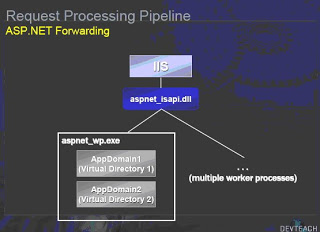
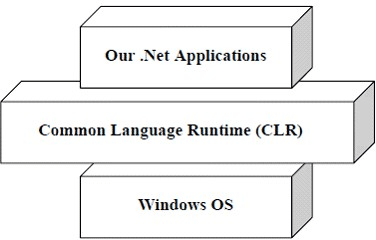
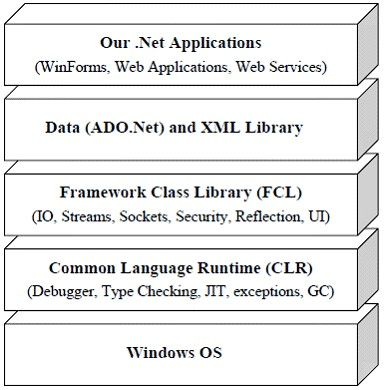
1. **Windows Application**
   1. Windows Forms (WinForms) is a graphical (**GUI**) class library included as a part of **Microsoft** .NET Framework, providing a platform to write rich client applications for desktop, laptop, and tablet PCs.
2. **Web application**
   1. A **Web application** (**Web** app) is an **application** program that is stored on a remote server and delivered over the Internet through a browser interface.
3. **Mobile application**
   1. A **mobile app** is a software **application** developed specifically for use on small, wireless computing devices, such as smartphones and tablets, rather than desktop or laptop computers.
4. **web service**
   1. A **web service** enables communication among various **applications** by using open standards such as HTML, XML, WSDL, and SOAP. A **web service** takes the help of: XML to tag the data. SOAP to transfer a message. WSDL to describe the availability of **service**.
5. **Windows service**
   1. A **Windows service** is a computer program that operates in the background. It is similar in concept to a Unix daemon. A **Windows service** must conform to the interface rules and protocols of the **Service** Control Manager, the component responsible for managing **Windows services**.

**ASP.NET**

1. ASP.NET **Page Life Cycle**. When a **page** is requested, it is loaded into the server memory, processed, and sent to the browser. Then it is unloaded from the memory. At each of these steps, methods and events are available, which could be overridden according to the need of the application.
2. **ASP.Net Page Life Cycle**
3. [](http://1.bp.blogspot.com/_JrGaH_ZNMPU/SQmWY5hekDI/AAAAAAAAABw/jzOt0mghm3c/s1600-h/AspLifeCycle2.gif)
5. **Introduction**  
   Many of us know that IIS is a web server and we use it in our .Net application since we need a web server to run a web application. But I wonder as many of us don't know the internal architecture of IIS. This article is written for beginners to know the architecture of IIS.  
   **How the simple web page execution happens?**  
   As all of us know a request comes from Client (Browser) and sends to Server (we call it as Web server) in turn server process the request and sends response back to the client in according to the client request.  
   But internally in the web server there is quite interesting process that happens. To get aware of that process we should first of all know about the architecture of the IIS  
   It mainly consists of 3 Parts/Files  
   Inetinfo.exec  
   ISAPI Filer (Container for Internet Server Application Interface dlls),  
   Worker Process (aspnet\_wp.exe)
6. When ever a request comes from the client:  
   Inetinfo.exe is the ASP.Net request handler that handles the requests from the client .If it's for static resources like HTML files or image files inetinfo.exe process the request and sent to client. If the request is with extension aspx/asp, inetinfo.exe processes the request to API filter. ISAPI filter will have several runtime modules called as ISAPI extensions. To process the request ISAPI filter takes the help of these runtime modules. The runtime module loaded for ASP page is asp.dll. And for ASP.NET page it's ASPNET\_ISAPI.dll. From here the request is processed to the "worker process". Worker Process will have several application domains.  
   Application Domain  
   The purpose of the application domain is in order to isolate one application from another. When ever we create a new application, application domains are created automatically by the CLRHost. Worker process will create a block of memory related to particular application. Application domains provide a more secure and versatile unit of processing that the common language runtime can use to provide isolation between applications. Application domains are normally created by runtime hosts. Runtime host is responsible for bootstrapping the common language runtime before an application is run.  
   Worker process sends the request to HTTPPIPE line.(HTTP Pipeline is nonetheless collection of .net framework classes). HTTP Pipeline compiles the request into a library and makes a call to HTTP runtime and runtime creates an instance of page class  
   Public Class File  
   Inherits System.Web.UI.Page  
   End Class 'File  
   ASP.Net web page is a class derived from page class, this page class resides in system.web.dll  
   After creating instance pf page class HTTP Runtime immediately invokes process request method of page class  
   Dim Req As New Page  
     
   Req.ProcessRequest()  
   Process Request Method does following things:  
   Intialize the memory  
   Load the view state  
   Page execution and post back events  
   Rendering HTML content  
   Releasing the memory  
   Process Request Method executes set of events for page class .These are called as Page life cycle events.  
   **Page Life Cycle Events**  
   Page\_InitThe server controls are loaded and initialized from the Web form's view state. This is the first step in a Web form's life cycle.  
   Page\_LoadThe server controls are loaded in the page object. View state information is available at this point, so this is where you put code to change control settings or display text on the page.  
   Page\_PreRenderThe application is about to render the page object.  
   Page\_UnloadThe page is unloaded from memory.  
   Page\_DisposedThe page object is released from memory. This is the last event in the life of a page object.  
   Page\_ErrorAn unhandled exception occurs.  
   Page\_AbortTransactionA transaction is aborted.  
   Page\_CommitTransactionA transaction is accepted.  
   Page\_DataBindingA server control on the page binds to a data source.  
   Process Request Method finally renders HTML Page  
   Dependencies:  
   When the request comes to ASP.net worker process, it will be forwarded to HTTP Application factory. This "Application Factory" will maintain address of the application domains which are currently executing under worker process. If the required virtual directory application domain is unavailable it will create a new application domain. If the application domain is already existing, the request will be forwarded to corresponding AppDomain.  
   Application Domain maintains page handler factory class. This will contain all libraries addresses corresponding to webpage. If the requested webpage library is available the instance of the page class is created, if the library is unavailable the request will be forwarded to HTTP pipeline.
7. **Please ref the following links.**
8. <http://www.codeproject.com/KB/aspnet/lifecycle.aspx>  
   <http://www.vbdotnetheaven.com/UploadFile/hima_.net/WebPageExecutioninIIS11262006235902PM/WebPageExecutioninIIS.aspx>  
   <http://msdn.microsoft.com/en-us/library/ms178472.aspx>

**.Net Architecture and .Net Framework basics:**

1. **Common Language Runtime (CLR):** The heart of the .Net Framework. It is also called the .Net runtime. It resides above the operating system and handles all .Net applications. It handles garbage collection, Code Access Security (CAS) etc.  
     
   
2. **Microsoft Intermediate Language (MSIL) Code:** When we compile our .Net code then it is not directly converted to native/binary code; it is first converted into intermediate code known as MSIL code which is then interpreted by the CLR. MSIL is independent of hardware and the operating system. Cross language relationships are possible since MSIL is the same for all .Net languages. MSIL is further converted into native code.  
     
   NET2.gif
3. **Just in Time Compilers (JIT):** It compiles IL code into native executable code (exe or dlls). Once code is converted to IL then it can be called again by JIT instead of recompiling that code.
4. **Framework class library:** The .Net Framework provides a huge class library called FCL for common tasks. It contains thousands of classes to access Windows APIs and common functions like string manipulations, Data structures, stream, IO, thread, security etc.
5. **Common Language Specification (CLS):** What makes a language to be .Net compliant? Answer is CLS. Microsoft has defined some specifications that each .Net language has to follow. For e.g.: no pointer, no multiple inheritances etc.
6. **Common Type System (CTS):** CTS defines some basic data types that IL can understand. Each .Net compliant language should map its data types to these standard data types. This makes it possible for two .Net compliant languages to communicate by ing/receiving parameters to and from each other. For example CTS defines Int32 for C# int and VB integer data types.
7. **The .Net Framework:** Is a combination of CLR, FCL, ADO.Net and XML classes, Web/Window applications and Web services.  
     
   
8. **.EXE vs DLL:**
   1. **EXE** 
      1. EXE is Executable File
      2. EXE is run individually
      3. EXE Has Main Function
      4. EXE Mainly is for standalone application
   2. **DLL** 
      1. DLL is Dynamic Link Library
      2. DLL can't run individually
      3. DLL doesn't contain Main Function
      4. DLL give support to exe

**NET Framework Fundamentals**

Understanding the .NET Framework

.NET framework is a Windows Component that supports the building and running of windows applications and XML Web services. The purpose of the component is to provide the user with a consistent object oriented programming environment whether the code is stored locally or remotely.

It aims to minimize software deployment and versioning conflicts and also promote safe execution of code including codes executed by trusted third parties. It is directed towards eliminating performance problems of scripted or interpreted environments. The effort is to make developer experience consistent across a variety of applications and platforms and create communication standards that help .NET framework applications integrate with all other web based applications.

The .NET framework has two major components– The Common Runtime (CLR) and the Class Library

The CLR is the foundation upon which the .NET Framework has been built. The runtime manages code at execution time and provides all the core services such as memory management, thread management and remoting. It also enforces strict type safety and ensures code accuracy in order to provide security and robustness to the applications. This capability to manage code at runtime is the distinguishing feature of the CLR. All code that is managed by the CLR is known as managed code while other codes are known as unmanaged code.

**CLR Features**

1. CLR manages memory, thread execution, code execution, compilation code safety verification and other system services.

2. For security reasons, managed code is assigned varying degrees of trust based on origin. This prevents or allows the managed component from performing file access operations, registry access operations or other sensitive functions even within the same active application.

3. The Runtime enforces code robustness by implementing strict type and code verification infrastructure called Common type System (CTS). The CTS ensures that all managed code is self describing and all Microsoft or third party language compiler generated codes conform to CTS. This enables the managed code to consume other managed types and enforce strict type fidelity and type safety.

4. CLR eliminates many common software issues like handling of object layout, references to objects and garbage clearance. This type of memory management prevents memory leaks and invalid memory references.

5. The CLR also accelerates developer productivity. The programmer is free to choose the language of the application without worrying about compatibility and integration issues. He is also enabled to take advantage of the runtime and the class library of the .NET Framework and also harvest components from other applications written in different languages by different developers. This implicitly eases the process of migration.

6. Though CLR aims to be futuristic software, it lends support to existing applications. The interoperability between the managed and unmanaged codes makes this process extremely simple.   
7. The design of the CLR is geared towards enhancing performance. The Just-in-time (JIT) compiling enables managed code to run in the native machine language of the system executing it. During the process the memory manager removes the possibilities of fragmented memory and increases memory locality-of-reference to enhance performance.

8. Finally, server side applications can host runtime. High performance servers like Microsoft SQL Server and Internet Information Services can host this CLR and the infrastructure so provided can be used to write business logic while enjoying the best benefits of enterprise server support.

