

Windows Server System Products

The Windows Server System accelerates the integration of systems, applications, and partners using Web services through the servers' deep level support of XML. This deep support of XML allows enterprises to build on legacy systems rather than ripping and replacing them. For example, Microsoft Host Integration Server provides simple access to mainframes and Microsoft BizTalk® Server offers automatic conversions of existing data formats into XML.

The Windows Server System includes:

- **Microsoft Application Center 2000** to deploy and manage highly available and scalable Web applications.
- **Microsoft BizTalk Server 2002** to build XML-based business processes across applications and organizations.
- **Microsoft Commerce Server 2002** for quickly building scalable e-commerce solutions.
- **Microsoft Content Management Server 2002** to manage content for dynamic e-business Web sites.
- **Microsoft Exchange Server 2000** to enable messaging and collaboration anytime, anywhere.
- **Microsoft Host Integration Server 2000** for bridging to data and applications on mainframe legacy systems.
- **Microsoft Internet Security and Acceleration Server 2000 (ISA Server)** for SSL-secured, fast Internet connectivity.
- **Microsoft Mobile Information Server 2002** to enable application support by handheld devices, such as mobile phones.
- **Microsoft Operations Manager 2000** to deliver enterprise-class solutions for operations management.
- **Microsoft Project Server 2002** provides an extensible technology platform to develop and deploy best practices for project management across an organization.
- **Microsoft SharePoint™ Portal Server 2001** to find, share, and publish business information.
- **Microsoft SQL Server™ 2000** to store, retrieve, and analyze structured XML data.

Conceptual Questions

What is the .NET Framework?

The Microsoft .NET Framework is a platform for building, deploying, and running Web Services and applications. It provides a highly productive, standards-based, multi-language environment for integrating existing investments with next-generation applications and services as well as the agility to solve the challenges of deployment and operation of Internet-scale applications. The .NET Framework consists of three main parts: the common language runtime, a hierarchical set of unified class libraries, and a componentized version of Active Server Pages called ASP.NET.

Runtime Technical Questions

Terminology

What is the common language runtime (CLR)?

The common language runtime is the execution engine for .NET Framework applications.

It provides a number of services, including the following:

- Code management (loading and execution)
- Application memory isolation
- Verification of type safety
- Conversion of IL to native code
- Access to metadata (enhanced type information)
- Managing memory for managed objects
- Enforcement of code access security
- Exception handling, including cross-language exceptions
- Interoperation between managed code, COM objects, and pre-existing DLLs (unmanaged code and data)
- Automation of object layout
- Support for developer services (profiling, debugging, and so on)

What is the common type system (CTS)?

The common type system is a rich type system, built into the common language runtime, that supports the types and operations found in most programming languages. The common type system supports the complete implementation of a wide range of programming languages.

What is the Common Language Specification (CLS)?

The Common Language Specification is a set of constructs and constraints that serves as a guide for library writers and compiler writers. It allows libraries to be fully usable from any language supporting the CLS, and for those languages to integrate with each other. The Common Language Specification is a subset of the common type system. The Common Language Specification is also important to application developers who are writing code that will be used by other developers. When developers design publicly

accessible APIs following the rules of the CLS, those APIs are easily used from all other programming languages that target the common language runtime.

What is the Microsoft Intermediate Language (MSIL)?

MSIL is the CPU-independent instruction set into which .NET Framework programs are compiled. It contains instructions for loading, storing, initializing, and calling methods on objects.

Combined with metadata and the common type system, MSIL allows for true cross-language integration.

Prior to execution, MSIL is converted to machine code. It is not interpreted.

What is managed code and managed data?

Managed code is code that is written to target the services of the common language runtime (see What is the Common Language Runtime?). In order to target these services, the code must provide a minimum level of information (metadata) to the runtime. All C#, Visual Basic .NET, and JScript .NET code is managed by default. Visual Studio .NET C++ code is not managed by default, but the compiler can produce managed code by specifying a command-line switch (/CLR).

