machines.
- **Other**—The application server can also host .NET Framework programs written as console applications or Windows services.

On the client, the .NET Framework can be used for the following types of applications:

- **Smart Client**—Programs that run on the familiar Windows desktop that take advantage of the rich user interface capabilities of these operating systems.
- **Smart Client on Smart Devices**—Applications that run on the Pocket PC and Windows CE .NET platforms and use the .NET Compact Framework.
- **Other**—As with the server, the Windows client can also execute .NET Framework programs written as console applications or Windows services.

All of these application types make use of the .NET Framework, though the details of how they are built, their user interfaces, and the functional areas of the .NET Framework used by the applications will all vary significantly.

.NET Framework Design Goals

The .NET Framework is a set of class libraries and a runtime for rapidly building and efficiently operating Web services and applications on the Windows operating system. The vision for the .NET Framework is to combine a simple-to-use programming paradigm with the scalable, open protocols of the Internet. To achieve this vision several intermediate goals had to be delivered.

The Common Language Runtime



Figure 5. Major components of the common language runtime (CLR)

The common language runtime (CLR) is a high-performance engine for running applications built using the .NET Framework. Code that targets the runtime and whose execution is managed by the runtime is referred to as *managed* code. Responsibility for tasks such as creating objects, making method calls, and so on is delegated to the CLR which enables it to provide additional services to the code as it executes.

While the component is running, the CLR provides services—such as memory management (including garbage collection), process management, thread management, and security enforcement—and satisfies any dependencies that the component may have on other components.

Despite its name, the CLR also has a role in a component's development-time experiences. Because it automates so much (for example, memory management), the CLR makes the developer's experience very simple. In particular, features such as lifetime management, strong type-naming, cross-language exception handling, delegate-based event management, dynamic binding, and reflection dramatically reduce the amount of code a developer must write in order to turn business logic into reusable components.

Runtimes are nothing new for languages; virtually every programming language has a runtime. Visual Basic has a well known runtime (the aptly-named VBRUN), but Microsoft Visual C++® also has one (MSVCRT), as do Microsoft JScript®, SmallTalk, Perl, Python, and Java. The critical role of the common language runtime, and what really sets it apart, is its provision of a *unified* runtime environment across all programming languages.

The key features of the runtime include a common type system (enabling cross-language integration), self-describing components, simplified deployment and versioning, and integrated security services.

Common type system and multilanguage integration

To fully interact with other objects regardless of the programming language they were implemented in, objects must expose to callers only those features that are common to all the languages with which they must interoperate. The CLR makes use of a new common type system capable of expressing the semantics of modern programming languages. The common type system defines a standard set of data types and rules for creating new types. The CLR understands how to create and execute these types. The different language compilers for the .NET Framework use the CLR to define data types, manage objects, and make method calls instead of using tool- or language-specific methods.

The Common Language Specification (CLS) defines a set of language features to which

programming languages must adhere in order to integrate with the .NET Framework. This goes beyond the common type system, specifying rules about events, exception handling, metadata (to be discussed in more detail later), and so on.

The CLS and common type system enable deep multilanguage integration, enabling the following activities to take place transparently across components written in different languages, without any additional work on the part of the developer:

- Calling methods on other objects
- Inheriting implementations from other objects
- Passing instances of a class to other objects
- Using a single debugger across multiple objects
- Trapping errors from other objects

This means that developers no longer need to create different versions of their reusable libraries for each programming language or compiler, and developers using class libraries are no longer limited to libraries developed for the programming language they are using or forced to create COM wrappers to facilitate the interaction.

Cross-language integration is useful in several scenarios. For example, development projects gain access to a larger skills base—project managers can choose developers skilled in any programming language and join them together on the same team. Alternately, developers writing components for a distributed application will find it helpful to know that no matter what language they choose to write their components in, those components can interact closely with each other and with components supplied by other developers.

Metadata and self-describing components

The .NET Framework enables the creation of self-describing components, which simplify development and deployment and improve system reliability. Self-description is accomplished through metadata—information contained in the binary that supplements the executable code, providing details about dependencies, versions, and so on. The metadata is packaged together with the component it describes, resulting in self-describing components.

A key advantage of self-describing components is that no other files are needed in order to use a component. This contrasts with typical application development today, which requires separate header files for class definitions, separate Interface Description Language (IDL) files, separate type libraries, and separate proxies and stubs. Since the metadata is generated from the source code during the compilation process and stored with the executable code, it is never out of sync with the executable.

Because each application contains a full description of itself, the runtime can dynamically assemble a cache of information about the components installed on a system. If that cache becomes damaged somehow, the runtime can rebuild it without the user even knowing. In addition to solving development challenges, self-description eliminates the dependency on the Windows registry for locating components. A benefit of not relying on the Windows registry is the ability to do XCOPY deployment.

Deployment and shared assemblies

Many server applications and services built with the .NET Framework can be deployed simply by copying the application's files to a target machine, without any registration of the dynamically-linked libraries (DLLs) or installation scripts. By default, applications are completely self-contained. An executable lives in an application directory with all the DLLs and resources it needs. Applications can be copied onto a target server

machine or delivered by common media—CD, DVD, floppy disk—as well as via application deployment infrastructure such as Microsoft Systems Management Server. Applications can also be deployed from a remote Web server using HTTP.

Administrators may share DLLs among multiple applications using the .NET Framework global assembly cache (GAC). The GAC effectively serves as an intelligent store for DLLs and should be used in place of the system registry when working with DLLs written using the .NET Framework. It can contain multiple versions of a given DLL. Each DLL entered into the GAC is entered with a "strong name." This consists of a digital signature, the name of the DLL, its version number, and the culture for which it was created (for example, English, German, or Japanese). This information enables the GAC to differentiate between different versions of a given DLL.

The isolation of DLLs in their own application directories in the XCOPY deployment scenario and the global assembly cache in the shared DLL scenario enable side-by-side execution without any conflicts. In fact, this is true of the .NET Framework as well: different versions of the .NET Framework may be installed and execute on a machine concurrently.

Side-by-side execution eliminates "DLL hell"—a common scenario in which a new version of a DLL would replace an older version in the system registry, resulting in incompatibilities with applications that had been depending upon functionality or behavior present only in the older version.

Code access security

The .NET Framework takes a major step forward in software security by introducing a fine-grained, evidence-based security system called code access security. This security system now gives the systems administrator a wide range of granular permissions that they can grant to code in place of the "all-or-nothing" or "sandbox" security models available with many earlier software technologies.

The Unified Class Libraries

The classes of the .NET Framework provide a unified, object-oriented, hierarchical, and extensible set of class libraries, or APIs, that developers can use from the languages with which they are already familiar.



Figure 6. The .NET Framework unified classes

Previously, Visual C++ developers used the Microsoft Foundation Classes, Visual J++ developers used the Windows Foundation Classes, and Visual Basic developers use the Visual Basic runtime. Simply put, the classes of the .NET Framework unify these

different classes, creating a unified set of their features across all languages. The result is that developers no longer have to learn multiple object models or class libraries. By creating a common set of APIs across all programming languages, the .NET Framework enables such powerful features as cross-language inheritance, error handling, and debugging. In effect, all programming languages—from JScript to C++—become equal and developers are free to choose the right language for the job at hand.

The .NET Framework provides classes that can be called from any programming language. These classes comply with a set of naming and design guidelines to further reduce training time for developers. Some of the key class libraries are shown in the figure at above.

The .NET Framework includes a base set of class libraries that developers would expect in any standard library, such as collections, input/output, data type, and numerical classes. In addition, there are classes that provide access to all of the operating system services, such as graphics, networking, threading, globalization, cryptography, and data access; classes that development tools can use, such as debugging; and a set of classes that supply the services necessary for building enterprise-scale applications, such as transactions, events, partitions, and messaging.

We will first take a look at the three high-level namespaces used to build server and client applications, specifically:

- **ASP.NET** (System.Web)
- **Web Services** (System.Web.Services)
- **Windows Forms** (System.Windows.Forms)

ASP.NET

A set of classes within the unified class library, ASP.NET provides a set of controls and infrastructure that make it simple to build Web applications.