The **Class Library** is an object oriented collection of reusable types. It is comprehensive and the types can be used to develop command line applications or GUI applications such as Web forms or XML Web services. Unmanaged components that load CLR into their processes can be hosted by the .NET Framework to initiate the execution of managed code. This creates a software environment that exploits both the managed and unmanaged codes. The.NET Framework also provides a number of runtime hosts and supports third party runtime hosts

**Class Library Features**

1. The class library is a collection of reusable types that integrate with the CLR.

2. It is object oriented and provides types from which user defined types can derive functionality. This makes for ease of use and is time saving.

3. Third party components can be integrated seamlessly with classes in the .NET framework.

4. It enables a range of common programming tasks such as string management, data collection and file access.

5. It supports a variety of specialized development scenarios such as console application development, Windows GUI applications, ASP.NET Applications, XML Web services.

{mospagebreak}

.

**The Common Type System (CTS)**

A number of types are supported by the CLR and are described by the CTS. Both value types are supported—primitive data types and reference types. The primitive data types include Byte, Int16, Double and Boolean while Reference types include arrays, classes and object and string types. Reference types are types that store a reference to the location of their values. The value is stored as part of a defined class and is referenced through a class member on the instance of a class.

User defined value types and enumerations are derived from the value types mentioned above.

Language compilers implement types using their own terminology.

The process of converting a value type to a reference type and vice versa is called boxing and unboxing. The implicit conversion of a value type to a reference type is referred to as boxing. The explicit conversion of an object type into a specific value type is referred to as unboxing.

**The Common Language Infrastructure (CLI)**

A subset of the .NET framework is the CLI. The CLI includes the functionality of the Common Language Runtime and specifications for the Common Type System, metadata and Intermediate language. A subset of the Framework Class Library incorporates the base class library, a Network library, a Reflection library, an XML library and Floating point and Extended Array Library. The shared source implementation of the CLI is available for both the FreeBSD and Windows operating Systems.

**The Common Language Specification (CLS)**

The CLR supports the CLS which is a subset of it. Additionally the CLR supports a set of rules that language and compiler designers follow. It provides robust interoperability between the .NET languages and the ability to inherit classes written in one language in any other .NET language. Cross language debugging also becomes a possibility in this scenario. It must be noted that the CLS rules apply only to publicly exposed features of a class.

**Classes**

A blueprint of an object is called a class. All definitions of haw a particular object will be instantiated at runtime, its properties and methods and storage structures are defined in the class. Classes are used by developers for creating instances of the class at runtime using the keyword “New”.

**Namespaces**

This is the key part of the .NET Framework. It provides scope for both preinstalled framework classes and custom developed classes. Vb.NET uses the “Imports” keyword to enable the use of member names from the namespace declared. C# uses the “using” keyword. In both cases the System Namespace is also imported so that the Console window can be written without explicitly referring to the System.Console.

**Assemblies**

Assemblies are also known as managed DLLs. They are the fundamental unit of deployment for the .NET platform. The .NET framework itself is made of a number of assemblies. An assembly contains the Intermediate language generated by the language compiler, an assembly manifest, type metadata and resources. They can be private or public. They are self describing and hence different versions of the same assembly can be run simultaneously.

**Intermediate language (IL)**

This is a processor independent representation of executable code. It is similar to assembly code and specific to the CLR. It is generated by the language compilers that target the CLR. At runtime, the CLR just-in-time compiles the IL to native code for execution. The tool ngen.exe which is part of the .NET framework pre-compiles assemblies to native code at install time and caches the precompiled code to the disk.

**Managed execution**

This refers to code whose execution is managed by the CLR. It includes memory management, access security, cross-language integration for debugging and exception handling etc. These assemblies are required for the creation of metadata on the code and the assemblies so that the CLR can manage the execution of the code.

**Manifests, Metadata and Attributes**

Metadata and manifests are key aspects of managed code execution. The portions of an assembly that contains descriptive information about the types contained in the assembly, the members exposed by the assembly and the resources required by the assembly are called manifests. Metadata is contained within the manifest. This metadata describes the assembly and some of it is generated by the language compiler at compile time. Other metadata may be added by the developer at design time. Declarations added to the code to describe or modify some aspect of the code’s behavior at runtime are known as Attributes. These are stored with an assembly as metadata. They serve many useful purposes in the .NET Framework

**Object Orientation in the .NET Framework**

Objects are the core of Object oriented programming. Classes are blueprints of objects and contain all the methods and properties of the object. Encapsulation, inheritance and polymorphism are attributes of an object. Encapsulation means the ability of an object to hide its internal data from outside view and allow access to only that data that is publicly available. Inheritance is the ability to derive one class from another.

New classes can be created from existing classes and the new class inherits all the properties and methods of the old class and new methods and events can be added to the new class. This is useful when users want to create specialized classes. Polymorphism is the ability of multiple classes derived from the same base class to expose methods in the same name, regardless of the underlying process of implementation.

**Rapid Development and Reuse**

The object orientation of the .NET Framework provides for faster development and deployment of applications. The use of classes, derived classes to provide common functionality has gone a long way in reducing development time. Object orientation is also the crucial element in the development of the code-behind concept and the latest code beside concept. Code behind allows developers to separate executable code form the HTML markup of the user interface. The executable code is placed in a module called code behind file. This file contains a class that inherits from the Page class. The ASP.NET page inherits from code-behind class and the two are compiled at runtime into a single executable assembly.

The BETA 2.0 has added a number of functionalities to aid in rapid development. We will be looking at these changes in the next unit “What’s new in BETA 2.0”

**Choosing a Language**

An important aspect of the .NET framework is that developers can continue to use the language of their choice in application development. The cross language interoperability in .NET makes it possible to create an application in any .NET supported language as all languages will work together smoothly using the CLR which translates all languages into Intermediary language.

## **Introduction**

The Microsoft® .NET Framework 1.1 is an integral Microsoft Windows® component for building and running the next generation of software applications and Extensible Markup Language (XML) Web services—components that facilitate integration by sharing data and functionality over a network through standard, platform-independent protocols such as XML, SOAP, and HTTP. The .NET Framework consists of two main parts: the common language runtime (CLR) and a unified set of class libraries, including ASP.NET for Web applications and Web services, Windows Forms for smart client applications, and ADO.NET for loosely coupled data access.

The Microsoft .NET Compact Framework is the smart device development framework for Microsoft .NET, bringing the world of managed code and XML Web services to devices. The .NET Compact Framework is a rich subset of the .NET Framework, thus providing the same benefits as the .NET Framework. But the .NET Compact Framework is designed specifically for resource-constrained devices, such as PDAs and smart mobile phones. The .NET Compact Framework greatly simplifies the process of creating and deploying applications to mobile devices while also allowing the developer to take full advantage of the capabilities of the device.

### .NET Framework

Code written on the .NET Framework platform is called managed code. The term "managed code" refers to the fact that the Common Language Runtime (CLR) provides several assurances for such code:

* Managed code cannot have bad pointers.
* Managed code cannot create memory leaks.
* Managed code supports strong type-safety.

A key benefit of managed code is that the managed development environment handles certain common errors that plague Win32® programmers.

Managed code is compiled down to a combination of MSIL (Microsoft Intermediate Language) and metadata. These are merged into a Pre Execution Environment (PE) file, which can then be executed on any CLR-capable machine. When you run this executable, the JIT starts compiling the IL down to native code. The result is that all .NET Framework components run as native code. Code that requires the CLR at run-time in order to execute is referred to as managed code. The CLR is responsible for managing execution of code that runs on the .NET Framework. There are two DLLs responsible for this: mscoree.dll (native code that loads managed code) and mscorlib.dll (managed code: base class library, the runtime library).

* [Overview](http://www.codeproject.com/Articles/430014/N-Tier-Architecture-and-Tips#Overview)
* [**N-Tier Architecture Introduction**](http://www.codeproject.com/Articles/430014/N-Tier-Architecture-and-Tips#N-Tier Architecture Introduction)
  + [Some Terminology's Difference and Relationship](http://www.codeproject.com/Articles/430014/N-Tier-Architecture-and-Tips#Some Terminology's Difference and Relationships)
    - [**Tier and Layer**](http://www.codeproject.com/Articles/430014/N-Tier-Architecture-and-Tips#Tier and Layer Relationship)
    - [Tier And Process](http://www.codeproject.com/Articles/430014/N-Tier-Architecture-and-Tips#Tier And Process Relationship)
    - [Layer and Process](http://www.codeproject.com/Articles/430014/N-Tier-Architecture-and-Tips#Layer and Process Relationship)
  + [**3-Tier Architecture**](http://www.codeproject.com/Articles/430014/N-Tier-Architecture-and-Tips#3-Tier Architecture)
  + [**1, 2, 3 or More Tier Architectures**](http://www.codeproject.com/Articles/430014/N-Tier-Architecture-and-Tips#1, 2, 3 or More Tier Architecture)
* [**Advantages and Disadvantages of Different Tier Architectures**](http://www.codeproject.com/Articles/430014/N-Tier-Architecture-and-Tips#Advantages and Disadvantages of Different Tier Architectures)
  + [**1 or 2-Tier Architecture**](http://www.codeproject.com/Articles/430014/N-Tier-Architecture-and-Tips#1 or 2-Tier)
    - [Advantages](http://www.codeproject.com/Articles/430014/N-Tier-Architecture-and-Tips#12Advantages)
    - [Disadvantages](http://www.codeproject.com/Articles/430014/N-Tier-Architecture-and-Tips#12Disadvantages)
  + [**N-Tier Architecture**](http://www.codeproject.com/Articles/430014/N-Tier-Architecture-and-Tips#N-Tier Architecture)
    - [Advantages](http://www.codeproject.com/Articles/430014/N-Tier-Architecture-and-Tips#nAdvantages)
    - [Disadvantages](http://www.codeproject.com/Articles/430014/N-Tier-Architecture-and-Tips#The Disadvantage of the N-Tier Deployment)
* [Business Data Validation in N-Tier Architecture](http://www.codeproject.com/Articles/430014/N-Tier-Architecture-and-Tips#Business Data Validation in N-Tier Architecture)
* [**How to Deploy N-Tier Application Correctly**](http://www.codeproject.com/Articles/430014/N-Tier-Architecture-and-Tips#How to Deploy N-Tier Application Correctly)
* [**How to Achieve N-Tier Deployment Capability by Software Technology**](http://www.codeproject.com/Articles/430014/N-Tier-Architecture-and-Tips#How to Achieve N-Tier Deployment Capability by Software Technologies)
* [Some Practical Tips on N-Tier Architecture Development](http://www.codeproject.com/Articles/430014/N-Tier-Architecture-and-Tips#Some Practical Tips on N-Tier Architecture Development)
* [Conclusions](http://www.codeproject.com/Articles/430014/N-Tier-Architecture-and-Tips#Conclusions)

## **Overview**

N-Tier architecture is an industry-proved software architecture model, suitable to support enterprise-level client/server applications by resolving issues like scalability, security, fault tolerance and etc. .NET has many tools and features, but .NET doesn’t have pre-defined ways to guard how to implement N-Tier architecture. Therefore, in order to achieve good design and implementation of N-Tier architecture in .NET, understanding fully its concepts is very important. However, many of us may hear, read or use N-Tier architecture for many years but still misunderstand its concepts more or less. This article tries to clarify many basic concepts in N-Tier architecture from all aspects, and also provide some practical **tips**. The tips in this article are based on the assumption that a team has a full control over all layers of the N-Tier architecture. We have another article to elaborate a N-Tier architecture sample in .NET: [A N-Tier Architecture Sample with ASP.NET MVC3, WCF and Entity Framework](http://www.codeproject.com/Articles/434282/A-N-Tier-Architecture-Sample-with-ASP-NET-MVC3-WCF).