Closely related to managed code is managed data—data that is allocated and de-allocated by the common language runtime's garbage collector. C#, Visual Basic, and JScript .NET data is managed by default. C# data can, however, be marked as unmanaged through the use of special keywords. Visual Studio .NET C++ data is unmanaged by default (even when using the /CLR switch), but when using Managed Extensions for C++, a class can be marked as managed by using the `__gc` keyword. As the name suggests, this means that the memory for instances of the class is managed by the garbage collector. In addition, the class becomes a full participating member of the .NET Framework community, with the benefits and restrictions that brings. An example of a benefit is proper interoperability with classes written in other languages (for example, a managed C++ class can inherit from a Visual Basic class). An example of a restriction is that a managed class can only inherit from one base class.

Assemblies

What is an assembly?

An assembly is the primary building block of a .NET Framework application. It is a collection of functionality that is built, versioned, and deployed as a single implementation unit (as one or more files). All managed types and resources are marked either as accessible only within their implementation unit, or as accessible by code outside that unit.

Assemblies are self-describing by means of their manifest, which is an integral part of every assembly.

The manifest:

- Establishes the assembly identity (in the form of a text name), version, culture, and digital signature (if the assembly is to be shared across applications).
- Defines what files (by name and file hash) make up the assembly implementation.

- Specifies the types and resources that make up the assembly, including which are exported from the assembly.
- Itemizes the compile-time dependencies on other assemblies.
- Specifies the set of permissions required for the assembly to run properly.

This information is used at run time to resolve references, enforce version binding policy, and validate the integrity of loaded assemblies. The runtime can determine and locate the assembly for any running object, since every type is loaded in the context of an assembly. Assemblies are also the unit at which code access security permissions are applied. The identity evidence for each assembly is considered separately when determining what permissions to grant the code it contains.

The self-describing nature of assemblies also helps makes zero-impact install and XCOPY deployment feasible.

What are private assemblies and shared assemblies?

A private assembly is used only by a single application, and is stored in that application's install directory (or a subdirectory therein). A shared assembly is one that can be referenced by more than one application. In order to share an assembly, the assembly must be explicitly built for this purpose by giving it a cryptographically strong name (referred to as a strong name). By contrast, a private assembly name need only be unique within the application that uses it.

By making a distinction between private and shared assemblies, we introduce the notion of sharing as an explicit decision. Simply by deploying private assemblies to an application directory, you can guarantee that that application will run only with the bits it was built and deployed with. References to private assemblies will only be resolved locally to the private application directory.

There are several reasons you may elect to build and use shared assemblies, such as the ability to express version policy. The fact that shared assemblies have a cryptographically strong name means that only the author of the assembly has the key to produce a new version of that assembly. Thus, if you make a policy statement that says you want to accept a new version of an assembly, you can have some confidence that version updates will be controlled and verified by the author. Otherwise, you don't have to accept them.

For locally installed applications, a shared assembly is typically explicitly installed into the global assembly cache (a local cache of assemblies maintained by the .NET Framework). Key to the version management features of the .NET Framework is that downloaded code does not affect the execution of locally installed applications. Downloaded code is put in a special download cache and is not globally available on the machine even if some of the downloaded components are built as shared assemblies.

The classes that ship with the .NET Framework are all built as shared assemblies.

If I want to build a shared assembly, does that require the overhead of signing and managing key pairs?

Building a shared assembly does involve working with cryptographic keys. Only the public key is strictly needed when the assembly is being built. Compilers targeting the .NET Framework provide command line options (or use custom attributes) for supplying the public key when building the assembly. It is common to keep a copy of a common public key in a source database and point build scripts to this key. Before the assembly is shipped, the assembly must be fully signed with the corresponding private key. This is done using an SDK tool called SN.exe (Strong Name).