ASP.NET comes with a set of server-side controls (sometimes called Web forms) that mirror the typical HTML user interface widgets (including list boxes, text boxes, and buttons), and an additional set of Web controls that are more complex (such as calendars and ad rotators). These controls actually run on the Web server and project their user interface as HTML to a browser. On the server, the controls expose an object-oriented programming model that brings the richness of object-oriented programming to the Web developer.

One important feature of these controls is that they can be written to adapt to client-side capabilities—the same pages can be used to target a wide range of client platforms and form factors. In other words, ASP.NET controls can sniff the client that is requesting a page and return an appropriate user experience—HTML 3.2 for a down-level browser and dynamic HTML for Internet Explorer 5.5. In the case of the ASP.NET mobile controls, which will be discussed in more detail below, the controls can even further, returning compact HTML (cHTML) or WML, for example, if required.

ASP.NET also provides features such as cluster session-state management and process recycling, which dramatically increase application reliability.

ASP.NET works with all development languages and tools (including Visual Basic, C++, C#, Visual J#, and JScript). How ASP.NET works is covered in the Web forms section of Service Presentation later in this document.

Web services

Integrating systems has historically been a daunting and expensive process.

Integration solutions were often point-to-point, minimizing opportunities for code reuse should one of the systems in question eventually need to integrate with other systems. Solutions were also often brittle, relying on technology such as HTML "screen-scraping," which broke down whenever the "client" Web site changed its format.

Web services provide a low-cost, scalable, and flexible solution to the challenges of integration. Simply put, Web services are components that facilitate integration by sharing data and functionality over the network through standard, platform-agnostic protocols such as XML, SOAP, and HTTP.

Web services solve the challenges facing Web developers by combining the tightly coupled, highly productive aspects of *n*-tier computing with the loosely coupled, message-oriented concepts of the Web. Thus, Web services enable Web-based component programming.

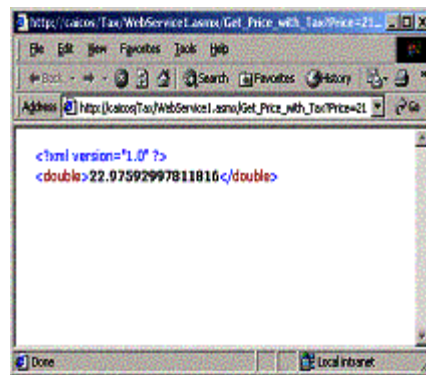


Figure 7. Example of XML being returned from a Web service

Conceptually, developers integrate Web services into their applications by calling the methods exposed in this manner just as they would call local services. The difference is that these calls can be routed across the Internet to a service residing on a remote system. For example, a Web service such as Microsoft Passport provides authentication functionality to client applications from a remote set of Microsoft-hosted servers. By programming against the Passport service, a developer can take advantage of Passport's infrastructure and rely on Passport to maintain the database of users, make sure that it is up and running, backed up properly, and so on—thus shifting a whole set of a development and operational chores.

Clearly, a considerable amount of infrastructure is required to make building Web services transparent to developers and users. The Microsoft .NET Framework provides that infrastructure in the **System.Web.Services** classes. There, the .NET Framework supplies an application model and key enabling technologies to simplify the creation, deployment, and ongoing evolution of secure, reliable, scalable, and highly available Web services while building on existing developer skills.

To the .NET Framework, all components can be Web services, and Web services are just another kind of component. The result is a powerful, productive Web component system that simplifies program plumbing, deeply integrates security, introduces an Internet-scale deployment system, and greatly improves application reliability and scalability.

Windows Forms

Although much attention has been given to the Microsoft .NET Framework as a way to develop Web services and Web applications, the unified class libraries also offer full support for developing Windows-based smart client applications (these applications can

use Web services, too).

Developers who write client applications for Windows can use the Windows Forms (in the **System.Windows.Forms** namespace) classes to take advantage of all of the rich user interface features of Windows, reuse existing Microsoft ActiveX® controls, and harness the graphical capabilities of the GDI+ graphics library. Smart client applications may consume Web services in the same manner as ASP.NET Web applications. Overall, developers will find the Windows Forms programming model and design-time support very intuitive, given their similarities to existing Windows-based forms packages.

Since it is part of the .NET Framework, Windows Forms can automatically take advantage of .NET Framework features such as a common type system; strongly typed code; memory management; and all of the deployment, security, and administration features of .NET Framework-based applications.

Data access with ADO.NET

Nearly all applications need to query or update persisted data, whether it is contained in simple files, relational databases, or any other type of store. To fulfill this need, the .NET Framework includes ADO.NET, a data access subsystem optimized for *n*-tier environments. For easy interoperability, ADO.NET uses XML as its native data format, and thus ADO.NET is interoperable with XML and XML documents. As the name implies, ADO.NET evolved from ADO (ActiveX Data Objects), and it builds on the huge library of ODBC drivers already available.

ADO.NET is designed for loosely coupled environments and provides high-performance stream APIs for disconnected data models, which maximize scalability and are thus more suitable for returning data to Web applications.

As applications are developed, there will be different requirements for working with data. In some cases, it may be necessary to simply display data on a form. In other cases, it may be necessary to devise a way to share information with another company. No matter what the requirements are, there are a few fundamental concepts that are key to understanding data handling in the .NET Framework.

Disconnected Data Design—In traditional two-tier applications, components establish a connection to a database and keep it open while the application is running. For a variety of reasons, this approach is impractical in many applications:

- Open database connections take up valuable system resources.
- Applications that require an open database connection are extremely difficult to scale.
- In Web applications, the components are inherently disconnected from each other.
- A model based on connected data can make it difficult to share data between components, especially components in different applications.

For all these reasons, data access in ADO.NET is designed around a disconnected design. Applications are connected to the database only long enough to fetch or update the data. Because the database is not hanging on to connections that are largely idle, it can service many more users.

Scalability—The Web can vastly increase the demands for data, and scalability has become critical. Internet applications have a limitless supply of potential users. An application that consumes resources such as database locks and database connections will not serve high numbers of users well, because the user demand for those limited resources will eventually exceed their supply. Applications using ADO.NET employ disconnected access to data, so database locks or active database connections are not maintained for long durations, thus improving performance and scalability.

XML and data

Data and XML are tightly integrated in the .NET Framework, where the **System.Xml** namespace provides standards-based support for processing XML.

In fact, in ADO.NET, XML is the fundamental format for sharing and remoting data. When data is shared, the ADO.NET APIs automatically create XML files or streams out of information in the dataset, and sends them to another component. The second component can invoke similar APIs to read the XML back into a dataset. In fact, if an XML file is available, it can be used like any data source and a dataset can be created out of it.

There are several reasons to use XML:

- **XML is an industry-standard format.**
- **XML is text-based.** This enables XML to be sent through any protocol, such as HTTP.
- **Interoperability.** ADO.NET enables easy creation of custom XML documents through the use of XSD schemas.

ADO.NET applications can take advantage of the flexibility and broad acceptance of XML. Because XML is the format for transmitting datasets among components and across tiers, any component that can read the XML format can process an ADO.NET dataset. As an industry standard, XML was designed with exactly this kind of interoperability in mind.

It is not necessary to know XML in order to share data using ADO.NET. ADO.NET automatically converts data into and out of XML as needed; the developer interacts with the data using ordinary programming methods.

Interop Services

Interop Services provide a collection of classes useful for accessing the Microsoft Win32® APIs and COM objects from .NET Framework-based code. This allows developers to harness the functionality of legacy code as well as functionality not directly available in the .NET Framework class libraries.

Interop Services' platform invoke enables managed code to call unmanaged functions implemented in dynamic-link libraries (DLLs), such as those in the Win32 API.

COM Interop allows existing COM types to be instantiated and called from managed code. To enable this, a managed definition of the COM type is made available to the managed code by creating an assembly based on the COM component's type library. The same mechanism can also be used to go the other way. By creating and registering a COM type library based on the managed assembly, it is possible for COM components to instantiate and call managed types. The InteropServices namespace supports COM Interop and provides managed definitions of many common interfaces. Custom interface definitions can also be used.

The .NET Framework extends the COM model for reusability by adding implementation inheritance. Managed types can derive directly or indirectly from a COM coclass; more specifically, from the runtime callable wrapper generated by the CLR. The derived type can expose all the methods and properties of the COM object as well as methods and properties implemented in managed code. The resulting object is partly implemented in managed code and partly implemented in unmanaged code.