## **N-Tier Architecture Introduction**

### Some Terminology's Difference and Relationship

#### Tier and Layer

Firstly we need to clarify the difference between two terms in N-Tier architecture: tier and layer. Tier usually means the physical deployment computer. Usually an individual running server is one tier. Several servers may also be counted as one tier, such as server failover clustering. By contrast, layer usually means logic software component group mainly by functionality; layer is used for software development purpose. Layer software implementation has many advantages and is a good way to achieve N-Tier architecture. Layer and tier may or may not exactly match each other. Each layer may run in an individual tier. However, multiple layers may also be able to run in one tier.

A layer may also be able to run in multiple tiers. For example, in Diagram 2 below, the persistence layer in .NET can include two parts: persistence Lib and WCF data service, the persistence lib in the persistence layer always runs in the same process as business layer to adapt the business layer to the WCF data service. However, the WCF data service in persistence layer can run in a separate individual tier. Here is another example: we may extract the data validation in business layer into a separate library (but still kept in business layer), which can be called by client presenter layer directly for a better client-side interactive performance. If this occurs, then data validation part of the business layer runs in the same process of the client presenter layer, the rest of business layer runs in a separate tier.

#### Tier And Process

If a layer can run in an individual process, usually it will also be able to run in an individual computer (tier), hence it can be considered capable for an individual tier in N-Tier architecture. However, this isn’t always true. For example, assume that there are two layers which are implemented to run in two individual processes; they communicate with each other too. However, if these two layers are implemented in a such way that their IPC (inter-process communication) is solely based on a non-distributed way, such as the local shared memory, then these two layers can run in two different processes only in the same computer, not in two different computers. Unless there is another alternative distributed IPC way (such as socket) available for these two layers, these two layers will be considered capable for only one tier even though they can run in two different processes of the same computer.

#### Layer and Process

A layer may run in an individual process; several layer may also run in an individual process; a layer may run in several processes too. If you read above section “Tier and Layer Relationship”, you can understand here easily.

### 3-Tier Architecture

We introduce the 3-Tier concept first so that we can understand other tier concepts later easily. The simplest of N-Tier architecture is 3-Tier which typically contain following software component layers listed from the top level to the low level: presentation layer, application layer and data layer, which are depicted in Diagram 1.

A layer can access directly only the public components of its directly-below layer. For example, presentation layer can only access the public components in application layer, but not in data layer. Application layer can only access the public components in data layer, but not in presentation layer. Doing so can minimize the dependencies of one layer on other layers. This dependency minimization will bring benefits for layer development/maintenance, upgrading, scaling and etc. Doing so also makes the tier security enforcement possible. For example, the client layer cannot access the data layer directly but through the application layer, so data layer has a higher security guarding. Finally, doing so can also avoid cyclic dependencies among software components.

In order to claim a complete 3-Tier architecture, all three layers should be able to run in separate computers.

***Diagram 1: 3-Tier Architecture***

These three layers are briefly described as below:

**Presentation layer**: a layer that users can access directly, such as desktop UI, web page and etc. Also called client.

**Application layer**: this layer encapsulates the business logic (such as business rules and data validation), domain concept, data access logic and etc. Also called middle layer.

**Data layer**: the external data source to store the application data, such as database server, CRM system, ERP system, mainframe or other legacy systems and etc. The one we meet often today is database server. For N-Tier architecture, we need to use the non-embedded database server, such as SQL server, Oracle, DB2, MySQL or PostgreSQL. The non-embedded database server can be run in an individual computer. Whereas, the embedded type databases, such as Microsoft access, dbase and etc, cannot run in an individual computer, and then cannot be used as the data layer of the 3-Tier architecture.

### 1, 2, 3 or More Tier Architecture

**1-Tier**: all above layers can only run in one computer. In order to achieve 1-Tier, we need to use the embedded database system, which cannot run in an individual process. Otherwise, there will be at least 2-Tier because non-embedded databases usually can run in an individual computer (tier).

**2-Tier**: either presentation layer and application layer can only run in one computer, or application layer and data layer can only run in one computer. The whole application cannot run in more than 2 computers.

**3-Tier**: the simplest case of N-Tier architecture; all above three layers are able to run in three separate computers. Practically, these three layers can also be deployed in one computer (3-Tier architecture, but deployed as 1-Tier).

**N-Tier**: 3 or more tiers architecture. Diagram 2 below depicts a typical N-Tier architecture. Some layers in 3-Tier can be broken further into more layers. These broken layers may be able to run in more tiers. For example, application layer can be broken into business layer, persistence layer or more. Presentation layer can be broken into client layer and client presenter layer. In diagram 2, in order to claim a complete N-Tier architecture, client presenter layer, business layer and data layer should be able to run in three separate computers (tiers). Practically, all these layers can also be deployed in one compute (tier).

***Diagram 2: N-Tier Architecture***

Below are brief summaries on all layers in Diagram 2:

***Client layer***: this layer is involved with users directly. There may be several different types of clients coexisting, such as WPF, Window form, HTML web page and etc.

***Client presenter layer***: contains the presentation logic needed by clients, such as ASP .NET MVC in IIS web server. Also it adapts different clients to the business layer.

**Business layer**: handles and encapsulates all of business domains and logics; also called domain layer.

**Persistence layer**: handles the read/write of the business data to the data layer, also called data access layer (DAL).

**Data layer**: the external data source, such as a database.

Sometimes, the number of tiers is able to be equal or more than 3, but client presenter layer, business layer and data layer cannot run in three separate computers (tiers). Is this a N-Tier architecture? we categorize this N-Tier as an incomplete N-Tier architecture because its client presenter layer, business layer and data layer cannot run in three separate computers (tiers).

If we use the modem non-embedded database such as Sql Server, Oracle and etc, these databases will always be able to run in an individual computer. Therefore, for this case in Diagram 1, the criteria of a 2-Tier architecture is that presentation layer and application layer can run in only one computer; the criteria of a complete 3-Tier architecture is that presentation layer and application layer can run in different computers. A complete N-Tier architecture has the same criteria as 3-Tier.

## **Advantages and Disadvantages of Different Tier Architectures**

### 1 or 2-Tier Architecture

**Advantages**: simple and fast for a lower number of users due to fewer processes and fewer tiers; low cost for hardware, network, maintenance and deployment due to less hardware and network bandwidth needed.

**Disadvantages**: will have issues when the number of users gets big; has limitation to solve issues like security, scalability, fault tolerance and etc because it can be deployed in only 1 or 2 computes.

### N-Tier Architecture

**Advantages**: there are following general advantages:

1. Scalable: this is due to its capability of multiple tier deployment and the tier decoupling it brought. For example, the data tier can be scaled up by database clustering without other tiers involving. The web client side can be scaled up by load-balancer easily without affecting other tiers. Windows server can be clustered easily for load balancing and failover. In addition, business tier server can also be clustered to scale up the application, such as Weblogic cluster in J2EE.
2. Better and finer security control to the whole system: we can enforce the security differently for each tier if the security requirement is different for each tier. For example, business tier and data tier usually need higher security level than presentation tier does, then we can put these two high security tiers behind firewall for protection. 1 or 2 tiers architecture cannot fully achieve this purpose because of a limited number of tiers. Also, for N-Tier architecture, users cannot access business layer and data layer directly, all requests from users are routed by client presenter layer to business layer, then to data layer. Therefore, client presenter layer also serves as a proxy-like layer for business layer, and business layer serves as a proxy-like layer for data layer. These proxy-like layers provides further protection for their layers below.
3. Better fault tolerance ability: for example, the databases in data layer can be clustered for failover or load balance purpose without affecting other layers.
4. Independent tier upgrading and changing without affecting other tiers: in object-oriented world, Interface-dependency implementation can decouples all layers very well so that each layer can change individually without affecting other layers too much. Interface-dependency means a layer depends on another layer by interfaces only, not concrete classes. Also, the dependency of a layer only on its directly-below layer also minimizes the side effect of a layer’s change on the whole system. For example, if keep the interfaces unchanged, we can update or replace the implementation of any layer independently without affecting the whole system. Due to the changing of business requirement and technology, changing the implementation of a layer to another totally different one does happen often. For example, originally we use Windows Form mainly, now we use WPF mainly. If our original system is implemented as the decoupled layer structure, then we will only need to update the client side from Windows Form to WPF without the need to change the server side layers.
5. Friendly and efficient for development: the decoupled layers are logic software component groups mainly by functionality, they are very software development friendly and efficient. Each layer can be assigned individually to a team who specializes in the specific functional area; a specialized team can handle the relevant task better and more efficiently.
6. Friendly for maintenance: N-Tier architecture groups different things together mainly by functionality and then makes things clear, easily understandable and manageable.
7. Friendly for new feature addition: due to the logical grouped components and the decoupling brought by N-Tier architecture, new features can be added easily without affecting too much on the whole system.
8. Better reusability: this is due to the logically grouped components and the loose couplings among layers. Loosely-coupled component groups are usually implemented in more general ways, so they can be reused by more other applications.

#### The Disadvantages of the N-Tier Deployment

1. The performance of the whole application may be slow if the hardware and network bandwidth aren’t good enough because more networks, computers and processes are involved.
2. More cost for hardware, network, maintenance and deployment because more hardware and better network bandwidth are needed.

The effects of N-Tier deployment on the application performance are a double edge issue. In one side, if the number of uses isn’t big enough, the performance may be slow due to more computers, process and network involved. Namely, if put everything in one tier or one process, performance will be better for a small number of users. However, if the number of user gets big, then the scalability brought by N-Tier will improve the overall performance, such as load balancing and database clustering which all improve the performance of N-Tier architecture. Why the performance result of the small number of users and the big number of users are different? this is because the bottleneck of the whole application are different for these two situations. For the case with the small number users, the bottleneck is the time for data to communicate among different processes. If more computer, more processes and longer network, then costs longer, then performance is bad. However, when the number of user gets big, the bottleneck shift to other things because of server’s capacity, such as cpu and memory resource contentions in one computer, database threshold in server, horsepower limitation of a web server and etc. Only the scalability of N-Tier architecture can solve these bottlenecks existing with a big number of users; usually load balancing by server clustering is used for N-Tier architecture’s scalability. With more computers are scaled up to share the task for big number of users, then performance is improved.  Besides gaining performance by scalability of N-Tier architecture, we can also improve the performance with better hardware and better network bandwidth to meet our business requirement.