Strong name signing does not involve certificates like Authenticode does. There are no third party organizations involved, no fees to pay, and no certificate chains. In addition, the overhead for verifying a strong name is much less than it is for Authenticode. However, strong names do not make any statements about trusting a particular publisher. Strong names allow you to ensure that the contents of a given assembly haven't been tampered with, and that the assembly loaded on your behalf at run time comes from the same publisher as the one you developed against. But it makes no statement about whether you can trust the identity of that publisher.

What is the difference between a namespace and an assembly name?

A namespace is a logical naming scheme for types in which a simple type name, such as MyType, is preceded with a dot-separated hierarchical name. Such a naming scheme is completely under the control of the developer. For example, types MyCompany.FileAccess.A and MyCompany.FileAccess.B might be logically expected to have functionality related to file access. The .NET Framework uses a hierarchical naming scheme for grouping types into logical categories of related functionality, such as the Microsoft® ASP.NET application framework, or remoting functionality. Design tools can make use of namespaces to make it easier for developers to browse and reference types in their code. The concept of a namespace is not related to that of an assembly. A single assembly may contain types whose hierarchical names have different namespace roots, and a logical namespace root may span multiple assemblies. In the .NET Framework, a namespace is a logical design-time naming convenience, whereas an assembly establishes the name scope for types at run time.

Application Deployment and Isolation

What options are available to deploy my .NET applications?

The .NET Framework simplifies deployment by making zero-impact install and XCOPY deployment of applications feasible. Because all requests are resolved first to the private application directory, simply copying an application's directory files to disk is all that is needed to run the application. No registration is required.

This scenario is particularly compelling for Web applications, Web Services, and self-contained desktop applications. However, there are scenarios where XCOPY is not sufficient as a distribution mechanism. An example is when the application has little private code and relies on the availability of shared assemblies, or when the application is not locally installed (but rather downloaded on demand). For these cases, the

.NET Framework provides extensive code download services and integration with the Windows Installer. The code download support provided by the .NET Framework offers several advantages over current platforms, including incremental download, code access security (no more Authenticode dialogs), and application isolation (code downloaded on behalf of one application doesn't affect other applications). The Windows Installer is another powerful deployment mechanism available to .NET applications. All of the features of Windows Installer, including publishing, advertisement, and application repair will be available to .NET applications in Windows Installer 2.0.

I've written an assembly that I want to use in more than one application. Where do I deploy it?

Assemblies that are to be used by multiple applications (for example, shared assemblies) are deployed to the global assembly cache. In the prerelease and Beta builds, use the /i option to the GACUtil SDK tool to install an assembly into the cache:

```
gacutil /i myDll.dll
```

Windows Installer 2.0, which ships with Windows XP and Visual Studio .NET will be able to install assemblies into the global assembly cache.

How can I see what assemblies are installed in the global assembly cache?

The .NET Framework ships with a Windows shell extension for viewing the assembly cache. Navigating to % windir%\assembly with the Windows Explorer activates the viewer.

What is an application domain?

An application domain (often AppDomain) is a virtual process that serves to isolate an application. All objects created within the same application scope (in other words, anywhere along the sequence of object activations beginning with the application entry point) are created within the same application domain. Multiple application domains can exist in a single operating system process, making them a lightweight means of application isolation.

An OS process provides isolation by having a distinct memory address space. While this is effective, it is also expensive, and does not scale to the numbers required for large web servers. The Common Language Runtime, on the other hand, enforces application isolation by managing the memory use of code running within the application domain. This ensures that it does not access memory outside the boundaries of the domain. It is important to note that only type-safe code can be managed in this way (the runtime cannot guarantee isolation when unsafe code is loaded in an application domain).

Garbage Collection

What is garbage collection?