InteropServices is also used to indicate how data should be transferred—or "marshaled"—between managed and unmanaged memory. Finally, InteropServices manages exceptional circumstances such as errors while performing certain operations, and serves as a way to bridge COM HRESULTs to .NET Framework

exceptions.

Remoting

.NET remoting (available through the **System.Runtime.Remoting** namespace) can be used to enable different applications to communicate with one another, whether those applications reside on the same computer, on different computers in the same local area network, or across the world in very different networks—even if the computers run different operating systems.

The .NET Framework provides a number of services such as activation and lifetime control, as well as communication channels responsible for transporting messages to and from remote applications. Formatters are used to encode and decode the messages before they are sent along a channel. Applications can use binary encoding where performance is critical, or XML encoding where interoperability with other remoting systems is essential. Remoting was designed with security in mind, so the call messages and serialized streams can be accessed in order to secure them before they are transported over the channel.

The .NET remoting infrastructure is an abstract approach to interprocess communication. Much of the system functions without drawing attention to itself. For example, objects that can be passed by value, or copied, are automatically passed between applications in different application domains or on different computers. It is only necessary to mark a custom class as serializable to make this work.

The real strength of the remoting system, however, resides in its ability to enable communication between objects in different application domains or processes using different transportation protocols, serialization formats, object lifetime schemes, and modes of object creation. In addition, it is necessary to intervene in almost any stage of the communication process, for any reason, remoting makes this possible.

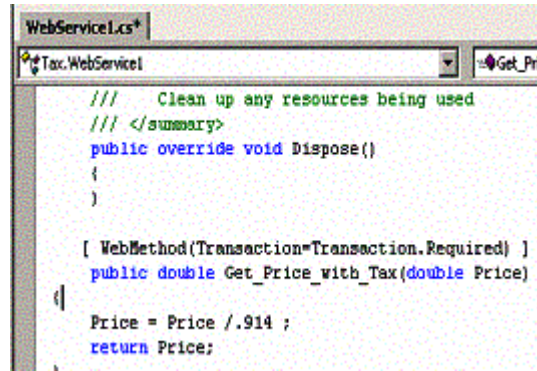
Whether implementing a number of distributed applications or simply moving components to other computers to increase the scalability of the program, it is easiest to understand the remoting system as a generic system of interprocess communication with some default implementations that easily handle most scenarios.

Enterprise Services

Using the Enterprise Services classes, developers can create "serviced components" that take advantage of COM+ services. COM+ provides a services framework for component programming models deployed in an enterprise environment. Some of these services available include just-in-time (JIT) activation, synchronization, object pooling, transactions, and shared property management. In a disconnected environment typical of Web-based applications, additional COM+ services can be used such as loosely coupled events, queued components, and role-based security.

Automatic transaction support

One of the most important COM+ features is support for transactions. It is easy to build applications that use transaction using the .NET Framework through simple keywords and settings. In fact, the .NET Framework takes advantage of the COM+ transaction infrastructure to provide both manual and automatic distributed transactions.



```

WebService1.cs+
Tax.WebService1
Get_Pri

/// Clean up any resources being used
/// </summary>
public override void Dispose()
{
}

[ WebMethod(Transaction=Transaction.Required) ]
public double Get_Price_with_Tax(double Price)
{
    Price = Price /.914 ;
    return Price;
}

```

Figure 8. Enlisting in a transaction

The beauty of the integration between the .NET Framework and COM+ transactions is that .NET Framework developers can use the high-performance COM+ transaction environment without needing to learn COM+. For example, you can simply copy your .NET Framework components that need COM+ Services such as transactions into a directory, and the .NET Framework will dynamically add the components to the COM+ transaction monitor.

Messaging

A number applications use synchronous communication through remote procedure calls, while Web-based applications rely on HTTP. But many distributed applications need the non-blocking, asynchronous communication provided by message queuing. Windows Server 2003, using a component of COM+ Services called COM+ Queued Components (previously known as Microsoft Message Queue, or MSMQ), provides applications with exactly this kind of service in. With Queued Components, an application can send messages to another application without waiting for a response (in fact, the target application might not even be running). Those messages are sent into a queue, where they are stored until a receiving application removes them. If a response is expected, the sender can check a response queue at its leisure—there's no obligation to block waiting for a message. Message queuing is a flexible, reliable approach to communication, one that's appropriate for many kinds of applications.

The **System.Messaging** namespace in the .NET Framework provides classes that allow you to work with the Queued Components. The two primary classes are the **MessageQueue** and the **Message**. The **MessageQueue** class is used to monitor and administer message queues on the network, and send, receive, or peek (examine without removing) messages. The **Message** class provides detailed control over the information you send to a queue, and is the object used when receiving or peeking messages from a queue. Besides the message body, the properties of the **Message** class include acknowledgment settings, formatter selection, identification, authentication and encryption information, timestamps, indications about using tracing, server journaling, and dead-letter queues, and transaction data.

Some of the important features of messaging using Queued Components with the .NET Framework include:

- Managed code access
- Integration with transactions
- Automatic message journaling
- Automatic notification
- Built-in data integrity, data privacy, and digital signature services
- Message priority support

- Support for multiple platforms

Service Presentation

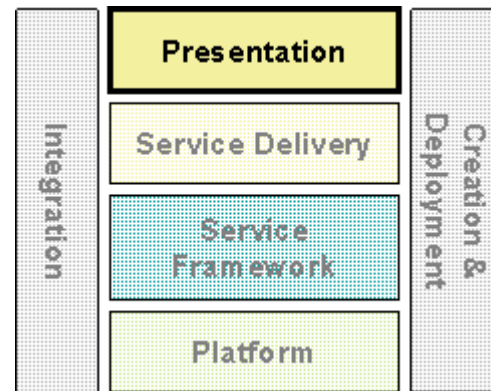


Figure 9. Service presentation

The .NET Framework enables the construction of several different application presentation models. In addition to providing support for Web-based applications that run on both traditional and device-based browsers, the .NET Framework provides substantial support for applications running on smart clients—including smart devices—that can take full advantage of their specific platform features to provide both a better user experience as well as increased functionality.

Web Applications with ASP.NET

ASP.NET is a .NET Framework component that can be used on an application server to build powerful Web applications. ASP.NET offers numerous important advantages over previous Web development models.

Web forms

The ASP.NET Web forms page framework is a scalable programming model that can be used on the server to dynamically generate Web pages. Intended as a logical evolution of ASP (ASP.NET provides syntax compatibility with existing pages), the ASP.NET Web forms framework has been specifically designed to address a number of key deficiencies in the previous model. In particular, it provides:

- The ability to create and use reusable UI controls that can encapsulate common functionality and thus reduce the amount of code that a page developer has to write.
- The ability for developers to cleanly structure their page logic in an orderly fashion (avoiding "spaghetti code").
- The ability for development tools to provide strong WYSIWYG design support for pages (existing ASP code is opaque to tools).

ASP.NET Web forms pages are text files with an .aspx file name extension. They can be deployed throughout an IIS virtual root directory tree. When a browser client requests .aspx resources, the ASP.NET runtime parses and compiles the target file into a .NET Framework class. This class can then be used to dynamically process incoming requests. (Note that the .aspx file is compiled only the first time it is accessed; the compiled type instance is then reused across multiple requests.)

An ASP.NET page can be created simply by taking an existing HTML file and changing its file name extension to .aspx (no modification of code is required).

Inside ASP.NET

At the core of ASP.NET is its HTTP runtime (different from the common language runtime), a high-performance execution engine for processing HTTP commands. The HTTP runtime is responsible for processing all incoming HTTP requests, resolving the URL of each request to an application, and then dispatching the request to the application for further processing. The HTTP runtime is multithreaded, and it processes requests asynchronously—which means it cannot be blocked by bad application code from processing new requests. Furthermore, the HTTP runtime has a resilient design that is engineered to recover automatically from access violations, memory leaks, deadlocks, and so on.

