.NET

short notes & interview questions

Ketan Jetty

The beauty of the brain is to forget

Introduction

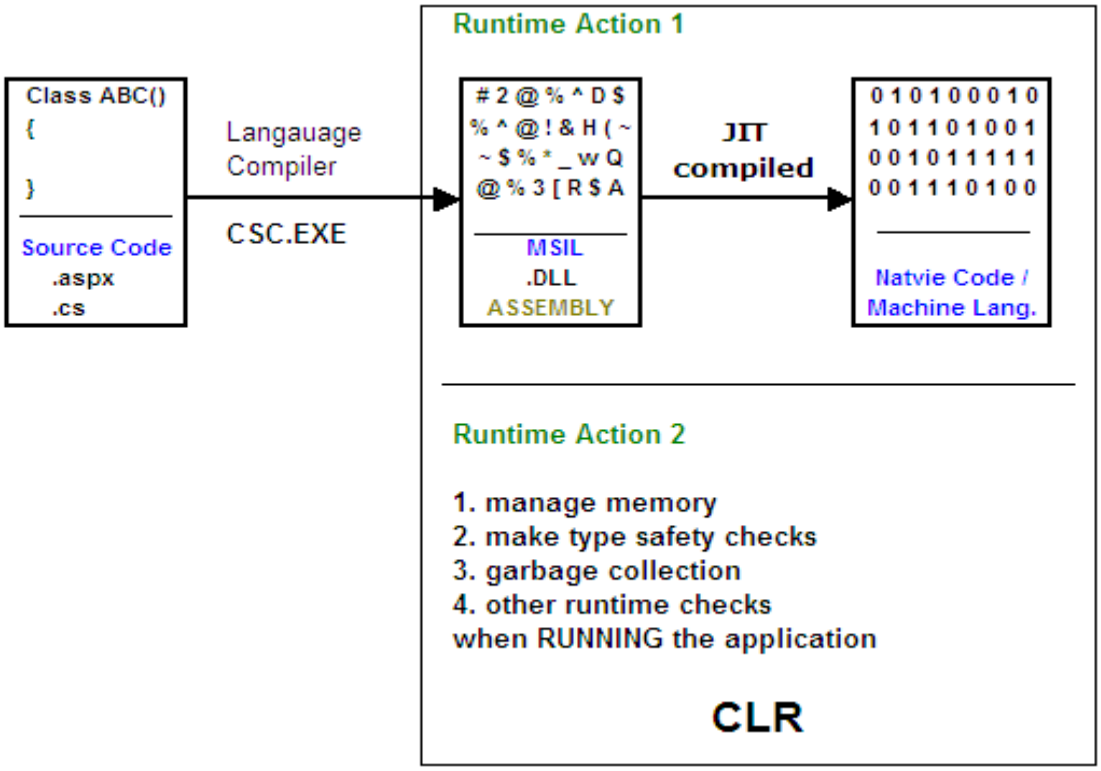
DOTNET framework is a managed type-safe environment for application development and execution. It manages all aspects of your program's execution like allocation of memory, permissions, etc. It has 2 main components

- **CLR** *(Common Language Runtime)* is also an [**Execution Engine**]
  Provides managed runtime environment to manage code execution, it provides core services, such as code compilaton, memory allocation, thread management, and garbage collection.
- **FCL** *(Framework Class Library)*
  Provides a collection of useful and reusable types (classes) organized in hierarchical namespaces. It provides a library of standard types.

When a .NET application is compiled, LANGUAGE COMPILERS compile the **SOURCE CODE** (written in higher language like C#, VB.net) to **MSIL** (MS Intermediate Language) and store it in a file called an **ASSEMBLY** [IL code + metadata] [.dll or .exe] also called PORTABLE EXECUTABLE. CLR now JIT-compiles the IL/PE to MACHINE SPECIFIC NATIVE CODE [**BINARY CODE**] at runtime.