Garbage collection is a mechanism that allows the computer to detect when an object can no longer be accessed. It then automatically releases the memory used by that object (as well as calling a clean-up routine, called a "finalizer," which is written by the user). Some garbage collectors, like the one used by .NET, compact memory and therefore decrease your program's working set.

How does non-deterministic garbage collection affect my code?

For most programmers, having a garbage collector (and using garbage collected objects) means that you never have to worry about deallocating memory, or reference counting objects, even if you use sophisticated data structures. It does require some changes in coding style, however, if you typically deallocate system resources (file handles, locks, and so forth) in the same block of code that releases the memory for an object. With a garbage collected object you should provide a method that releases the system resources deterministically (that is, under your program control) and let the garbage collector release the memory when it compacts the working set.

Can I avoid using the garbage collected heap?

All languages that target the runtime allow you to allocate class objects from the garbage-collected heap. This brings benefits in terms of fast allocation, and avoids the need for programmers to work out when they should explicitly 'free' each object.

The CLR also provides what are called ValueTypes—these are like classes, except that ValueType objects are allocated on the runtime stack (rather than the heap), and therefore reclaimed automatically when your code exits the procedure in which they are defined. This is how "structs" in C# operate.

Managed Extensions to C++ lets you choose where class objects are allocated. If declared as managed Classes, with the `__gc` keyword, then they are allocated from the garbage-collected heap. If they don't include the `__gc` keyword, they behave like regular C++ objects, allocated from the C++ heap, and freed explicitly with the "free" method.

Remoting

How do in-process and cross-process communication work in the Common Language Runtime?

There are two aspects to in-process communication: between contexts within a single application domain, or across application domains. Between contexts in the same application domain, proxies are used as an interception mechanism. No marshaling/serialization is involved. When crossing application domains, we do marshaling/serialization using the runtime binary protocol.

Cross-process communication uses a pluggable channel and formatter protocol, each suited to a specific purpose.

- If the developer specifies an endpoint using the tool `soapsuds.exe` to generate a metadata proxy, HTTP channel with SOAP formatter is the default.
- If a developer is doing explicit remoting in the managed world, it is necessary to be explicit about what channel and formatter to use. This may be expressed administratively, through configuration files, or with API calls to load specific channels. Options are:

HTTP channel w/ SOAP formatter (HTTP works well on the Internet, or anytime traffic must travel through firewalls)

TCP channel w/ binary formatter (TCP is a higher performance option for local-area networks (LANs))

When making transitions between managed and unmanaged code, the COM infrastructure (specifically, DCOM) is used for remoting. In interim releases of the CLR, this applies also to serviced components (components that use COM+ services). Upon final release, it should be possible to configure any remotable component.

Distributed garbage collection of objects is managed by a system called "leased based lifetime." Each object has a lease time, and when that time expires, the object is disconnected from the remoting infrastructure of the CLR. Objects have a default renew time—the lease is renewed when a successful call is made from the client to the object. The client can also explicitly renew the lease.

Interoperability

Can I use COM objects from a .NET Framework program?

Yes. Any COM component you have deployed today can be used from managed code, and in common cases the adaptation is totally automatic.

Specifically, COM components are accessed from the .NET Framework by use of a runtime callable wrapper (RCW). This wrapper turns the COM interfaces exposed by the COM component into .NET Framework-compatible interfaces. For OLE automation interfaces, the RCW can be generated automatically from a type library. For non-OLE automation interfaces, a developer may write a custom RCW and manually map the types exposed by the COM interface to .NET Framework-compatible types.

Can .NET Framework components be used from a COM program?

Yes. Managed types you build today can be made accessible from COM, and in the common case the configuration is totally automatic. There are certain new features of the managed development environment that are not accessible from COM. For example, static methods and parameterized constructors cannot be used from COM. In general, it is a good idea to decide in advance who the intended user of a given type will be. If the type is to be used from COM, you may be restricted to using those features that are COM accessible.

Depending on the language used to write the managed type, it may or may not be visible by default.