Tracing and debugging

When you are developing an application, it is often helpful to be able to insert debugging print statements into your code to output variables or structures, assert whether a condition is met, or just generally trace through the execution path of the application. ASP.NET provides two levels of tracing services that make it easy to do just that.

Page-level Tracing—At the page level, developers can use the **TraceContext** intrinsic to write custom debugging statements that appear at the end of the client output delivered to the requesting browser. ASP.NET also inserts some helpful statements regarding the start/end of lifecycle methods, like **Init**, **Render**, and **PreRender**, in addition to the inputs and outputs to a page, such as form and **QueryString** variables or headers, and important statistics about the page's execution (control hierarchy, session state, and application state). Because tracing can be explicitly enabled or disabled for a page, these statements can be left in the production code for a page with no impact to the page's performance. Each statement can be associated with a user-defined category for organizational purposes, and timing information is automatically collected by the ASP.NET runtime. The resulting output can be ordered by either time or category.

Application-level Tracing—Application-level tracing provides a view of several requests to an application's pages at once. Like page-level tracing, it also displays inputs and outputs to a page, such as form and **QueryString** variables or headers, as well as some important statistics (control hierarchy, session state, and application state). Application-level tracing is enabled through the ASP.NET configuration system, and accessed as a special mapped URL into that application (**Trace.axd**). When application tracing is enabled, page-level tracing is automatically enabled for all pages in that application (provided there is no page-level directive to explicitly disable trace).

Application configuration

A central requirement of any Web application server is a rich and flexible configuration system—one that enables developers to easily associate settings with an installable application (without having to "bake" values into code) and enables administrators to easily customize these values post-deployment. The ASP.NET configuration system has been designed to meet the needs of both of these audiences. It provides a hierarchical configuration infrastructure that enables extensible configuration data to be defined and used throughout an application, site, and/or machine. It has the following qualities that make it uniquely suited to building and maintaining Web applications:

- ASP.NET allows configuration settings to be stored together with static content, dynamic pages, and business objects within a single application directory hierarchy. A user or administrator simply needs to copy a single directory tree to set up an ASP.NET framework application on a machine.
- Configuration data is stored in plain XML text files that are both human-readable and human-writable by administrators and developers and can be accessed using any standard text editor, XML parser, or scripting language.

- ASP.NET provides an extensible configuration infrastructure that enables third-party developers to store their own configuration settings and participate in their processing
- Changes to ASP.NET configuration files are automatically detected by the system and are applied without requiring any user intervention.
- Configuration sections can be locked down and can be prevented from being overridden.

Updating applications

ASP.NET uses the Microsoft .NET Framework deployment technologies, thus gaining benefits such as XCOPY deployment and side-by-side deployment of applications.

Another major benefit of ASP.NET is support for live updating of applications. An administrator does not need to shut down the Web server or even the application to update application files: Application files are never locked, so they can be overwritten even when the application is running. When files are updated, the system gracefully switches over to the new version.

Extensibility

Within an ASP.NET application, HTTP requests are routed through a pipeline of HTTP modules, ultimately to a request handler. HTTP modules and request handlers are simply managed .NET classes that implement specific interfaces defined by ASP.NET. This modular architecture makes it very easy to add services to applications: Just supply an HTTP module. For example, security, state management, and tracing are implemented as HTTP modules by ASP.NET. Higher-level programming models, such as Web services and Web forms, are also implemented as request handlers. An application can be associated with multiple request handlers—one per URL—but all HTTP requests in a given application are routed through the same HTTP modules.

State management

The Web is a fundamentally stateless model with no correlation between HTTP requests. This can make writing Web applications difficult, since applications usually need to maintain state across multiple requests. ASP.NET enhances the state management services introduced by ASP to provide three types of state to Web applications: application, session, and user. ASP.NET session state is stored in a separate process and can even be configured to be stored on a separate machine or persisted to a Microsoft SQL Server™ database. This makes session state scalable even when an application is deployed across the largest Web farms.

User state resembles session state, but generally does not time out and is persisted. Thus user state is useful for storing user preferences and other personalization information. All the state management services are implemented as HTTP modules, so they can be added, extended, or even removed from an application's pipeline easily. If additional state management services are required beyond those supplied by ASP.NET, they can be provided by a third-party module.

Caching

The ASP.NET programming model provides a cache API that enables programmers to activate caching services (on enterprise software) to improve performance. An output cache saves completely rendered pages, and a fragment cache stores partial pages. Classes are provided so that applications, HTTP modules, and request handlers can store arbitrary objects in the cache as needed.

Aggressive caching capabilities will be provided as part of ASP.NET. ASP.NET is designed to provide a robust Web application environment capable of running mission-

critical projects for long periods of time.

ASP.NET is delivering some significant improvements in Web application performance—a throughput improvement of up to three times over existing ASP-based applications, and even more dramatic productivity improvements.

See the Microsoft guide [Caching Architecture Guide for .NET Framework Applications](#) for an in-depth analysis of .NET applications caching.

Security

An important part of many Web applications is the ability to identify users and control access to resources. The act of determining the identity of the requesting entity is known as authentication. Generally, the user must present credentials, such as a name/password pair in order to be authenticated. Once an authenticated identity is available, it must be determined whether that identity is permitted to access a given resource. This process is known as authorization. ASP.NET works in conjunction with IIS to provide authentication and authorization services to applications.

An important feature of COM objects is the ability to control the identity under which COM object code is executed. When a COM object executes code with the identity of the requesting entity, this is known as impersonation. ASP.NET Framework applications can optionally choose to impersonate requests.

Some applications also want to be able to dynamically tailor content, based on the requesting identity or based on a set of roles that a requesting identity belongs to. ASP.NET Framework applications can dynamically check whether the current requesting identity participates in a particular role. For example, an application might want to check to see whether the current user belongs to the manager's role, in order to conditionally generate content for managers.

Web Application for Mobile Devices

ASP.NET mobile controls (previous known as the Microsoft Mobile Internet Toolkit, or "MMIT") let you easily target cell phones and PDAs (over 80 mobile Web devices) using ASP.NET. You write your application just once, and the mobile controls automatically generate WAP/WML, HTML, or iMode as required by the requesting device.

Over the past few years, the world has seen an explosion of new wireless devices, such as cell phones, pagers, and personal digital assistants (PDAs), which enable users to browse Web sites at any time from any location. Developing applications for these devices is challenging:

- Different markup languages are necessary, including HTML for PDAs, wireless markup language (WML) for wireless application protocol (WAP) cell phones, and compact HTML (cHTML) for Japanese i-mode phones.
- Devices have different form factors. For example, devices have varying numbers of display lines, horizontal or vertical screen orientation, and color or black and white displays.
- Devices have different network connectivity, ranging from 9.6 KB cellular connections to 11 MB Wireless LANs.
- Devices have different capabilities. Some devices can display images, some can make phone calls, and some can receive notification messages.



Figure 10. Cell phone with a WAP browser

The ASP.NET mobile controls address these challenges by isolating them from the details of wireless development. Thus, developers can quickly and easily build a single, mobile Web application that delivers appropriate markup for a wide variety of mobile devices.

Mobile Web Forms Controls—The mobile Web Forms controls are ASP.NET server-side controls that provide user interface elements such as list, command, call, calendar, and so on. At execution time, the mobile controls generate the correct markup for the device that makes the request. As a result, you can write a mobile application once and access it from multiple devices.

Because these mobile controls are based on the ASP.NET controls, you can leverage your current desktop development skill set when creating mobile applications. You can also reuse the same business logic and data access code that you use in your desktop application. Mobile and desktop Web forms can reside in the same Visual Studio .NET project. This makes an application faster to develop and lowers your maintenance cost.

The ASP.NET mobile controls also enable you to customize the markup that is generated by mobile controls for a specific device. You can designate templates and styles for a specific device within the mobile page.

Smart Client Applications

Software developers targeting the corporate environment have faced a difficult tradeoff when deciding between the browser-based, thin client application model and its rich client counterpart.

The browser-based application is easy to install and maintain, can target many desktops, and has no impact on the state of the client computer. Yet in spite of these advantages, the browser-based model is far from perfect. Rich client applications provide a richer user interface (UI) along with access to the local disk and local

application programming interfaces (APIs), further empowering the developer and resulting in a more productive user experience. Because they run locally on the client computer, rich client applications also make for more efficient use of available resources, eliminate the issue of network latency, and enable the user to work offline. And because there is a similar programming model for the .NET Compact Framework, developers can easily transfer their skills to developing applications for smart devices.