## **Business Data Validation in N-Tier Architecture**

Data validation is important and a MUST in N-Tier architecture in order to keep the whole business system healthy and integral. The first question for business data validation will be: where or which layer should handle the data validation? There are some rules and facts as below for the business data validation, which will give us some tips and can answer this question too:

1. Data validation can be checked in any layer. Usually, the closer to the client layer the validation is, the more efficient the performance is. The farther to the client layer the validation is, the more reliable and robust the application is. When the validation is checked in business layer or persistence layer, it is guaranteed that every type client will get the validation regardless whether or not the client side will check the validation.
2. When we decide which layer should do the validation, we need to achieve a balance result between performance, reliability and robustness, also we need to make decisions based on the actual situation. If we have full control of the all layers, we can let all validations happen in client/client presenter layers only to gain performance. However, if the business layer is also exposed to some client/client presenter layers which is out of our control, then the business or lower layer must do all validations to gain reliability regardless of whether or not our clients do the same validation.
3. Client side validation is efficient, such as Javascript validation in a web page. However, users may bypass the client side validation easily and intentionally, such as webpage hacking. Therefore, it is needed to do data validation in both client side and server side to achieve both performance and reliability. The business layer and other further lower layers usually belong to the server side. Client presenter layer may or may not be in the server side; a webserver client presenter layer such as ASP.NET is in the server side. Client presenter layer for WPF may not be in the server side.
4. A more practical way is to do the simple data validation in client side for performance and further do a full validation in server side for reliability. The simple data validation is mainly the single property checking of an entity instance. The full validation includes the simple data validation and some complex data validation. The complex data validation can include class-level data validation which crosses multiple properties of an entity instance and the data validation which crosses multiple entity instances of similar or different types.
5. For some interactive client application, we need to do client side validation anyway for acceptable interactive performance, regardless of whether or not we will do the validation in server side.  Some game applications belong to this category.
6. We should implement and maintain one version of validation logics in one place, regardless of where the validation will be checked. All layers should share this one version of validation logics. Why? Doing so has better reusability; it can avoid duplicate and conflicting validation logic in many places and makes the development, maintenance and deployment easier; it keeps the whole validation logics consistent throughout the whole application. In addition, the places to check validation may change with the changing and growing of the business, so validation should be kept in one place with one version but flexible to be called by any layer if needed.

## **How to Deploy N-Tier Application Correctly**

More tiers brings extra complexity, extra deployment/maintenance effort and extra cost. Therefore, the number of tiers should be kept as minimal as enough to solve issues like the scalability, security, failover and etc. If these issues are solved as needed, don’t deploy more tiers further. But, in order to solve these issues as better as possible, usually 3-Tier will be needed at least. If these issues aren’t concerned at all in certain cases, then we can select 1 or 2-Tier architecture or 1 or 2-Tier deployment of N-Tier architecture to gain performance. What is the best number of tiers? no fixed answer. In order to meet our business requirement, we need to select the number of tiers to achieve a best balance result among those good and bad things of N-Tier architecture.

We should differ the following two cases: a) all layers run in one process of a computer, and b) all layers run in different processes of a computer. The 1st case is actually 1 tier architecture; the 2nd case is usually N-Tier architecture but deployed in just 1 tier.  Even both are in one computer, the 1 tier architecture will have better performance because of fewer processes involved.  Compared to one process, communication crossing process boundaries is more complex and slower, regardless what type of IPC (inter-process communication) technique is used: TCP/IP, named pipe, message queue or shared memory and etc. Therefore, in a deployed computer, we need to keep the number of the application processes as few as possible to gain performance. How to achieve this? the N-Tier architecture can be implemented in such a way that switching among different tier architectures is as easy as updating the configuration file only. This is introduced in detail in our sample application article: [A N-Tier Architecture Sample with ASP.NET MVC3, WCF and Entity Framework](http://www.codeproject.com/Articles/434282/A-N-Tier-Architecture-Sample-with-ASP-NET-MVC3-WCF).

Practically, there are many variations of N-Tier architecture; they are existing for reasons. For example, one possibility is to put the client presenter layer and business layer in the same process to achieve a better interactive performance. You can explore and research this topic further by yourself if you are interested in it.

In addition, 3-Tier architecture can be deployed as 3 or fewer tiers. But a 2-Tier architecture cannot be deployed as 3-Tier; otherwise it should be called 3-Tier architecture, not 2-Tier.

## **How to Achieve N-Tier Deployment Capability by Software Technologies**

The main characteristic of N-Tier is the ability to deploy one or more layers in different computers to deal with issues like scalability, security, fault tolerance and etc; two related tiers need to communicate with each other. How to achieve this? Application running in a tier is in processes. So, communication between two tiers actually comes down to IPC (Inter-process Communication) issue. A distributed IPC method can support two processes in two different computers to communicate with each other, such as socket, distributed message queue and etc. So, natively, these distributed IPC methods can support N-Tier deployment capability. For example, if two layers are implemented with TCP/IP socket as the communication way, these two layers can be deployed in two different computers (tiers) to communicate with each other. In .NET, WCF can achieve N-Tier deployment requirement easily since WCF supports communication among processes - either on the same computer or different computer; WCF is built on top of the basic IPC methods. Another advantage to use WCF for N-Tier architecture is that WCF is able to achieve SOA (service oriented architecture) result with a very loosely-coupled dependency among layers.

## **Some Practical Tips on N-Tier Architecture Development**

Design, implement, deploy and maintain a N-Tier architecture are daunting tasks. If you don’t have clear top thoughts in the beginning, you probably end up with a lot of time wasting due to detouring and twisting here and there. We already talks some tips on deployment and data validation above, here, we will provide some extra practical tips on N-Tier architecture development. The tips in this article are based on the assumption that a team has a full control over all layers of the N-Tier architecture.

1. Decouple a layer from another layer as much as possible by some loosely coupling techniques, such as soap xml and interface etc. In object-oriented world, each layer should depend on its immediate lower layer by interfaces only, not by concrete classes. By doing so, we can achieve the maximum decoupling between two layers, this decoupling will bring many benefits for development, unit test, maintenance, upgrading, inter-changeability, reusability and etc.
2. Try as much as possible to auto-generate and maintain only one version poco business entity classes, which can be reused in the whole application, why? Business entity classes are the base of the N-Tier architecture, they convey the info from the topest layer to the lowest layer. Modern application tends to continue to grow bigger and bigger, therefore, manually creating the mass entity classes is daunting and error-prone, particularly for different versions in different layers as some people prefer. So, we suggest that in the whole application, we should try to use only one lightweight POCO version of entity classes, which should be auto-generated by code generators. Doing so will save us a lot of efforts, also wipe out the headache of the mappings and the inconsistency of different versions of entity classes. There are many code generators available nowadays for doing this, such as code generators for Entity Framework. Certainly, different layers may have different requirement on entity classes, if so, we can use annotation feature in System.ComponentModel.Annotation and partial class feature in C# to restrict or expand these auto-generated entity classes for our special needs in a specific layer.  If there are compelling reasons that we must use difference versions for certain entity classes in different layer, we can use data transfer object (DTO) to map these specific entities only, and still keep the majority of the entity classes one version.
3. Use some auto tools or packages to generate the mappings between business entity classes and a traditional relational database (data layer). Modem business and database get bigger and bigger, so manually create these mappings isn’t easy and is error-prone. There are many existing packages or tools to help us,  such as Entity Framework and NHibernate for .NET, and Hibernate for Java.
4. Use code generators as much as possible further for other massive and style-similar code. If you cannot find any existing code generator for your purpose, develop one for yourself. By using an object-oriented programming  languages (like C# or Java) together with XSLT (Extensible Stylesheet Language Transformations), it won’t be harder to develop a code generator for your special need. XSLT is xml-based, very helpful and flexible for code generator purpose because it can convert any xml document into any text document easily.
5. Business layer is easily prone to be tightly coupled with persistence layer; we should avoid this. For example, in .NET, a WCF business service may access Entity Framework directly. This situation is very popular. However, doing so has issue that the business layer and the persistence layer are tightly coupled. This tightly coupling will bring many issues on layers for unit-testing, upgrading, inter-changeability and etc. Usually we need an adapter layer between these two so that they will be loosely coupled by interface only. There is a physical sample for this in our N-Tier sample article: [A N-Tier Architecture Sample with ASP.NET MVC3, WCF and Entity Framework](http://www.codeproject.com/Articles/434282/A-N-Tier-Architecture-Sample-with-ASP-NET-MVC3-WCF).
6. In client presenter layer, we should put all common code for all clients to a separate library as much as possible to maximize the code reusability for all types of clients.
7. A cache layer can be added into any existing layer to speed up performance. For example, Varnish accelerator can be used for ASP .Net, Drupal or other web applications as a cache layer sitting between client layer and client presenter layer to speed up performance. Memcached and APC cache are PHP cache packages which can be added in many PHP layers as extra layer for caching business data. Requests will go to cache layer first, if valid data is in cache, then request will return and won’t go to the lower layer further, then performance is improved. Usually updating or expiration of data will invalidate the old data in a cache. In .NET 4, namespace System.Web.Caching can be used for caching in ASP.NET; namespace System.Runtime.Caching can be used for caching in any place; the Caching Application Block in Enterprise Library is also an option for caching.
8. In order to adapt to the changeable business requirements and technologies, it is good to implement N-Tier architecture in such a way that it can be flexible to any type of deployment easily, including 2-Tier architecture deployment. For example, the N-Tier architecture can be implemented in such a way that different tier architectures can be switched simply by some parameter-value updating in configuration files. There is a physical sample implementation for this in our N-Tier sample article: [A N-Tier Architecture Sample with ASP.NET MVC3, WCF and Entity Framework](http://www.codeproject.com/Articles/434282/A-N-Tier-Architecture-Sample-with-ASP-NET-MVC3-WCF).