Specifically, .NET Framework components are accessed from COM by using a COM callable wrapper (CCW). This is similar to an RCW (see previous question), but works in the opposite direction. Again, if the .NET Framework development tools cannot automatically generate the wrapper, or if the automatic behavior is not what you want, a custom CCW can be developed.

Can I use the Win32 API from a .NET Framework program?

Yes. Using platform invoke, .NET Framework programs can access native code libraries by means of static DLL entry points.

Here is an example of C# calling the Win32 **MessageBox** function:

```
using System;
```

```
using System.Runtime.InteropServices;

class MainApp
{
    [DllImport("user32.dll", EntryPoint="MessageBox")]
    public static extern int MessageBox(int hWnd, String strMessage, String strCaption,
    uint uiType);

    public static void Main()
    {
        MessageBox( 0, "Hello, this is PInvoke in operation!", ".NET", 0 );
    }
}
```

Security

What do I have to do to make my code work with the security system?

Usually, not a thing—most applications will run safely and will not be exploitable by malicious attacks. By simply using the standard class libraries to access resources (like files) or perform protected operations (such as a reflection on private members of a type), security will be enforced by these libraries. The one simple thing application developers may want to do is include a permission request (a form of declarative security) to limit the permissions their code may receive (to only those it requires). This also ensures that if the code is allowed to run, it will do so with all the permissions it needs.

Only developers writing new base class libraries that expose new kinds of resources need to work directly with the security system. Instead of all code being a potential security risk, code access security constrains this to a very small bit of code that explicitly overrides the security system.

Why does my code get a security exception when I run it from a network shared drive?

Default security policy gives only a restricted set of permissions to code that comes from the local intranet zone. This zone is defined by the Internet Explorer security settings, and should be configured to match the local network within an enterprise. Since files named by UNC or by a mapped drive (such as with the NET USE command) are being sent over this local network, they too are in the local intranet zone.

The default is set for the worst case of an unsecured intranet. If your intranet is more secure you can modify security policy (with the .NET Framework Configuration tool or the CASPol tool) to grant more permissions to the local intranet, or to portions of it (such as specific machine share names).

How do I make it so that code runs when the security system is stopping it?

Security exceptions occur when code attempts to perform actions for which it has not been granted permission. Permissions are granted based on what is known about code; especially its location. For example, code run from the Internet is given fewer permissions than that run from the local machine because experience has proven that it is generally less reliable. So, to allow code to run that is failing due to security exceptions, you must increase the permissions granted to it. One simple way to do so is to move the code to a more trusted location (such as the local file system). But this won't work in all cases (web applications are a good example, and intranet applications on a corporate network are another). So, instead of changing the code's location, you can also change security policy to grant more permissions to that location. This is done using either the .NET Framework Configuration tool or the code access security policy utility (caspol.exe). If you are the code's developer or publisher, you may also digitally sign it and then modify security policy to grant more permissions to code bearing that signature. When taking any of these actions, however, remember that code is given fewer permissions because it is not from an identifiable trustworthy source—before you move code to your local machine or change security policy, you should be sure that you trust the code to not perform malicious or damaging actions.

How do I administer security for my machine? For an enterprise?

The .NET Framework includes the .NET Framework Configuration tool, an MMC snap-in (mscorcfg.msc), to configure several aspects of the CLR including security policy. The snap-in not only supports administering security policy on the local machine, but also creates enterprise policy deployment packages compatible with System Management Server and Group Policy. A command line utility, CASPol.exe, can also be used to script policy changes on the computer. In order to run either tool, in a command prompt, change the current directory to the installation directory of the .NET Framework (located in %windir%\Microsoft.Net\Framework\v1.0.2914.16\) and type **mscorcfg.msc** or **caspol.exe**.

How does evidence-based security work with Windows 2000 security?