- MSIL (Microsoft intermediate language) is a low-level language that the CLR can read and understand.
- All DLL's and EXE's exist in MSIL. (A .NET executable is stored as an IL file)
- .NET Base Class Library is a collection of object-oriented types and interfaces that provides many of the services and objects you need when writing your applications. The class library is organized into namespaces. A namespace is a logical grouping of types that perform related functions. Namespaces are logical grouping or related classes.

BROWSER ←→ IIS (inetinfo.exe) ←→ aspnet_isapi.dll ←→ aspnet_wp.exe



ASP.NET runtime host (Worker Process **aspnet_wp.exe**) uses FCL, loads CLR into the process, compiles the .aspx page into an assembly, creates an application domain (AppDomain) and then loads & executes the requested page within the AppDomain. AppDomain contains one or more THREADS.

CLR – is an execution engine that handles runtime services such as language integration, code compilation, security, memory allocation, thread management, and garbage collection. CLR also loads, executes the IL code for running an application and while running, the CLR provides memory management, type-safety checks and other run-time tasks for the application. Loads, compiles, and executes IL Code and enforce security, type safety and thread support.

FCL – provides reusable types that are designed to integrate with CLR.

.NET pages extensions

.aspx	asp.net page
.ascx	user control
.asmx	web service
.ashx	httphandlers
.axd	trace.axd
.asax	global.asax
.config	web.config

Memory

Application data memory is divided into 2 areas STACK and HEAP.

- STACK is used for running a program. Stack is analogus to a stack of dinner plates (LIFO)
  - All data associated with a value type is allocated on the stack.
  - Access to the stack is designed to be light and fast
- HEAP is used for the creation of reusable types (objects).
  - All reference types are allocated memory on the heap (objects) as well on the stack (references/pointers)

Data Types

- Value Type
  - Stores actual data
  - Value type data is allocated on the stack.
  - When a variable of a value type goes out of scope, it is destroyed and its memory is reclaimed.
  - Value types are all primitive types like int, float, decimal, bool, char, **. . . .**
  - User-defined types such as struct and enum are also value types.
  - A value type variable contains all the data associated with that type.
- Reference Type also referred as objects
  - Stores reference to actual data
  - Class, interface, delegate and arrays are reference types
  - Built in reference types are object and string
  - A reference type exists on both stack and heap.
  - Type *(address/reference/pointer)* is allocated on the stack and the object is created on the heap.
  - A reference type variable contains a pointer to an instance of an object of that type.
- Pointer Types used in unsafe mode

Using

- Using statement is used to referer members of a namescape without using the fully qualified name.
- `Using System.Web.Util;`

Class & Objects

- Classes are templates for objects. They describe the amount of data that an object will contain, but they do not represent any particular instance of an object.
- Classes describe the properties and behaviors of the objects they represent through members.
- Class members are fields, properties, methods and events that belong to a particular class.
  - *Fields* and *Properties* represent the data of an object
  - *Method* represents actions your class can take
    - Main() - start point of a class
    - Constructors - used for object initialization
    - Destructors - called just prior to objects destruction
  - *Events* are notifications a class receives from or sends to other objects when an activity happens in the application.

```
public class CLASS1
{     public CLASS1()            // constructor
      {
      }

      public static void Main(String[] args)
      {
      }

      ~CLASS1()                  // destructor
      {
      }

      private int I;             // field

      public int propI           // properties
      {     get { return I;  }
            set { I = value; }
      }
}
```

- Objects are created from a template called class. Classes can be thought of as blueprints for object.
- When a class is instantiated, an in-memory instance of that class is created. This instance is called an object.
- The ability of programmatic objects to represent real-world objects is called *abstraction*.
- You can create an object using the new keyword

```
CLASS1 obj = new CLASS1();
```

Property

- Property is essentially a specialized method that looks like a field. Properties are set and retrieved in the same manner as fields.
- Keyword "*value*" is a special keyword used in the setter of a property. It always represents the value to which the property is being set.