## **Conclusions**

1. A complete 3-Tier architecture should be able to run its presentation layer, application layer and data layer in 3 separate computers (refer to Diagram 1). A complete N-Tier architecture should be able to run its client presenter layer, business layer and data layer in at least 3 separate computers (refer to Diagram 2). A layer can only access directly the public components of its directly-below layer; doing so can minimize the dependencies of one layer on other layers and also makes the tier security enforcement possible. The complete N-Tier architecture has the best capability to handle issues like scalability, security, fault tolerance and etc.
2. Tier usually means physical deployment computer; layer usually means logic software component group mainly by functionality. Layer implementation is a common and best way to achieve a N-Tier architecture. Tier and layer may or may not exactly match. A layer may run as an individual tier; a layer may also be able to run in multiple tiers; multiple layers may also run as one tier only.
3. If a layer can run in an individual process, usually it will also be able to run in an individual computer (tier). However, this isn’t always true. If this process communicates with other layer processes only by a non-distributed IPC (inter-process communication) method, this process won't be able to run in an individual computer (tier).
4. N-Tier architecture has following advantages: better scalability, better and finer security control, better fault tolerance ability, independent tier upgrading and changing ability without affecting other tiers, friendly and efficient development, friendly maintenance, friendly new feature addition, better reusability and etc.
5. If a modem non-embedded database (such as Sql Server, Oracle and etc) is used as the data layer, this database will always be able to run in an individual tier. For this case in Diagram 1, the criteria of 2-Tier architecture is that presentation layer and application layer can only run in one computer; the criteria of a complete 3-Tier or other complete N-Tier architecture is that presentation layer and application layer can run in separate computers.
6. More tiers brings extra complexity, extra deployment/maintenance effort and extra cost. Therefore, the number of tiers should be kept as minimal as enough to solve issues like the scalability, security, failover and etc. If these issues are solved as needed, don’t deploy more tiers further. In addition, in a deployed computer, we need to keep the number of processes as few as possible to gain performance; good architecture design can achieve this easily.
7. Distributed IPC (inter-process communication) methods (such as socket) can make a layer’s process deployable in a tier. In .NET, WCF is a good way to achieve the deployment capability of N-Tier architecture.
8. There are following facts and rules on data validation in N-Tier architecture:
   1. Data validation can be checked in any layer. Usually, the closer to the client layer the validation is, the more efficient the performance is. The farther to the client layer the validation is, the more reliable and robust the application is. When the validation is checked in business layer or persistence layer, it is guaranteed that every type client will get the validation regardless whether or not the client side will check the validation.
   2. When we decide which layer should do the validation, we need to achieve a balance result between performance, reliability and robustness, also we need to make decisions based on the actual situation.
   3. Client side validation is efficient, such as Javascript validation in a web page. However, users may bypass the client side validation easily and intentionally, such as webpage hacking. Therefore, it is needed to do data validation in both client side and server side to achieve both performance and reliability.
   4. A more practical way is to do the simple data validation in client side for performance and further do a full validation in server side for reliability. The simple data validation is mainly the single property checking of an entity instance. The full validation includes the simple data validation and some complex data validation crossing multiple properties of an entity instance or crossing multiple entity instances of similar or different types.
   5. For some interactive client application, we need to do client side validation anyway for acceptable interactive performance, regardless of whether or not we will do the validation in server side.  Some game applications belong to this category.
   6. We should implement and maintain one version of validation logics in one place, regardless of where the validation will be checked. All layers should share this one version of validation logics. Doing so has better reusability and makes the development, maintenance and deployment easier, and also keeps the validation logics consistent throughout the whole application.
9. Below are some extra practical tips on N-Tier architecture development, based on the assumption that a team has its full control over all layers:
   1. Decouple a layer from another layer as much as possible by some loosely coupling techniques, such as soap xml and interface etc. In object-oriented world, each layer should depend on its immediate lower layer by interfaces only, not  by concrete classes. By doing so, we can achieve the maximum decoupling between two layers, this decoupling will bring many benefits for development, unit test, maintenance, upgrading, inter-changeability, reusability and etc.
   2. Try as much as possible to auto-generate and maintain only one version poco business entity classes, which can be used in the whole application. This will reduce all hassles for version conflicting, mapping, errors from manual coding and etc. In some special cases, we can mix the one version of entity classes with some data transfer objects (DTO) which map with some special entity classes.
   3. Use some auto tools or packages to generate the mappings between business entity classes and a traditional relational database (data layer), such as packages Entity Framework and NHibernate for .NET, and Hibernate for Java.
   4. Use code generators as much as possible further for other massive and style-similar code. If you cannot find any existing code generator for your purpose, develop one for yourself. C#, java, XSLT and etc can help to develop code generators easily.
   5. Business layer is easily prone to be tightly coupled with persistence layer; we should avoid this by an adapter layer between these two layers so that they will be loosely coupled by interface only, such as a persistence adapter between WCF business layer and Entity Framework.
   6. In client presenter layer,  we should put all common code for all clients to a separate library as much as possible to maximize the code reusability for all types of clients.
   7. A cache layer can be added into any existing layer to speed up performance.
   8. In order to adapt to the changeable business requirements and technologies, it is good to implement N-Tier architecture in such a way that different tier architectures can be switched simply by some parameter-value updating in configuration files.

# **ASP.NET Page Life Cycle Overview**

[Other Versions](javascript:;)

https://i-msdn.sec.s-msft.com/Areas/Epx/Content/Images/ImageSprite.png?v=635949642794941893

When an ASP.NET page runs, the page goes through a life cycle in which it performs a series of processing steps. These include initialization, instantiating controls, restoring and maintaining state, running event handler code, and rendering. It is important for you to understand the page life cycle so that you can write code at the appropriate life-cycle stage for the effect you intend.

If you develop custom controls, you must be familiar with the page life cycle in order to correctly initialize controls, populate control properties with view-state data, and run control behavior code. The life cycle of a control is based on the page life cycle, and the page raises many of the events that you need to handle in a custom control.

This topic contains the following sections:

* [General Page Life-cycle Stages](https://msdn.microsoft.com/en-us/library/ms178472.aspx#general_page_lifecycle_stages)
* [Life-cycle Events](https://msdn.microsoft.com/en-us/library/ms178472.aspx#lifecycle_events)
* [Additional Page Life Cycle Considerations](https://msdn.microsoft.com/en-us/library/ms178472.aspx#additional_page_life_cycle_considerations)
* [Catch-Up Events for Added Controls](https://msdn.microsoft.com/en-us/library/ms178472.aspx#catch_up_events_for_added_controls)
* [Data Binding Events for Data-Bound Controls](https://msdn.microsoft.com/en-us/library/ms178472.aspx#data_binding_events_for_databound_controls)
* [Login Control Events](https://msdn.microsoft.com/en-us/library/ms178472.aspx#login_control_events)

## [**General Page Life-Cycle Stages**](javascript:void(0))

In general terms, the page goes through the stages outlined in the following table. In addition to the page life-cycle stages, there are application stages that occur before and after a request but are not specific to a page. For more information, see [Introduction to the ASP.NET Application Life Cycle](http://go.microsoft.com/fwlink/?LinkId=133108) and [ASP.NET Application Life Cycle Overview for IIS 7.0](https://msdn.microsoft.com/en-us/library/bb470252.aspx).

Some parts of the life cycle occur only when a page is processed as a postback. For postbacks, the page life cycle is the same during a partial-page postback (as when you use an [UpdatePanel](https://msdn.microsoft.com/en-us/library/system.web.ui.updatepanel.aspx) control) as it is during a full-page postback.

|  |  |
| --- | --- |
| **Stage** | **Description** |
| Page request | The page request occurs before the page life cycle begins. When the page is requested by a user, ASP.NET determines whether the page needs to be parsed and compiled (therefore beginning the life of a page), or whether a cached version of the page can be sent in response without running the page. |
| Start | In the start stage, page properties such as [Request](https://msdn.microsoft.com/en-us/library/system.web.ui.page.request.aspx) and [Response](https://msdn.microsoft.com/en-us/library/system.web.ui.page.response.aspx) are set. At this stage, the page also determines whether the request is a postback or a new request and sets the [IsPostBack](https://msdn.microsoft.com/en-us/library/system.web.ui.page.ispostback.aspx) property. The page also sets the [UICulture](https://msdn.microsoft.com/en-us/library/system.web.ui.page.uiculture.aspx) property. |
| Initialization | During page initialization, controls on the page are available and each control's [UniqueID](https://msdn.microsoft.com/en-us/library/system.web.ui.control.uniqueid.aspx) property is set. A master page and themes are also applied to the page if applicable. If the current request is a postback, the postback data has not yet been loaded and control property values have not been restored to the values from view state. |
| Load | During load, if the current request is a postback, control properties are loaded with information recovered from view state and control state. |
| Postback event handling | If the request is a postback, control event handlers are called. After that, the [Validate](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.basevalidator.validate.aspx) method of all validator controls is called, which sets the [IsValid](https://msdn.microsoft.com/en-us/library/system.web.ui.ivalidator.isvalid.aspx) property of individual validator controls and of the page. (There is an exception to this sequence: the handler for the event that caused validation is called after validation.) |
| Rendering | Before rendering, view state is saved for the page and all controls. During the rendering stage, the page calls the [Render](https://msdn.microsoft.com/en-us/library/system.web.ui.control.render.aspx)method for each control, providing a text writer that writes its output to the [OutputStream](https://msdn.microsoft.com/en-us/library/system.web.httpresponse.outputstream.aspx) object of the page's [Response](https://msdn.microsoft.com/en-us/library/system.web.ui.page.response.aspx)property. |
| Unload | The [Unload](https://msdn.microsoft.com/en-us/library/system.web.ui.control.unload.aspx) event is raised after the page has been fully rendered, sent to the client, and is ready to be discarded. At this point, page properties such as [Response](https://msdn.microsoft.com/en-us/library/system.web.ui.page.response.aspx) and [Request](https://msdn.microsoft.com/en-us/library/system.web.ui.page.request.aspx) are unloaded and cleanup is performed. |

## [**Life-Cycle Events**](javascript:void(0))

Within each stage of the life cycle of a page, the page raises events that you can handle to run your own code. For control events, you bind the event handler to the event, either declaratively using attributes such as **onclick**, or in code.

Pages also support automatic event wire-up, meaning that ASP.NET looks for methods with particular names and automatically runs those methods when certain events are raised. If the **AutoEventWireup** attribute of the [@ Page](https://msdn.microsoft.com/en-us/library/ydy4x04a.aspx) directive is set to **true**, page events are automatically bound to methods that use the naming convention of **Page\_***event*, such as **Page\_Load** and **Page\_Init**. For more information on automatic event wire-up, see[ASP.NET Web Forms Server Control Event Model](https://msdn.microsoft.com/en-us/library/y3bwdsh3.aspx).

The following table lists the page life-cycle events that you will use most frequently. There are more events than those listed; however, they are not used for most page-processing scenarios. Instead, they are primarily used by server controls on the ASP.NET Web page to initialize and render themselves. If you want to write custom ASP.NET server controls, you need to understand more about these events. For information about creating custom controls, see [Developing Custom ASP.NET Server Controls](https://msdn.microsoft.com/en-us/library/zt27tfhy.aspx).