Evidence-based security (which authorizes code) works together with Windows 2000 security (which is based on log on identity). For example, to access a file, managed code must have both the code access security file permission and must also be running under a log on identity that has NTFS file access rights. The managed libraries that are included with the .NET Framework also provide classes for role-based security. These allow the application to work with Windows log on identities and user groups.

MCAD or MCSA

Create ASP.NET pages.

- Add and set directives on ASP.NET pages.
- Separate user interface resources from business logic.

Add Web server controls, HTML server controls, user controls, and HTML code to ASP.NET pages.

<ul style="list-style-type: none"> ▪ Set properties on controls. ▪ Load controls dynamically. ▪ Apply templates. ▪ Set styles on ASP.NET pages by using cascading style sheets. ▪ Instantiate and invoke an ActiveX® control. 			
<p>Implement navigation for the user interface.</p> <ul style="list-style-type: none"> ▪ Manage the view state. ▪ Manage data during postback events. ▪ Use session state to manage data across pages. 	●		●
<p>Validate user input.</p> <ul style="list-style-type: none"> ▪ Validate non-Latin user input. 	◐		◐
<p>Implement error handling in the user interface.</p> <ul style="list-style-type: none"> ▪ Configure custom error pages. ▪ Implement Global.asax, application, page-level, and page event error handling. 	◐		◐
<p>Implement online user assistance.</p>			
<p>Incorporate existing code into ASP.NET pages.</p>			◐
<p>Display and update data.</p> <ul style="list-style-type: none"> ▪ Transform and filter data. ▪ Bind data to the user interface. ▪ Use controls to display data. 	●	◐	●
<p>Instantiate and invoke Web services or components.</p> <ul style="list-style-type: none"> ▪ Instantiate and invoke a Web service. ▪ Instantiate and invoke a COM or COM+ component. ▪ Instantiate and invoke a .NET component. ▪ Call native functions by using platform invoke. 	◐	○	●
<p>Implement globalization.</p> <ul style="list-style-type: none"> ▪ Implement localizability for the user interface. ▪ Convert existing encodings. ▪ Implement right-to-left and left-to-right mirroring. ▪ Prepare culture-specific formatting. 			
<p>Handle events.</p> <ul style="list-style-type: none"> ▪ Create event handlers. 	●		●

<ul style="list-style-type: none"> Raise events. 			
Implement accessibility features.	●		
Use and edit intrinsic objects. Intrinsic objects include response, request, session, server, and application.			
<ul style="list-style-type: none"> Retrieve values from the properties of intrinsic objects. Set values on the properties of intrinsic objects. Use intrinsic objects to perform operations. 	●		●
Creating and Managing Components and .NET Assemblies			
Create and modify a .NET assembly.			
<ul style="list-style-type: none"> Create and implement satellite assemblies. Create resource-only assemblies. 			
Create Web custom controls and Web user controls.	◐		◐
Consuming and Manipulating Data			
Access and manipulate data from a Microsoft SQL Server™ database by creating and using ad hoc queries and stored procedures.	●	●	●
Access and manipulate data from a data store. Data stores include relational databases, XML documents, and flat files. Methods include XML techniques and ADO.NET.	●	●	●
Handle data errors.	◐	●	◐
Testing and Debugging			
Create a unit test plan.			
Implement tracing.			
<ul style="list-style-type: none"> Add trace listeners and trace switches to an application. Display trace output. 	●		●
Debug, rework, and resolve defects in code.			
<ul style="list-style-type: none"> Configure the debugging environment. Create and apply debugging code to components, pages, and applications. Provide multicultural test data to components, pages, and applications. Execute tests. Resolve errors and rework code. 	◐		◐
Deploying a Web Application			
Plan the deployment of a Web application.			
<ul style="list-style-type: none"> Plan a deployment that uses removable media. Plan a Web-based deployment. Plan the deployment of an application to a Web garden, a Web farm, or a 			