Overall, relative to the smart client, the browser-based model is wonderful for information technology (IT) administrators, but leaves much to be desired for both developers and end users. The Microsoft .NET Framework programming model serves the interests of all three parties. Its smart client application model combines all the power and flexibility of the rich client model with the ease of deployment and stability of the browser-based model.

We will now provide an overview of the .NET Framework's smart client application model, explaining how it works from a high-level point of view, and elaborating on how this style of application compares with the browser-based model.

No more versioning conflicts

As mentioned previously, components built using the .NET Framework are not subject to versioning conflicts. By default, .NET Framework applications are self-contained and isolated, resolving assemblies (the .NET version of components) from the local application directory rather than from a global or shared location. With this approach, multiple versions of the same assembly can coexist on the same system without conflict.

It remains possible to share assemblies, using a central repository known as the global assembly cache. Each assembly registered here is assigned a strong internal name, derived from its file name, version, culture (such as English, German, or Japanese), digital signature, and public key. Thus, each shared assembly is uniquely identifiable. This allows multiple versions of a given assembly to coexist in the global assembly cache and even run concurrently without conflict, a scenario known as side-by-side execution.

Smart client versus browser-based

With these smart client technologies, it's easy to see how the platform-specific application model reflects a practical approach to software development. But how does it compare with the browser-based model? Consider the following:

Work Offline—One obvious but critical advantage that smart client applications have over browser-based applications is the capability to work offline. Practical Internet access is still anything but ubiquitous. According to IDC, at the end of 2002, approximately 38.9 percent of U.S. households will engage in some form of home-office activity. At that time, according to The Yankee Group, only 16 percent of those households are expected to have any form of high-speed connectivity to the Internet. This means that, assuming a remote worker has access to the Internet (not a given for those traveling and seeking to work from hotels, airports, or airplanes), chances are he or she will be forced to endure an irritatingly slow dial-up connection every time an attempt is made to access or manipulate data using a browser-based application. Even if high-speed access is available, what happens when the server goes down? With smart client applications, these problems can be avoided. Once the necessary assemblies are downloaded to the local disk, the user can work productively offline.

Use Resources Efficiently and Safely—Web-based applications are often designed so that all processing is handled by the servers. The advantage of placing the entire burden on the server is that it makes it easier to push an application to multiple

different types of clients. The disadvantage is that, for even the simplest request, the client-side user must endure both network latency and the amount of time it takes the server to queue up the given request and process it.

Performing some or all of the work locally with smart client applications means quicker response time and reduced network and server workload, making for more scalable infrastructure. A smart client application can run without any interaction with the network and the server, completely freeing up that infrastructure, or it can go to the server as needed to dynamically download an assembly or to invoke a Web service. The fact that even some of the work will be done locally—loading and implementing the UI controls, for example—results in a more dynamic user experience and much more efficient use of the infrastructure available, both client and networked.

Web-based applications can be designed to offload some of the processing to the client using scripting or ActiveX controls, but both have limitations when compared with the smart client application model. By default, Web-based script essentially runs in a "sandbox" and is prevented from accessing local resources. Scripted applications cannot read from or write to a client user's local disks, so all data must be stored on a remote server. Again, this causes problems when the user is faced with a slow Internet connection or wishes to work offline. Scripted applications are also unable interact with applications housed on the client computer, such as Microsoft Excel or the Microsoft Outlook® messaging and collaboration client. They can certainly offload some of the work from the server, but they do not have the crucial flexibility of smart client applications when it comes to interacting with the local computer. Furthermore, they are incredibly difficult and expensive to develop well. Dynamic Hypertext Markup Language (DHTML) applications can replicate some of the look and feel of a Microsoft Windows®-based application.

Smart client applications built on the .NET Framework have the best of all possible worlds. The Windows Forms libraries enable quick and easy Windows-style UI development that can be implemented either by hand-coding or by dragging and dropping controls onto a form in Visual Studio .NET. Smart client applications can access the local system, but only to the extent that they are allowed by the .NET Framework's evidence-based security policies, as described above.

Security

Again, security is no longer an all-or-nothing security decision but a fine-grained determination that can permit some operations but forbid others. The burden of this decision is taken off the user and handled transparently by the CLR, using either the default security settings or those configured by the system administrator.

Access Local APIs

Smart client applications can leverage powerful Windows-based APIs that are unavailable to non-ActiveX browser-based applications. For example, smart client applications can use the graphical power of the GDI+ libraries. Smart client applications can tie into the Microsoft Office APIs, enabling developers to manipulate Microsoft Word, Microsoft Access, Excel, Outlook, and Microsoft PowerPoint® through their own applications. Smart client applications also have the ability to listen for incoming network requests, thus enabling peer-to-peer connectivity. This example, in particular, deserves a section all its own.

Use peer-to-peer technology

In a peer-to-peer scenario, a computer can act as both client and server, making requests of other peer computers and responding to requests placed on it by these peers. One example of this is a file-sharing system, in which an application running on

one computer can search for and retrieve a desired file from a peer computer while making its own files available in reciprocation. Another example is an application running on one computer that can seek out those computers on the network with spare processor power and disk space and harness these unused resources for its own use, enabling a corporation to more fully leverage the infrastructure available to it.

Peer-to-peer technology represents a powerful paradigm with tremendous promise, but it can only be implemented using smart client applications. Web browsers, being user-driven by design, do not have the necessary capacity to listen for and respond to spontaneous requests made by other clients on the network.

Consume Web services

Just like browser-based applications, with just a few lines of code, .NET Framework smart client applications can consume functionality provided by Web services. Web services provide a model for distributed computing based on standard, platform-neutral protocols such as SOAP, XML, and HTTP. Web services allow applications to interact regardless of what underlying platforms they may be running on and provide a perfect way to integrate new applications with legacy code.

When to implement smart clients

The smart client design is not ideal for every scenario. In situations such as e-commerce, where user platforms are unknown or diverse, the browser-based model continues to represent the most practical approach. However, in the context of a corporate computing environment, where clients are known to be running the Windows operating system, the smart client model is the design of choice, combining the power and flexibility of rich client applications with the stability and ease of deployment associated with browser-based applications.

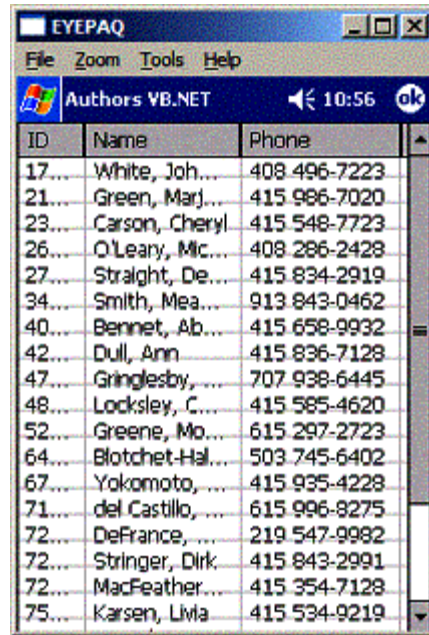
Smart Devices

The .NET Framework has built-in support for two different forms of mobile device applications: Those that are capable of hosting a compact version of the .NET Framework and those that support a Web browser for accessing Web sites that can provide content tailored for devices.

.NET Compact Framework

The Microsoft .NET Compact Framework is a version of the .NET Framework for rapidly building and securely deploying and running distributed Web services and applications on smart devices, such as cellular telephones, enhanced televisions, and personal digital assistants (PDAs). The .NET Compact Framework supports a large subset of the .NET Framework class libraries and common language runtime.

Supported devices include the Pocket PC 2000, Pocket PC 2002, Pocket PC 2002 Phone Edition, and custom-designed embedded devices built with the Windows CE .NET 4.1 operating system. Earlier versions of Windows CE .NET are not supported.