|  |  |
| --- | --- |
| **Page Event** | **Typical Use** |
| [PreInit](https://msdn.microsoft.com/en-us/library/system.web.ui.page.preinit.aspx) | Raised after the start stage is complete and before the initialization stage begins.  Use this event for the following:   * Check the [IsPostBack](https://msdn.microsoft.com/en-us/library/system.web.ui.page.ispostback.aspx) property to determine whether this is the first time the page is being processed. The [IsCallback](https://msdn.microsoft.com/en-us/library/system.web.ui.page.iscallback.aspx)and [IsCrossPagePostBack](https://msdn.microsoft.com/en-us/library/system.web.ui.page.iscrosspagepostback.aspx) properties have also been set at this time. * Create or re-create dynamic controls. * Set a master page dynamically. * Set the [Theme](https://msdn.microsoft.com/en-us/library/system.web.ui.page.theme.aspx) property dynamically. * Read or set profile property values.  |  | | --- | | **Note Note** | | If the request is a postback, the values of the controls have not yet been restored from view state. If you set a control property at this stage, its value might be overwritten in the next event. | |
| [Init](https://msdn.microsoft.com/en-us/library/system.web.ui.control.init.aspx) | Raised after all controls have been initialized and any skin settings have been applied. The [Init](https://msdn.microsoft.com/en-us/library/system.web.ui.control.init.aspx) event of individual controls occurs before the [Init](https://msdn.microsoft.com/en-us/library/system.web.ui.control.init.aspx) event of the page.  Use this event to read or initialize control properties. |
| [InitComplete](https://msdn.microsoft.com/en-us/library/system.web.ui.page.initcomplete.aspx) | Raised at the end of the page's initialization stage. Only one operation takes place between the [Init](https://msdn.microsoft.com/en-us/library/system.web.ui.control.init.aspx) and [InitComplete](https://msdn.microsoft.com/en-us/library/system.web.ui.page.initcomplete.aspx) events: tracking of view state changes is turned on. View state tracking enables controls to persist any values that are programmatically added to the [ViewState](https://msdn.microsoft.com/en-us/library/system.web.ui.control.viewstate.aspx) collection. Until view state tracking is turned on, any values added to view state are lost across postbacks. Controls typically turn on view state tracking immediately after they raise their [Init](https://msdn.microsoft.com/en-us/library/system.web.ui.control.init.aspx) event.  Use this event to make changes to view state that you want to make sure are persisted after the next postback. |
| [PreLoad](https://msdn.microsoft.com/en-us/library/system.web.ui.page.preload.aspx) | Raised after the page loads view state for itself and all controls, and after it processes postback data that is included with the[Request](https://msdn.microsoft.com/en-us/library/system.web.ui.page.request.aspx) instance. |
| [Load](https://msdn.microsoft.com/en-us/library/system.web.ui.control.load.aspx) | The [Page](https://msdn.microsoft.com/en-us/library/system.web.ui.page.aspx) object calls the [OnLoad](https://msdn.microsoft.com/en-us/library/system.web.ui.control.onload.aspx) method on the [Page](https://msdn.microsoft.com/en-us/library/system.web.ui.page.aspx) object, and then recursively does the same for each child control until the page and all controls are loaded. The [Load](https://msdn.microsoft.com/en-us/library/system.web.ui.control.load.aspx) event of individual controls occurs after the [Load](https://msdn.microsoft.com/en-us/library/system.web.ui.control.load.aspx) event of the page.  Use the [OnLoad](https://msdn.microsoft.com/en-us/library/system.web.ui.control.onload.aspx) event method to set properties in controls and to establish database connections. |
| Control events | Use these events to handle specific control events, such as a [Button](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.button.aspx) control's [Click](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.button.click.aspx) event or a [TextBox](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.textbox.aspx) control's [TextChanged](https://msdn.microsoft.com/en-us/library/system.web.ui.mobilecontrols.textbox.textchanged.aspx)event.   |  | | --- | | **Note Note** | | In a postback request, if the page contains validator controls, check the [IsValid](https://msdn.microsoft.com/en-us/library/system.web.ui.page.isvalid.aspx) property of the [Page](https://msdn.microsoft.com/en-us/library/system.web.ui.page.aspx) and of individual validation controls before performing any processing. | |
| [LoadComplete](https://msdn.microsoft.com/en-us/library/system.web.ui.page.loadcomplete.aspx) | Raised at the end of the event-handling stage.  Use this event for tasks that require that all other controls on the page be loaded. |
| [PreRender](https://msdn.microsoft.com/en-us/library/system.web.ui.control.prerender.aspx) | Raised after the [Page](https://msdn.microsoft.com/en-us/library/system.web.ui.page.aspx) object has created all controls that are required in order to render the page, including child controls of composite controls. (To do this, the [Page](https://msdn.microsoft.com/en-us/library/system.web.ui.page.aspx) object calls [EnsureChildControls](https://msdn.microsoft.com/en-us/library/system.web.ui.control.ensurechildcontrols.aspx) for each control and for the page.)  The [Page](https://msdn.microsoft.com/en-us/library/system.web.ui.page.aspx) object raises the [PreRender](https://msdn.microsoft.com/en-us/library/system.web.ui.control.prerender.aspx) event on the [Page](https://msdn.microsoft.com/en-us/library/system.web.ui.page.aspx) object, and then recursively does the same for each child control. The[PreRender](https://msdn.microsoft.com/en-us/library/system.web.ui.control.prerender.aspx) event of individual controls occurs after the [PreRender](https://msdn.microsoft.com/en-us/library/system.web.ui.control.prerender.aspx) event of the page.  Use the event to make final changes to the contents of the page or its controls before the rendering stage begins. |
| [PreRenderComplete](https://msdn.microsoft.com/en-us/library/system.web.ui.page.prerendercomplete.aspx) | Raised after each data bound control whose [DataSourceID](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.databoundcontrol.datasourceid.aspx) property is set calls its [DataBind](https://msdn.microsoft.com/en-us/library/system.web.ui.control.databind.aspx) method. For more information, see[Data Binding Events for Data-Bound Controls](https://msdn.microsoft.com/en-us/library/ms178472.aspx#data_binding_events_for_databound_controls) later in this topic. |
| [SaveStateComplete](https://msdn.microsoft.com/en-us/library/system.web.ui.page.savestatecomplete.aspx) | Raised after view state and control state have been saved for the page and for all controls. Any changes to the page or controls at this point affect rendering, but the changes will not be retrieved on the next postback. |
| [Render](https://msdn.microsoft.com/en-us/library/system.web.ui.control.render.aspx) | This is not an event; instead, at this stage of processing, the [Page](https://msdn.microsoft.com/en-us/library/system.web.ui.page.aspx) object calls this method on each control. All ASP.NET Web server controls have a [Render](https://msdn.microsoft.com/en-us/library/system.web.ui.control.render.aspx) method that writes out the control's markup to send to the browser.  If you create a custom control, you typically override this method to output the control's markup. However, if your custom control incorporates only standard ASP.NET Web server controls and no custom markup, you do not need to override the[Render](https://msdn.microsoft.com/en-us/library/system.web.ui.control.render.aspx) method. For more information, see [Developing Custom ASP.NET Server Controls](https://msdn.microsoft.com/en-us/library/zt27tfhy.aspx).  A user control (an .ascx file) automatically incorporates rendering, so you do not need to explicitly render the control in code. |
| [Unload](https://msdn.microsoft.com/en-us/library/system.web.ui.control.unload.aspx) | Raised for each control and then for the page.  In controls, use this event to do final cleanup for specific controls, such as closing control-specific database connections.  For the page itself, use this event to do final cleanup work, such as closing open files and database connections, or finishing up logging or other request-specific tasks.   |  | | --- | | **Note Note** | | During the unload stage, the page and its controls have been rendered, so you cannot make further changes to the response stream. If you attempt to call a method such as the **Response.Write** method, the page will throw an exception. | |

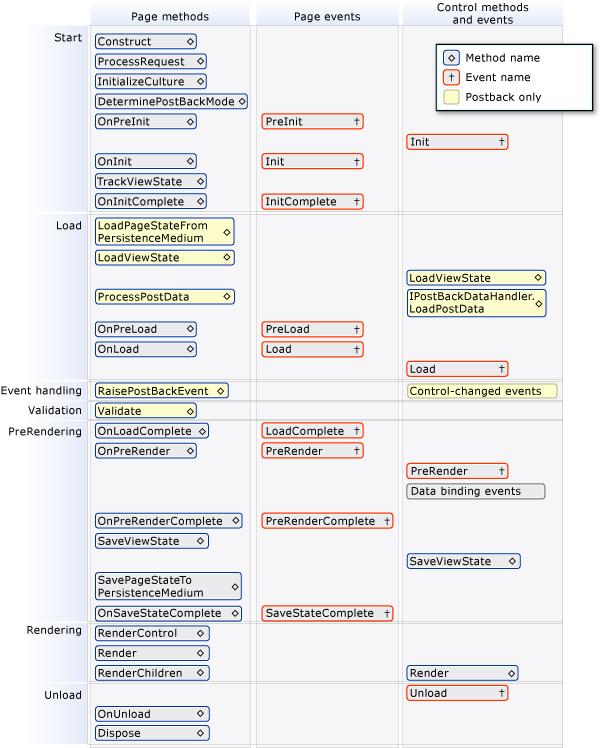
## [**Additional Page Life Cycle Considerations**](javascript:void(0))

Individual ASP.NET server controls have their own life cycle that is similar to the page life cycle. For example, a control's [Init](https://msdn.microsoft.com/en-us/library/system.web.ui.control.init.aspx) and [Load](https://msdn.microsoft.com/en-us/library/system.web.ui.control.load.aspx) events occur during the corresponding page events.

Although both [Init](https://msdn.microsoft.com/en-us/library/system.web.ui.control.init.aspx) and [Load](https://msdn.microsoft.com/en-us/library/system.web.ui.control.load.aspx) recursively occur on each control, they happen in reverse order. The [Init](https://msdn.microsoft.com/en-us/library/system.web.ui.control.init.aspx) event (and also the [Unload](https://msdn.microsoft.com/en-us/library/system.web.ui.control.unload.aspx) event) for each child control occur before the corresponding event is raised for its container (bottom-up). However the [Load](https://msdn.microsoft.com/en-us/library/system.web.ui.control.load.aspx) event for a container occurs before the [Load](https://msdn.microsoft.com/en-us/library/system.web.ui.control.load.aspx) events for its child controls (top-down). Master pages behave like child controls on a page: the master page [Init](https://msdn.microsoft.com/en-us/library/system.web.ui.control.init.aspx) event occurs before the page [Init](https://msdn.microsoft.com/en-us/library/system.web.ui.control.init.aspx) and [Load](https://msdn.microsoft.com/en-us/library/system.web.ui.control.load.aspx) events, and the master page [Load](https://msdn.microsoft.com/en-us/library/system.web.ui.control.load.aspx) event occurs after the page [Init](https://msdn.microsoft.com/en-us/library/system.web.ui.control.init.aspx) and [Load](https://msdn.microsoft.com/en-us/library/system.web.ui.control.load.aspx) events.

When you create a class that inherits from the [Page](https://msdn.microsoft.com/en-us/library/system.web.ui.page.aspx) class, in addition to handling events raised by the page, you can override methods from the page's base class. For example, you can override the page's [InitializeCulture](https://msdn.microsoft.com/en-us/library/system.web.ui.page.initializeculture.aspx) method to dynamically set culture information. Note that when an event handler is created using the **Page\_***event* syntax, the base implementation is implicitly called and therefore you do not need to call it in your method. For example, the base page class's [OnLoad](https://msdn.microsoft.com/en-us/library/system.web.ui.control.onload.aspx) method is always called, whether you create a **Page\_Load** method or not. However, if you override the page [OnLoad](https://msdn.microsoft.com/en-us/library/system.web.ui.control.onload.aspx) method with the **override** keyword (**Overrides** in Visual Basic), you must explicitly call the base method. For example, if you override the [OnLoad](https://msdn.microsoft.com/en-us/library/system.web.ui.control.onload.aspx) method on the page, you must call **base.Load** (**MyBase.Load** in Visual Basic) in order for the base implementation to be run.

The following illustration shows some of the most important methods of the [Page](https://msdn.microsoft.com/en-us/library/system.web.ui.page.aspx) class that you can override in order to add code that executes at specific points in the page life cycle. (For a complete list of page methods and events, see the [Page](https://msdn.microsoft.com/en-us/library/system.web.ui.page.aspx) class.) The illustration also shows how these methods relate to page events and to control events. The sequence of methods and events in the illustration is from top to bottom, and within each row from left to right.



## [**Catch-Up Events for Added Controls**](javascript:void(0))

If controls are created dynamically at run time or declaratively within templates of data-bound controls, their events are initially not synchronized with those of other controls on the page. For example, for a control that is added at run time, the [Init](https://msdn.microsoft.com/en-us/library/system.web.ui.control.init.aspx) and [Load](https://msdn.microsoft.com/en-us/library/system.web.ui.control.load.aspx) events might occur much later in the page life cycle than the same events for controls created declaratively. Therefore, from the time that they are instantiated, dynamically added controls and controls in templates raise their events one after the other until they have caught up to the event during which it was added to the[Controls](https://msdn.microsoft.com/en-us/library/system.web.ui.control.controls.aspx) collection.