cluster.			
Create a setup program that installs a Web application and allows for the application to be uninstalled.			●
Deploy a Web application.			
Add assemblies to the global assembly cache.			
Maintaining and Supporting a Web Application			
Optimize the performance of a Web application.			
Diagnose and resolve errors and issues.			
Configuring and Securing a Web Application			
Configure a Web application. <ul style="list-style-type: none"> Modify the Web.config file. Modify the Machine.config file. Add and modify application settings. 	●		●
Configure security for a Web application. <ul style="list-style-type: none"> Select and configure authentication type. Authentication types include Windows® Authentication, None, forms-based, Microsoft Passport, Internet Information Services (IIS) authentication, and custom authentication. 	●		●
Configure authorization. Authorization methods include file-based methods and URL-based methods. <ul style="list-style-type: none"> Configure role-based authorization. Implement impersonation. 	◐		◐
Configure and implement caching. Caching types include output, fragment, and data. <ul style="list-style-type: none"> Use a cache object. Use cache directives. 	●		●
Configure and implement session state in various topologies such as a Web garden and a Web farm. <ul style="list-style-type: none"> Use session state within a process. Use session state with session state service. Use session state with Microsoft SQL Server. 	◐		◐
Install and configure server services. <ul style="list-style-type: none"> Install and configure a Web server. Install and configure Microsoft FrontPage® Server Extensions. 			

Compare MCAD to MCSD

Identify Your Role in the Software Development Cycle



Choose the

MCAD credential if you:

- Develop, test, deploy, and maintain department-level applications, components, Web or desktop clients, or database and network services using Microsoft tools and technologies.
- Have one to two years of experience building, deploying, and maintaining applications.

The MCAD credential was created in response to industry demand for a certification that allows developers to show they have the skills necessary to successfully implement functional specifications and build, deploy, and maintain Microsoft Windows® and Web applications. Achieving the MCAD credential can be a step toward earning the MCSD credential for advanced developers.

Related job titles: programmer, programmer/analyst, and software developer.

Choose the MCSD credential if you:

- Analyze and design leading-edge enterprise solutions with Microsoft development tools, technologies, and platforms.
- Have at least two years of experience in a lead developer role analyzing business and technical requirements, and defining solution architecture.

The MCSD credential is one of the most widely recognized technical certifications in the industry. By earning the premier MCSD for .NET credential, individuals demonstrate that they have the skills necessary to lead organizations in the successful design, implementation, and administration of business solutions with Microsoft products.

Related job titles: software engineer, software development engineer, software architect, and consultant.

OOP's Concepts:

VB.NET now supports the four major defining concepts required for a language to be fully object-oriented:

- Abstraction
- Encapsulation
- Polymorphism
- Inheritance

In the following article, we'll define and discuss each of these four major object oriented concepts.

Abstraction

VB has supported abstraction since VB4. Abstraction is merely the ability of a language to create "black box" code -- to take a concept and create an abstract representation of that concept within a program. A Customer object, for instance, is an abstract representation of a real-world customer. A Recordset object is an abstract representation of a set of data. Abstraction allows us to recognize how things are similar and ignore differences ¶ to think in general terms, and not the specifics. A text box control is an abstraction, because we can place it on a form, and then tailor it to our needs by setting properties. Visual Basic allows us to define abstractions using class modules.

Any language that allows a developer to create a class from which objects can be instantiated meets this criteria, and Visual Basic is no exception. We can easily create a class to represent a customer, essentially providing an abstraction. We can then create instances of that class, where each object can have its own attributes such that it represents a specific customer.

In VB.NET we implement abstraction by creating a class using the Class keyword.

Encapsulation

Perhaps the most important of the object-oriented concepts is that of **encapsulation**. Encapsulation is the concept that an object should totally separate its interface from its implementation. All the data and implementation code for an object should be entirely hidden behind its interface.