ID	Name	Phone
17...	White, Joh...	408 496-7223
21...	Green, Marj...	415 986-7020
23...	Carson, Cheryl	415 548-7723
26...	O'Leary, Mic...	408 286-2428
27...	Straight, De...	415 834-2919
34...	Smith, Mea...	913 843-0462
40...	Bennet, Ab...	415 658-9932
42...	Dull, Ann	415 836-7128
47...	Gringlesby, ...	707 938-6445
48...	Locksley, C...	415 585-4620
52...	Greene, Mo...	615 297-2723
64...	Blotchet-Hal...	503 745-6402
67...	Yokomoto, ...	415 935-4228
71...	del Castillo, ...	615 996-8275
72...	DeFrance, ...	219 547-9982
72...	Stringer, Dirk	415 843-2991
72...	MacFeather...	415 354-7128
75...	Karsen, Livla	415 534-9219

Figure 11. .NET Compact Framework application on a smart device

The .NET Compact Framework provides the following key features:

- A compact common language runtime that brings the benefits of managed code, such as memory management, code reliability, and programming language neutrality, to devices
- Consistency with desktop and server programming models
- Seamless connection with Web services
- Rich enterprise-class data access features with XML classes and ADO.NET
- Classes to program applications that access data using Microsoft SQL Server 2000 Windows CE Edition 2.0
- Full access to native platform features through platform invoke
- Just-in-time (JIT) compilation for optimal performance

The Smart Device Projects for Visual Studio .NET are used to develop applications that target the .NET Compact Framework. Smart Device Projects enhance Visual Basic .NET and Visual C# .NET with device-specific project types and a form designer to implement .NET Compact Framework Windows Forms controls. You can debug and deploy directly to a device or to Pocket PC and Windows CE .NET emulators.

Microsoft Office

Microsoft Office XP is a widely used tool for knowledge workers (such as financial analysts and production planners) to use for productivity applications that help with organizing and analyzing information. Workflow applications—based on the routing of documents throughout the enterprise—can be constructed that leverage the unique capabilities of the Microsoft Office XP suite.

Using the Microsoft Office XP Web Services Toolkit 2.0, it is possible for developers to integrate the power of Web services with custom Office XP applications.

Visual Studio tools for Office

"Visual Studio Tools for Office" is the code name for a new technology that will bring the power and productivity of Visual Studio .NET and the Microsoft .NET Framework to

business solutions built on the next versions of Microsoft Word and Microsoft Excel. With this technology, developers using Visual Studio .NET 2003 can use Visual Basic .NET and C# to write code behind Word- and Excel-based applications. "Visual Studio Tools for Office" provides developers with full access to the Word and Excel object models, as well as the benefit of Microsoft IntelliSense® statement completion when developers write code against these object models.

Microsoft Office System

The Microsoft Office System helps enable developers to create intelligent business solutions that address today's demanding business requirements while giving information workers a powerful user interface. Using the support for customer-defined XML schemas and Web services in the Microsoft Office System, developers can more easily build documents and applications that connect with business processes and data. In addition, new tools in the Microsoft Office System help you build managed code for the Microsoft .NET Framework and take advantage of the ease and security of deploying solutions from a server.

.NET Development

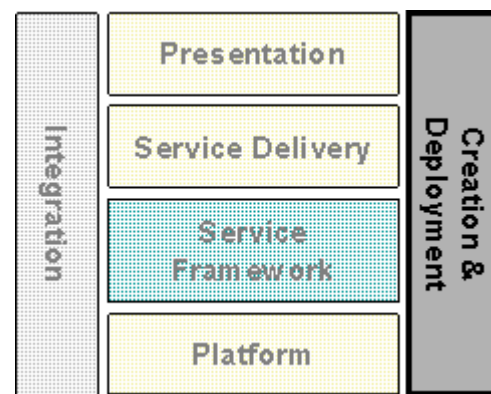


Figure 12. Creation and deployment

As has been discussed elsewhere in this document, a platform and a framework are required to support the creation of an enterprise-scale service oriented architecture. At the same time, it is important to have a comprehensive set of tools to support the design and development phases of the life cycle of these systems.

Design and development are typically accomplished by a diverse team of specialists. These specialists require integrated tools that enable application modeling, rapid application development (RAD), and integrated database support.

Data-driven development

With the .NET Framework, Microsoft provides ADO.NET, a set of intuitive interfaces that enable developers to manage and interact with a variety of data sources. Tools that take advantage of this rich database functionality should provide support for data-driven development, including visual data designers that streamline the development process.

Working with servers

In a service-oriented architecture, the majority of development typically occurs on servers. For this reason, it is important to have visual design tools for the creation, testing, and deployment of server-based solutions. These tools make it easy to integrate platform and framework features—such as message queues, remote servers, and Windows services—into complete server solutions. And these tools need to be able

to support multiple common development languages.

Team development

Source control is a significant requirement in team-based development. It allows an organization to protect and manage their valuable code assets. Developers must be able to perform all source control operations without ever leaving their tool suite. Many features trigger automatically, such as when a file is about to be changed, providing a safety net for team members and ensuring the protection of the project.

Creating Web Services

As discussed above, Microsoft is making Web-based development easier and more reliable than ever before. With the .NET Framework, developers can make an object Internet-accessible simply by marking it with the **WebMethod** keyword.

The following Visual Basic code shows how to create a Web service that exposes a function to Web requests. As we mentioned previously, this functionality can be found in the **System.Web.Services** namespace, which we first import into our project. This then allows us to specify both the **WebService** attribute on the class (which is inherited from **System.Web.Services.WebService**) as well as the **WebMethod** attribute on the appropriate function (in this example **CalcTemp**, for converting degrees Fahrenheit into Celsius):

```
Imports System.Web.Services
<WebService(Namespace := _
    "http://tempuri.org/Test/Service1")> _
    Public Class Service1
    Inherits System.Web.Services.WebService
    ...
    <WebMethod(Description:="Converts F to C")> _
        Public Function CalcTemp(ByVal dF As Double)_
            As Double
            CalcTemp = ((dF - 32) * 5) / 9
        End Function
    End Class
```

Testing Web Services

Developers can use the WSDL.exe utility to discover how to call a Web service. This utility takes the path to a WSDL file and a language (C#, Visual Basic, JScript, or Visual J#) as input arguments and then generates a single source file as output. This file contains the source code for a proxy class that exposes methods for each of the corresponding methods exposed by the Web service. Each proxy method contains the appropriate network invocation and marshaling code necessary to invoke and receive a response from the remote service. Visual Studio .NET makes this even easier by providing an explorer interface to Web services. Running this utility against our simple Web service creates a test page for developers to use.

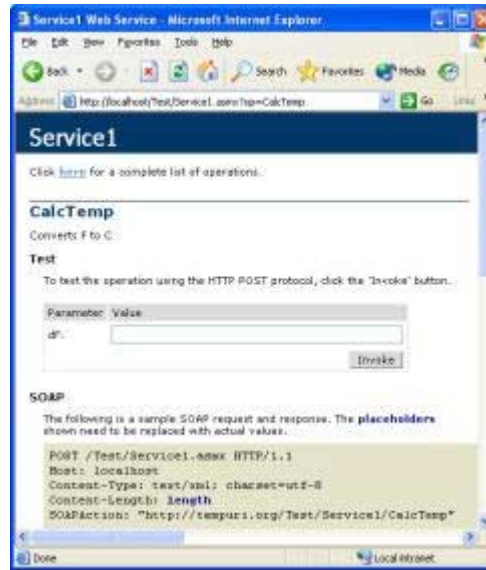


Figure 13. Auto-generated Web service test client

This test page contains test entry points for all of the service's members given the **WebMethod** attribute as well as additional documentation about working with Web services including sample data for SOAP, HTTP GET, and HTTP POST requests and responses and pointers to online reference material. Invoking the test page for a specific method generates an XML-formatted response.

Using Web Services

Accessing a Web service from managed code is also a straightforward process. First, a Web reference is added to the project for the Web service to be accessed. The Web reference creates a proxy class with methods that serve as proxies for each exposed method of the Web service. Next, the namespace for the Web reference is added to the project. Finally, an instance of the proxy class is created and then the methods of that class are accessed as with the methods of any other class.

By default, adding a Web reference also adds methods to the proxy class for accessing the Web service asynchronously.