## [**Data Binding Events for Data-Bound Controls**](javascript:void(0))

To help you understand the relationship between the page life cycle and data binding events, the following table lists data-related events in data-bound controls such as the [GridView](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.gridview.aspx), [DetailsView](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.detailsview.aspx), and [FormView](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.formview.aspx) controls.

|  |  |
| --- | --- |
| **Control Event** | **Typical Use** |
| [DataBinding](https://msdn.microsoft.com/en-us/library/system.web.ui.control.databinding.aspx) | Raised after the control's [PreRender](https://msdn.microsoft.com/en-us/library/system.web.ui.control.prerender.aspx) event, which occurs after the page's [PreRender](https://msdn.microsoft.com/en-us/library/system.web.ui.control.prerender.aspx) event. (This applies to controls whose [DataSourceID](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.databoundcontrol.datasourceid.aspx) property is set declaratively. Otherwise the event happens when you call the control's **DataBind** method.)  This event marks the beginning of the process that binds the control to the data. Use this event to manually open database connections, if required, and to set parameter values dynamically before a query is run. |
| [RowCreated](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.gridview.rowcreated.aspx) ([GridView](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.gridview.aspx) only) or [ItemCreated](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.datalist.itemcreated.aspx)([DataList](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.datalist.aspx), [DetailsView](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.detailsview.aspx), [SiteMapPath](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.sitemappath.aspx), [DataGrid](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.datagrid.aspx),[FormView](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.formview.aspx), [Repeater](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.repeater.aspx), and [ListView](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.listview.aspx) controls) | Raised after the control's [DataBinding](https://msdn.microsoft.com/en-us/library/system.web.ui.control.databinding.aspx) event.  Use this event to manipulate content that is not dependent on data binding. For example, at run time, you might programmatically add formatting to a header or footer row in a[GridView](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.gridview.aspx) control. |
| [RowDataBound](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.gridview.rowdatabound.aspx) ([GridView](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.gridview.aspx) only) or[ItemDataBound](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.datalist.itemdatabound.aspx) ([DataList](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.datalist.aspx), [SiteMapPath](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.sitemappath.aspx), [DataGrid](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.datagrid.aspx),[Repeater](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.repeater.aspx), and [ListView](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.listview.aspx) controls) | Raised after the control's [RowCreated](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.gridview.rowcreated.aspx) or [ItemCreated](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.datalist.itemcreated.aspx) event.  When this event occurs, data is available in the row or item, so you can format data or set the [FilterExpression](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.sqldatasource.filterexpression.aspx) property on child data source controls in order to display related data within the row or item. |
| [DataBound](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.basedataboundcontrol.databound.aspx) | Raised at the end of data-binding operations in a data-bound control. In a [GridView](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.gridview.aspx)control, data binding is complete for all rows and any child controls.  Use this event to format data-bound content or to initiate data binding in other controls that depend on values from the current control's content. (For more information, see[Catch-Up Events for Added Controls](https://msdn.microsoft.com/en-us/library/ms178472.aspx#catch_up_events_for_added_controls) earlier in this topic.) |

### [Nested Data-Bound Controls](javascript:void(0))

If a child control has been data bound, but its container control has not yet been data bound, the data in the child control and the data in its container control can be out of sync. This is true particularly if the data in the child control performs processing based on a data-bound value in the container control.

For example, suppose you have a [GridView](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.gridview.aspx) control that displays a company record in each row, and it displays a list of the company officers in a[ListBox](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.listbox.aspx) control. To fill the list of officers, you would bind the [ListBox](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.listbox.aspx) control to a data source control (such as [SqlDataSource](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.sqldatasource.aspx)) that retrieves the company officer data using the company ID in a query.

If the [ListBox](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.listbox.aspx) control's data-binding properties, such as [DataSourceID](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.basedatalist.datasourceid.aspx) and [DataMember](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.basedatalist.datamember.aspx), are set declaratively, the [ListBox](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.listbox.aspx) control will try to bind to its data source during the containing row's [DataBinding](https://msdn.microsoft.com/en-us/library/system.web.ui.control.databinding.aspx) event. However, the CompanyID field of the row does not contain a value until the[GridView](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.gridview.aspx) control's [RowDataBound](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.gridview.rowdatabound.aspx) event occurs. In this case, the child control (the [ListBox](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.listbox.aspx) control) is bound before the containing control (the[GridView](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.gridview.aspx) control) is bound, so their data-binding stages are out of sync.

To avoid this condition, put the data source control for the [ListBox](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.listbox.aspx) control in the same template item as the [ListBox](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.listbox.aspx) control itself, and do not set the data binding properties of the [ListBox](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.listbox.aspx) declaratively. Instead, set them programmatically at run time during the [RowDataBound](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.gridview.rowdatabound.aspx) event, so that the [ListBox](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.listbox.aspx) control does not bind to its data until the CompanyID information is available.

For more information, see [Binding to Data Using a Data Source Control](https://msdn.microsoft.com/en-us/library/ms228089.aspx).

## [**Login Control Events**](javascript:void(0))

The [Login](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.login.aspx) control can use settings in the Web.config file to manage membership authentication automatically. However, if your application requires you to customize how the control works, or if you want to understand how [Login](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.login.aspx) control events relate to the page life cycle, you can use the events listed in the following table.

|  |  |
| --- | --- |
| **Control Event** | **Typical Use** |
| [LoggingIn](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.login.loggingin.aspx) | Raised during a postback, after the page's [LoadComplete](https://msdn.microsoft.com/en-us/library/system.web.ui.page.loadcomplete.aspx) event has occurred. This event marks the beginning of the login process.  Use this event for tasks that must occur prior to beginning the authentication process. |
| [Authenticate](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.login.authenticate.aspx) | Raised after the [LoggingIn](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.login.loggingin.aspx) event.  Use this event to override or enhance the default authentication behavior of a [Login](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.login.aspx) control. |
| [LoggedIn](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.login.loggedin.aspx) | Raised after the user name and password have been authenticated.  Use this event to redirect to another page or to dynamically set the text in the control. This event does not occur if there is an error or if authentication fails. |
| [LoginError](https://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.login.loginerror.aspx) | Raised if authentication was not successful.  Use this event to set text in the control that explains the problem or to direct the user to a different page. |

# **A brief version history of .NET Framework**

*Posted By : Shailendra Chauhan, 17 Dec 2013*

*Updated On : 24 Nov 2015*

*Total Views : 60,776*

.NET is a software development platform developed by Microsoft. It runs on Microsoft Windows OS. .NET provides tools and libraries that allow developers to develop applications and services much easily, faster and secure by using a convenient way.

.NET Framework Version History

.NET Version

Introduced IDE

Features Detail

4.6

Visual Studio 2015

1. Introduced RyuJIT, a new JIT compiler for 64-bit systems
2. Introduced Open Source .Net Framework Packages
3. Support for Code page encodings
4. Improvements to event tracing

4.5.1

Visual Studio 2013

1. Includes performance and debugging improvements
2. Support for automatic binding redirection
3. Expanded support for Windows Store apps

4.5

Visual Studio 2012

1. Features Enhancements to CLR 4.0
2. Async Support
3. Support for building Windows Store apps
4. Features Enhancement to WPF, WCF, WF, and ASP.NET

4.0

Visual Studio 2010

1. Introduced CLR 4.0
2. Managed Extensibility Framework (MEF)
3. Dynamic Language Runtime (DLR)
4. Task Parallel Library

3.5

Visual Studio 2008

1. Built-In AJAX Support
2. LINQ
3. Dynamic Data
4. Multi-targeting Framework Support

3.0

Visual Studio 2005

1. Windows Presentation Foundation (WPF)
2. Windows Communications Foundation (WCF)
3. Windows Workflow Foundation (WF), and CardSpace

2.0

Visual Studio 2005

1. Introduced CLR 2.0
2. Generics and generic collections
3. Partial classes
4. Nullable types
5. Anonymous methods
6. Introduced many new controls and features to ASP.NET

1.1

Visual Studio .NET 2003

1. Features Enhancement to ASP.NET and ADO.NET
2. Built-in support for mobile ASP.NET controls
3. Security Enhancement
4. Built-in support for ODBC and databases
5. Internet Protocol version 6 (IPv6) support

1.0

Visual Studio .NET

1. Introduced CLR 1.0
2. Support for Object-oriented Web application development
3. Use of DLL class libraries

#### Note

CLR 3.0 is skipped by Microsoft.

**1. Introduction  
  
2. .Net Framework 2.0 Features**

* ADO.NET 2.0
* SQL Server data provider (SqlClient)
* XML
* .NET Remoting
* ASP.NET 2.0

**3. .Net Framework 3.0/3.5 Features**

* Windows Presentation Foundation (WPF)
* Windows Communication Foundation (WCF)
* Windows Workflow Foundation (WWF)
* Windows Card Space (WCS)
* Core New Features and Improvements:  
    
  + Auto Implemented
  + Implicit Typed local variable
  + Implicitly Typed Arrays
  + Anonymous Types
  + Extension Methods (3.5 new feature)
  + Object and Collection Initializers
  + Lambda Expressions

**4. .Net Framework 4.0 Features**

* Application Compatibility and Deployment
* Core New Features and Improvements  
    
  + BigInteger and Complex Numbers
  + Tuples
  + Covariance and Contravariance
  + Dynamic Language Runtime
* Managed Extensibility Framework
* Parallel Computing
* Networking
* Web
* Client
* Data
* Windows Communication Foundation
* Windows Workflow Foundation

**5. .Net Framework 4.5 Features**

* .NET for Windows Store Apps
* Portable Class Libraries
* Core New Features and Improvements
* Tools
* Parallel Computing
* Web   
  Windows Presentation Foundation (WPF)
* Windows Communication Foundation (WCF)
* Windows Workflow Foundation (WF)

**6. Conclusion**  
  
**Introduction**  
This article discusses the features introduced in Microsoft .Net Framework 2.0, 3.0, 3.5, 4.0 and the newly introduced 4.5 framework.   
  
**.Net Framework 2.0 Features**  
**ADO.NET**  
New features in ADO.NET include support for user-defined types (UDT), asynchronous database operations, XML data types, large value types, snapshot isolation, and new attributes that allow applications to support multiple active result sets (MARS) with SQL Server 2005.  
  
**ASP.NET**  
The Microsoft .NET Framework 2.0 includes significant enhancements to all areas of ASP.NET. For Web page development, new controls make it easier to add commonly used functionality to dynamic Web pages. New data controls make it possible to display and edit data on an ASP.NET Web page without writing code. An improved code-behind model makes developing ASP.NET pages easier and more robust. Caching features provide several new ways to cache pages, including the ability to build cache dependency on tables in a SQL Server database.  
  
ASP.NET accommodates a wide variety of browsers and devices. By default, controls render output that is compatible with XHTML 1.1 standards. You can use device filtering to specify different property values on the same control for different browsers.  
  
**.NET Remoting**  
  
.NET Framework Remoting now supports IPv6 addresses and the exchange of generic types. The classes in the System.Runtime.Remoting.Channels.Tcp namespace support authentication and encryption using the Security Support Provider Interface (SSPI). Classes in the new System.Runtime.Remoting.Channels.Ipc namespace allow applications on the same computer to communicate quickly without using the network. Finally, you can now configure the connection cache time-out and the number of method retries, which can improve the performance of network load-balanced remote clusters.  
  
**XML**  
The new System.Xml.XmlReaderSettings class allows specification of the type of verifications to be done when using a subclass of XmlReader to read XML data.  
  