The idea is that we can create an interface (Public methods in a class) and, as long as that interface remains consistent, the application can interact with our objects. This remains true even if we entirely rewrite the code within a given method ¶ thus the interface is independent of the implementation.

Encapsulation allows us to hide the internal implementation details of a class. For example, the algorithm we use to find prime numbers might be proprietary. We can expose a simple API to the end user, but we hide all of the logic used for our algorithm by encapsulating it within our class.

This means that an object should completely contain any data it requires, and that it should also contain all the code required to manipulate that data. Programs should interact with our object through an interface, using properties and methods. Client code should never work directly with the data owned by the object.

In object-speak, programs interact with objects by sending messages to the object indicating which method or property they'd like to have invoked. These messages are generated by other objects, or by external sources such as the user. The way the object reacts to these messages is through methods or properties.

Visual Basic has provided full support for encapsulation through class modules since version 4.0. Using these modules, we can create classes that entirely hide their internal data and code, providing a well-established interface of properties and methods to the outside world.

Polymorphism

Likewise, **polymorphism** was introduced with VB4. Polymorphism is often considered to be directly tied to inheritance (which we'll discuss next). In reality, however, it's independent to a large degree. Polymorphism means that we can have two classes with different implementations or code, but with a common set of methods or properties. We can then write a program that operates upon that interface and doesn't care about which type of object it operates at runtime. For instance, if both Customer and Vendor objects have a Name property, and we can write a routine that calls the Name property regardless of whether we're using a Customer or Vendor object, then we have polymorphism.

To really understand polymorphism we need to explore the concept of a **method signature**, also sometimes called a prototype. All methods have a signature, which is defined by the method's name and the data types of its parameters.

Polymorphism merely says that we should be able to write some client code that calls methods on an object -- and as long as the object provides our methods with the method signatures we expect, we don't care which *class* the object was created from.

We can use several techniques to achieve polymorphic behavior:

- Late binding
- Multiple interfaces
- .NET Reflection
- Inheritance

Late binding actually allows us to implement "pure" polymorphism at the cost of performance and ease of programming. Through multiple interfaces and inheritance we can also achieve polymorphism with much better performance and ease of programming. Reflection allows us to use either late binding or multiple interfaces, but against objects created in a very dynamic way -- going even so far as to dynamically load a DLL into our application at runtime so we can use its classes.

Inheritance

Inheritance is the concept that a new class can be based on an existing class, inheriting its interface and functionality from the original class. This is done by inheriting these behaviors from the existing class through a process known as subclassing. VB.NET is the first version of VB that supports inheritance. With the introduction of full inheritance, VB is now a fully OBJECT-ORIENTED language by any reasonable definition.

Inheritance is one of the most powerful object-oriented features a language can support. At the same time, inheritance is one of the most dangerous and misused object-oriented features.

Properly used, inheritance allows us to increase the maintainability, readability and reusability of our application, by offering us a clear and concise way to reuse code -- both via interface and implementation. Improperly used, inheritance allows us to create applications that are very fragile, where a change to a class can cause the entire application to break or require changes.

Inheritance allows us to implement an *is-a* relationship. In other words, it allows us to implement a new class that *is* a more specific type of its base class. This means that properly used, inheritance allows us to create child classes that *really are* the same as the base class.

This is the challenge. Inheritance is *not* just a mechanism for code reuse. It is a mechanism to create classes that flow naturally from some other class. If we use it anywhere we want code reuse, we'll end up with a real mess on our hands. If we use it anywhere we just want a common interface, but where the child class is not *really* the same as the base class then we should be using multiple interfaces -- something we discuss in our book.

The question we must ask, when using inheritance, is whether the child class *is* a more specific version of the base class.

Properly applied, object-oriented design and programming can allow us to create very large and complex applications that remain maintainable and readable over time. However, OO is no magic bullet, and improperly applied these technologies and concepts can create the same hard to maintain code that we might create using procedural or modular design techniques.