To access a Web service in managed code a developer simply needs to:

- Create the application that needs to access a Web service. This application could even be another Web service.
- Add a Web reference for the Web service.
- Create an instance of the proxy object in the client code where the Web service needs to be accessed.
- Access the methods of the Web service as with any other component.

An important aspect of Web service programming is that this code can actually run anywhere in an application: it could be part of an ASP.NET application, part of a smart client application, or even called from some other form of server-based application that has Web access to the Web service.

Visual Studio .NET 2003

Microsoft Visual Studio .NET 2003 provides a comprehensive set of development tools for building all types of applications using the .NET Framework. In addition to providing an integrated design, development, test, and debugging environment with such powerful features as IntelliSense for code development and designers for Web,

Windows, and device applications, Visual Studio features numerous advanced capabilities that will be of interest to enterprise application developers.

Integrated tools for data

Visual Database Tools—Visual Studio .NET maximizes the productivity of developers working with databases. Instead of requiring multiple external tools for creating database schemas, stored procedures, indexes, triggers, and other items, developers can perform all of these tasks within the Visual Studio .NET IDE:

The Visual Studio .NET integrated Database Designer provides Oracle, SQL Server, and other database users a visual view of their schema and lets them directly add, modify, or remove tables, columns, indexes, views, stored procedures, and other database objects. In addition, relationships between tables can be viewed and modified, providing complete control over the physical database design.

The Query Designer enables developers to visually create complex SQL queries and directly edit the corresponding SQL script. The results from the query can be viewed to verify accuracy, making it much faster for developers to work with data.

The Script Editor enables programmers to work with stored procedures, triggers, or any SQL script. Color-coded syntax makes it easy to view SQL keywords while the Query Designer can be invoked for visually designing a code block by right-clicking a Select statement.

Seamless stored procedure debugging for developers using Microsoft SQL Server version 6.5 or higher makes it easier to diagnose errors in database code.

RAD for the server

To simplify multiple-tier development, Visual Studio .NET extends the RAD principles of visual design to the creation, testing, and deployment of server-based solutions. So-called "RAD for the Server," these features enable programmers to drag and drop visual representations of server components, such as message queues, remote servers, Windows services, and more, onto a design surface. Once placed on the design surface, code can be written in any language to connect the server applications together.

Server Explorer—One of the biggest challenges in writing a middle-tier component is simply discovering what application services are available on the corporate network, and then integrating these services into an application. In Visual Studio .NET, the Server Explorer displays all available server-side resources on a given computer including databases, message queues, event logs, Windows services, and performance counters.

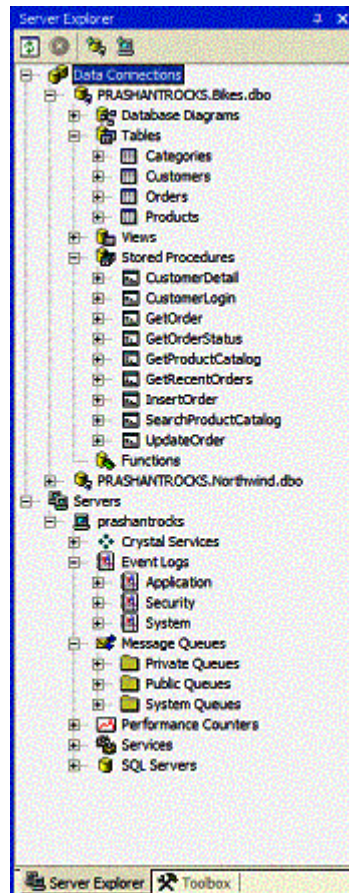


Figure 14. The Server Explorer provides easy access to server-based resources.

Component Designer—Together with the Server Explorer, the Component Designer provides a design surface on which developers can build new middle-tier business logic components. In the same way that forms designers enable rapid creation of client applications, the Component Designer provides the means for graphically constructing server-side components. Rather than manually writing low-level server-based code, developers simply drag and drop server components to a design surface, write event-handling code to integrate the server functionality, and then use Visual Studio .NET to deploy and execute the component on the server.

You can find out more about Visual Studio .NET on the MSDN site in the [.NET Development](#) area.

Enterprise lifecycle support

Microsoft offers enterprise versions of Visual Studio .NET which incorporate an integrated set of technologies for building complex, enterprise-scale applications. These features are delivered in the Enterprise Architect and Enterprise Developer versions of Visual Studio .NET.

Visual Studio .NET Enterprise Architect edition provides a full range of design and modeling capabilities with the included and integrated Microsoft Visio® toolset. Through the use of industry standard modeling methodology, Unified Modeling Language (UML), Visual Studio .NET helps senior developers and architects visually design applications and requirements. Visual Studio .NET Enterprise Architect combines conceptual and physical database modeling using Visio to deliver powerful

Object Role Modeling (ORM) functionality to architects.

Visual Studio .NET helps software architects and senior developers who lead large development teams better communicate and enforce their design decisions. Enterprise Templates jump-start the process of creating new applications, and allow architects to limit the options available from within Visual Studio .NET using XML-based Template Description Language (TDL) to define development policies.

Instrumentation allows software or hardware to publish or be queried for relevant information during its execution. Visual Studio .NET 2003 includes the new Enterprise Instrumentation Framework (EIF), providing unified tracing and eventing services for enterprise applications. The framework allows decoupled and distributed enterprise applications to be monitored and diagnosed by Microsoft and third-party tools for overall application health, faults, or other internal conditions.

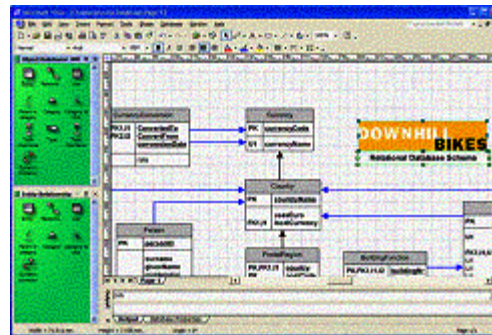


Figure 15. Integrated Visio modeling with Visual Studio .NET Enterprise Architect

Team development with Visual Studio .NET

Visual Studio .NET reduces the complexity of source control by making it an extension of project file management. Developers may perform all source control operations without ever leaving the IDE or opening another application. The Microsoft Visual SourceSafe® version control system also enables development teams to automatically protect and track their most valuable source code, documentation, binaries, and all other file types as they change throughout the software life cycle.

Other .NET Development Tools

A large number of tools are available to support .NET application development. These tools are provided both by Microsoft and by third party tool vendors, some of which are available at no charge.

Developers may wish to start with the .NET Framework Software Development Kit (SDK), which is free. This SDK comes with comprehensive documentation on the Framework, including all of the namespaces and types within. The SDK also includes several hundred samples—in Visual Basic, C#, and often JScript or C++—that illustrate how to work with the major features of the Framework. There are even several complete tutorials to help developers get up to speed on writing and debugging programs, using resources, and packaging and deployment. Finally, the SDK also includes a large number of tools for debugging, managing, and securing .NET Framework programs. Since several programming language compilers are included with the .NET Framework distribution, this might be all you need to get started. The .NET Framework SDK can be found at the [.NET Framework Home page](#).

However, most developers will want to make use of an integrated development environment (IDE) tailored specifically for writing .NET Framework applications. The

[ASP.NET Web Matrix](#) is primarily targeted at developing Web applications and is a small (1.2 megabytes), free, community-supported tool for building applications with the ASP.NET class libraries. It features a page designer, integrated database management, support for Web services, mobile applications, and even a personal Web server for development and testing.

In addition to Microsoft's tools for developing with the .NET Framework, other application tools vendors have announced their support for this new programming model. Developers wishing to program with some of the languages not implemented directly by Microsoft may purchase the compilers from vendors and integrate those compilers into the Visual Studio .NET shell (you can find third-party Visual Studio .NET tools at the [Visual Studio Industry Partners](#) Web site). Ultimately, you can use the same IDE for developing with each of the programming languages supported by the .NET Framework.

Summary

This document describes the technical basis for building enterprise-scale Web-based applications using the Microsoft .NET Framework and Windows Server 2003. This new generation of applications is needed to meet the demands of enterprise computing over an Internet standard distributed network infrastructure utilizing a highly flexible, standards-based, powerful software infrastructure for integrating existing investments with next-generation applications and services.