It is now possible to partially validate a DOM tree loaded within an instance of XmlDocument.  
  
It is now possible to modify a DOM tree stored in an XmlDocument instance through the XPathNavigator cursor API.  
  
**.Net Framework 3.0/3.5 Features**  
**Windows Presentation Foundation (WPF)**  
  
Windows Presentation Foundation (WPF) is a next-generation presentation system for building Windows client applications. The core of WPF is a resolution-independent and vector-based rendering engine that is built to take advantage of modern graphics hardware.  
  
WPF extends the core with a comprehensive set of application-development features that include Extensible Application Markup Language (XAML), controls, data binding, layout, 2-D and 3-Dgraphics, animation, styles, templates, documents, media, text, and typography. WPF is included in the Microsoft .NET Framework, so you can build applications that incorporate other elements of the .NET Framework class library.  
  
To support some of the more powerful WPF capabilities and to simplify the programming experience, WPF includes additional programming constructs that enhance properties and events: dependency properties and routed events.  
  
**Windows Communication Foundation (WCF)**  
  
Windows Communication Foundation (WCF) is Microsoft's unified programming model for building service-oriented applications. WCF allows you to build many kinds of distributed applications including "traditional" Web Services so that your services support SOAP and will therefore be compatible with older .NET (and other) technologies. WCF is not just about pure SOAP over the wire - you can work with an Info set, and create a binary representation of your SOAP message that can then be sent along with your choice of protocol. This is for those who are particularly concerned about performance and have traditionally turned to .NET remoting.  
  
**Windows Workflow Foundation (WWF)**  
  
Windows Workflow Foundation, a core component of .NET Framework 3.0, provides a programming model, run-time engine, and tools for building workflow applications.  
  
A workflow is created and maintained by the workflow run-time engine. There can be several workflow engines within an application domain, and each workflow engine can support multiple workflows running concurrently. The run-time enables idle workflows to be unloaded from memory, persisted to a store, and reloaded whenever input is received.  
  
Workflows can be authored in code, XAML markup, or a combination of both, known as code-separation, which is similar to the ASP.NET mode.  
  
**Windows CardSpace (WCS)**  
  
Windows CardSpace (InfoCard) is a Digital Identity to online services. Digital Identity is how a user will be electronically represented. Such as for a debit/credit card, each card has a digital identity and password. If any user uses the site on internet then he enters their username and password, for identity, but this is not secure. WCS reduces these types of problems.   
  
WCS (originally called Info Card) helps people keep track of their digital identities as distinct information cards. If a Web site accepts WCS logins, users attempting to log in to that site will see a WCS selection. By choosing a card, users also choose a digital identity that will be used to access this site. CardSpace and the new supporting technologies will change how you authenticate into an application, whether it sits on the Web, your phone, or your desktop.  
  
**Core New Features and Improvements**  
  
Some core new features and improvements are implemented in .Net 3.0/3.5; they are:

1. Auto Implemented Property
2. Implicit Typed local variable
3. Implicitly Typed Arrays
4. Anonymous Types
5. Extension Methods (3.5 new feature)
6. Object and Collection Initializers
7. Lambda Expressions

**.Net Framework 4.0 Features**  
**Application Compatibility and Deployment**  
  
The .NET Framework 4 is highly compatible with applications that are built with earlier .NET Framework versions, except for some changes that were made to improve security, standards compliance, correctness, reliability, and performance.  
  
The .NET Framework 4 does not automatically use its version of the common language runtime to run applications that are built with earlier versions of the .NET Framework. To run older applications with .NET Framework 4, you must compile your application with the target .NET Framework version specified in the properties for your project in Visual Studio, or you can specify the supported runtime with the <supportedRuntime> Element in an application configuration file.  
  
**Core New Features and Improvements**  
  
Some new features are introduced in .Net framework 4.0.

The following sections describe new features and improvements provided by the common language runtime and the base class libraries.

1. BigInteger and Complex Numbers
2. Tuples
3. Covariance and Contravariance
4. Dynamic Language Runtime

**Managed Extensibility Framework**  
  
The Managed Extensibility Framework (MEF) is a new library in the .NET Framework 4 that helps you build extensible and composable applications. MEF enables you to specify points where an application can be extended, to expose services to offer to other extensible applications and to create parts for consumption by extensible applications.  
  
It also enables easy discoverability of available parts based on metadata, without the need to load the assemblies for the parts.  
  
**Parallel Computing**  
  
The .NET Framework 4 introduces a new programming model for writing multithreaded and asynchronous code that greatly simplifies the work of application and library developers. The new model enables developers to write efficient, fine-grained, and scalable parallel code in a natural idiom without having to work directly with threads or the thread pool. The new System.Threading.Tasks namespace and other related types support this new model.  
  
**Web**  
  
ASP.NET version 4 introduces new features in the following areas:

* Core services, including a new API that lets you extend caching, support for compression for session-state data, and a new application preload manager (autostart feature).
* Web Forms, including more integrated support for ASP.NET routing, enhanced support for Web standards, updated browser support, new features for data controls, and new features for view state management.
* Web Forms controls, including a new Chart control.
* MVC, including new helper methods for views, support for partitioned MVC applications, and asynchronous controllers.
* Dynamic Data, including support for existing Web applications, support for many-to-many relationships and inheritance, new field templates and attributes, and enhanced data filtering.
* Microsoft Ajax, including additional support for client-based Ajax applications in the Microsoft Ajax Library.
* Visual Web Developer, including improved IntelliSense for JScript, new auto-complete snippets for HTML and ASP.NET markup, and enhanced CSS compatibility.
* Deployment, including new tools for automating typical deployment tasks.
* Multi-targeting, including better filtering for features that are not available in the target version of the .NET Framework

**Windows Presentation Foundation (WPF) Features in 4.0**  
Windows Presentation Foundation (WPF) version 4 contains changes and improvements in the following areas:

* New controls, including Calendar, Data Grid, and Date Picker.
* VisualStateManager supports changing states of controls.
* Touch and Manipulation enables you to create applications that receive input from multiple touches simultaneously on Windows 7.
* Graphics and animation supports layout rounding, Pixel Shader version 3.0, cached composition, and easing functions.
* Text has improved text rendering and supports customizing the caret color and selection color in text boxes.
* Binding is supported on the Command property of an InputBinding, dynamic objects, and the Text property.
* XAML browser applications (XBAPs) support communication with the Web page and support full-trust deployment.
* New types in the System.Windows.Shell namespace enable you to communicate with the Windows 7 taskbar and pass data to the Windows shell.
* The WPF and Silverlight Designer in Visual Studio 2010 has various designer improvements to help create WPF or Silverlight applications.

**Windows Communication Foundation Features in 4.0**  
  
Windows Communication Foundation (WCF) provides the following improvements:

* Configuration-based activation: Removes the requirement for having an .svc file.
* System.Web.Routing integration: Gives you more control over your service's URL by allowing the use of extensionless URLs.
* Multiple IIS site bindings support: Allows you to have multiple base addresses with the same protocol on the same Web site.
* Routing Service: Allows you to route messages based on content.
* Support for WS-Discovery: Allows you to create and search for discoverable services.
* Standard endpoints: Predefined endpoints that allow you to specify only certain properties.
* Workflow services: Integrates WCF and WF by providing activities to send and receive messages, the ability to correlate messages based on content, and a workflow service host.

**Windows Workflow Foundation Features in 4.0**  
  
Windows Workflow Foundation (WF) provides improvements in the following areas:

* Improved workflow activity model: The Activity class provides the base abstraction of workflow behavior.
* Rich composite activity options: Workflows benefit from new flow-control activities that model traditional flow-control structures, such as Flowchart, TryCatch, and Switch<T>.
* Expanded built-in activity library: New features of the activity library include new flow-control activities, activities for manipulating member data, and activities for controlling transactions.

**.Net Framework 4.5 Features**  
  
**.NET for Windows Store Apps**  
  
Windows Store apps are designed for specific form factors and leverage the power of the Windows operating system. A subset of the .NET Framework 4.5 is available for building Windows Store apps for Windows by using C# or Visual Basic.  
  
**Portable Class Libraries**  
  
The Portable Class Library project in Visual Studio 2012 enables you to write and build managed assemblies that work on multiple .NET Framework platforms. Using a Portable Class Library project, you choose the platforms (such as Windows Phone and .NET for Windows Store apps) to target.  
  
**ASP.NET 4.5**   
  
ASP.NET 4.5 includes the following new features:

* Support for new HTML5 form types.
* Support for model binders in Web Forms. These let you bind data controls directly to data-access methods, and automatically convert user input to and from .NET Framework data types.
* Support for unobtrusive JavaScript in client-side validation scripts.
* Improved handling of client script through bundling and minification for improved page performance.
* Integrated encoding routines from the AntiXSS library (previously an external library) to protect from cross-site scripting attacks.
* Support for WebSockets protocol.

**Windows Presentation Foundation (WPF) Features in 4.5**In the .NET Framework 4.5, Windows Presentation Foundation (WPF) contains changes and improvements in the following areas:

* The new Ribbon control, which enables you to implement a ribbon user interface that hosts a Quick Access Toolbar, Application Menu, and tabs.
* The new INotifyDataErrorInfo interface, which supports synchronous and asynchronous data validation.
* New features for the VirtualizingPanel and Dispatcher classes.
* Improved performance when displaying large sets of grouped data, and by accessing collections on non-UI threads.
* Data binding to static properties, data binding to custom types that implement the ICustomTypeProvider interface, and retrieval of data binding information from a binding expression.
* Repositioning of data as the values change (live shaping).
* Ability to check whether the data context for an item container is disconnected.
* Ability to set the amount of time that should elapse between property changes and data source updates.
* Improved support for implementing weak event patterns. Also, events can now accept markup extensions.

**Windows Communication Foundation (WCF) Features in 4.5**  
  
In the .NET Framework 4.5, the following features have been added to make it simpler to write and maintain Windows Communication Foundation (WCF) applications:

* Simplification of generated configuration files.
* Support for contract-first development.
* Ability to configure ASP.NET compatibility mode more easily.
* Changes in default transport property values to reduce the likelihood that you will have to set them.
* Updates to the XmlDictionaryReaderQuotas class to reduce the likelihood that you will have to manually configure quotas for XML dictionary readers.
* Validation of WCF configuration files by Visual Studio as part of the build process, so you can detect configuration errors before you run your application.
* New asynchronous streaming support.
* New HTTPS protocol mapping to make it easier to expose an endpoint over HTTPS with Internet Information Services (IIS).
* Ability to generate metadata in a single WSDL document by appending ?singleWSDL to the service URL.
* Websockets support to enable true bidirectional communication over ports 80 and 443 with performance characteristics similar to the TCP transport.
* Support for configuring services in code.
* XML Editor tooltips.

**Windows Workflow Foundation (WF) Features in 4.5**  
  
Several new features have been added to Windows Workflow Foundation (WF) in the .NET Framework 4.5. These new features include:

* State machine workflows, which were first introduced as part of the .NET Framework 4.0.1 (.NET Framework 4 Platform Update 1). This update included several new classes and activities that enabled developers to create state machine workflows. These classes and activities were updated for the .NET Framework 4.5 to include:
* The ability to set breakpoints on states.
* The ability to copy and paste transitions in the workflow designer.
* Designer support for shared trigger transition creation.