The .NET Framework is an integral component of the Windows Server platform, the end-to-end Internet platform built on the Windows operating system for rapidly building and deploying enterprise applications. These applications include Web services and Web applications that integrate customers, businesses, and applications. The .NET Framework enables developers to rapidly create Web services and Web applications through the use of developer productivity features, such as multiple-language support, adherence to public Internet standards, and the use of a loosely coupled, scalable architecture. Microsoft is delivering a comprehensive set of application services fully integrated into the Windows operating systems and available through the .NET Framework.

Glossary

Term	Definition
.NET Framework	The .NET Framework is a component of the Windows operating system that provides the programming model for building, deploying and running Web-based applications, smart client applications and Web services. The .NET Framework consists of the common language runtime (CLR) and a unified class library.
Active Directory	The Windows directory service that provides a unified, hierarchical view of complex networks.
ADO.NET	The suite of data access technologies included in the .NET Framework class libraries
ASP.NET	The development component for building server-based Web applications. An evolution of ASP into the .NET Framework.
assembly	The primary building block—also the unit of deployment and versioning—of a .NET Framework application. An assembly includes an assembly manifest, which describes the contents of the assembly
C#	A new ECMA-approved programming language designed for the .NET Framework. C#, which is an evolution of C and C++, is type safe and object oriented. Because it is compiled as managed code, it benefits from the services of the common language runtime, such as language interoperability, enhanced security, and garbage collection.
class library, .NET	A library of classes, interfaces, and value types that are included

Framework	in the Microsoft .NET Framework and can be used from any CLS-compliant language. The .NET Framework class library provides access to system functionality and is designed to be the foundation on which .NET Framework applications, components, and controls are built.
common language runtime (CLR)	The engine at the core of .NET Framework-managed code execution. The runtime supplies managed code with services such as cross-language integration, code access security, object lifetime management, and debugging and profiling support.
Common Language Specification (CLS)	A subset of .NET Framework features that are supported by a broad set of compliant languages and tools. CLS-compliant languages and tools are guaranteed to interoperate with other CLS-compliant languages and tools.
ECMA	A European standards body created in 1961. Internationally accredited ECMA has fast-track approval for ISO and is the forum for successful standards such as ECMAScript.
evidence-based security	The .NET Framework introduces the concept of <i>evidence-based security</i> , referring to inputs to the security policy about code—such as from what site, security zone, or URL was an assembly obtained, what is its strong name, and whether it has a digital signature and from whom. Based on these and other answers—which can come from multiple sources depending on where the code is run—the appropriate security policy can be applied, and the appropriate permissions may be granted to the assembly..
Extensible Markup Language (XML)	A subset of Standard Generalized Markup Language (SGML) that is optimized for delivery over the Web. XML provides a uniform method for describing and exchanging structured data that is independent of applications or vendors.
garbage collection (GC)	The process of transitively tracing through all pointers to actively used objects to locate all objects that can be referenced and then arranging to reuse any heap memory that was not found during this trace. The CLR garbage collector also compacts the memory that is in use to reduce the working space needed for the heap.
HTTP	Hyper Text Transfer Protocol is a standard Internet protocol for transfer of information between servers and between clients and servers.
IDL	Interface Definition Language. A language used by applications to specify the various interfaces they intend to offer to other applications.
Microsoft intermediate language (MSIL)	A language used as the output of a number of compilers and as the input to a just-in-time (JIT) compiler, which produces native code. MSIL defines an abstract, stack-based execution model.
JIT	An acronym for "just-in-time", a phrase that describes an action that is taken only when it becomes necessary, such as just-in-time compilation or just-in-time object activation.
loosely coupled architecture	A distributed application in which you can change the implementation of one tier without affecting any of the other tiers. Contrast tightly coupled architecture.
managed code	Managed code supplies the metadata necessary for the CLR to provide services, such as memory management, cross-language integration, code access security, and automatic lifetime control of objects. All code based on MSIL executes as managed code.
manifest	An integral part of every assembly that renders the assembly self-describing via metadata. The metadata describes which modules and resource files are part of a particular assembly, which types are exported, and which other assemblies are referenced. It also specifies which security permissions are required to run, what additional permissions are optionally requested, and what permissions the assembly refuses.
metadata	Data (or information) about data. Many different systems use metadata—for example, type libraries in COM provide metadata and databases have schemas. In the CLR, metadata is used to describe assemblies and types. It is stored with them in the executable files, and is used by compilers, tools, and the runtime

	to provide a wide range of services. Metadata is essential for runtime type information and dynamic method invocation.
native code	Code that has been compiled to processor-specific machine code.
<i>n</i> -tier	System architecture that separates presentation, business logic, data access, and database (or other persistence mechanism) tiers.
reflection	.NET Framework technology that allows you to examine metadata that describes types and their members. Reflection can be used to create, invoke, and access type instances at run time.
serviced component	The mechanism that enables COM+ services to be available to .NET Framework classes.
side-by-side execution	The ability to run multiple versions of the same assembly simultaneously. This can be on the same computer or in the same process or application domain. Allowing assemblies to run side-by-side is essential to support robust versioning in the common language runtime. Side-by-side is also used to describe to describe two versions of the .NET Framework running simultaneously on the same computer.
SOAP	Simple Object Access Protocol, a W3C standard. A lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML-based protocol for exchanging structured and type information on the Web. The SOAP protocol contains no application or transport semantics, which makes it highly modular and extensible.
tightly coupled architecture	A distributed application where a change to any tier affects some or all the other remaining tiers. Contrast loosely coupled architecture.
UDDI	Universal Description, Discovery, and Integration (UDDI) specification. An initiative that creates a global, platform-independent, open framework to enable Web service providers to advertise the existence of their Web services and for Web service consumers to locate Web services of interest.
unmanaged code	Code that was created without knowledge for the conventions and requirements of the .NET Framework. Unmanaged code executes in the .NET Framework environment with minimal services (for example, no garbage collection, limited debugging, and no declarative security).
Web forms	The ASP.NET page framework, which supports server-side controls that render HTML user interface on Web browsers.
Web services	A programming model that provides the ability to exchange messages in a scalable, loosely coupled, and platform-neutral environment using standard protocols such as HTTP, XML, XSD, SOAP, and WSDL. The SOAP-based XML messages exchanged between a Web service and its clients can be structured and typed, or loosely defined. The flexibility of using a text format such as XML enables the message exchange to evolve over time in a loosely coupled way. Because they are based on standard protocols and are platform neutral, Web services enable communication with a broad variety of implementations, platforms, and devices.
Web Services Description Language (WSDL)	An XML-based contract language for describing network services offered by a server.
Windows Forms	A rich Windows client library that encapsulates native Win32 APIs and exposes secure, managed classes for creating smart Windows client applications. The Windows Forms class library provides many controls, such as buttons, check boxes, drop-down lists, combo boxes, data grid, and others, that encapsulate user-interface and other client-side functionality.
Windows Management Instrumentation (WMI)	A component of the Windows operating system that provides management information and control in an enterprise environment using industry-wide standards.

WSDL	(see Web Services Description Language)
XML	(see Extensible Markup Language).
XML Schema Definition (XSD)	A W3C Recommendation that specifies how to formally describe the elements of an XML document. The schema can be used to verify the conformance of elements in an XML document.





For More Information

[The .NET Architecture Center](#)

For the latest information on .NET, please see the [Microsoft .NET](#) home page.

Developer information on the .NET Framework can be found at the [.NET Framework](#) home page.

Design patterns and guidance for developing and administrating .NET can be found at [patterns and practices](#).

 Print
  E-Mail
  Discuss
  Add to Favorites

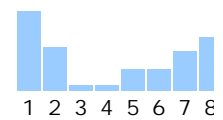
How would you rate the quality of this content?

1 2 3 4 5 6 7 8 9
 Poor Outstanding

Tell us why you rated the content this way. (optional)

Submit

Average rating:
5 out of 9



108 people have r

[Contact Us](#) | [E-mail this Page](#) | [MSDN Flash Newsletter](#) | [Legal](#)

©2003 Microsoft Corporation. All rights reserved. [Terms of Use](#) | [Privacy Statement](#)