Constructors

- Constructors are the first method that is run when an instance of type is created.
- It is a method with the same name as the class.
- You use a constructor to initialize class and structure data.
- Constructors never return a value and can be overridden and overloaded to provide custom intialization.

Destructors

- Destructor is the last method run by a class. This should contain code to clean up when class is destroyed.
- This is a method with the same class name but ~ (tilde) in front.
- You cannot control when a destructor is called as object clean up is controlled by CLR.

Struct

```
public struct STRUCT1
{
}
```

Class vs Struct

Classes are references types whereas *structs* are value types. Structs are best used for smaller, lightweight objects that contain relatively little instance data or for objects that do not persist for long. Both classes and structs can contain nested types.

Nested Types

Types within types are called nested types. A nested class/type usually represents an object that the parent might need to create and manipulate, but which an external object would never need to create independently. Ex. Wheel class. A Wheel class might need to create and maintain a collection of Spoke objects internally, but outside users would probably never need to create a Spoke object independent of a Wheel.

```
public class WHEEL
{    private class SPOKE
     {
     }
}

note: access level of the nested class cannot be more (explicit) than that of the
parent class. If the parent is internal, and nested is public, the nested class
practically has an access level of internal.
```

Parameters and Arguments

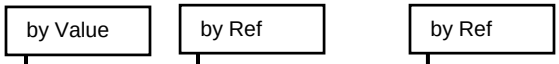
Parameters are arguments that are passed to the methods. Parmeters are enclosed in parentheses after the method name in the method declaration.

Parameters can be passed in 2 ways

- By value (parameters are passed by value by default)
- By reference (to pass parameters by reference, use the keyword *ref*)

Primitive types are passed by value, whereas Objects are passed by Reference.

You can also have output parameters. Output parameters are always passed by reference and need not be initiazed before use.




```
public class CLASS1
{    public void METHOD1(int PARAM1, ref int PARAM2, out string PARAM3)
     {
     }
}

abc(arg1, arg2);          function abc(param1, param2);
```

Access Modifiers
Access modifiers define how types are instantiated and how members are accessed.

- public - globally accessible
- private - access is limited only to the containing class
- protected - access is limited to containing class + *classes derived from the containing class*
- protected internal- access is limited to current project + *classes derived from the containing class*
- internal - access is limited to current project.

The default access modifer for a CLASS with undefined modifier is public. (for CLASS and STRUCT)
The default access modifer for a METHOD with undefined modifier is private. (for CLASS and STRUCT)
The default access modifer for a VARIABLE with undefined modifier is private. (for CLASS)
The default access modifer for a VARIABLE with undefined modifier is public. (for STRUCT)

```
public class CLASS1()
{
      private int I;

      private void Method1()
      {
      }
}
```

```
public struct STRUCT1()
{
      public int I;

      private void Method1()
      {
      }
}
```

Static

- Static members belong to class/type but not to an object.
- Only one instance of the static member can exist, no matter how many instances of a particular type have been created.
- Methods can be static too, but as thay belong to a type, thay cannot access instance data from any objects.
- Static methods cannot refer to any instance data.

Form

- Forms are the primary element of a Microsoft windows application.
- Every Form is a class.

Controls and Components

- Controls have visual representation, whereas, components do not.
- Components are added to the component tray.

Container Controls
Container control acts as a host for other controls.

- Panel - scrollable container
- GroupBox - provides a caption for labeling the group of controls within it
- TabControl - used to group controls on a set of tabs

Docking / Anchoring
Anchor allows you to define constant between a control and edges of the form.
Docking allows you to attach a control to an edge of the form or to completely fill the form. A docked control will resize itself when the form is resized. Ex. Menu bar at the top of the form.

Extender Provider components

- ToolTipProvider
- HelpProvider
- ErrorProvider

Validation

Form-level validation - verifies data after the user has filled in all fields
Field-level validation - verifies data in each field on every key press
Validating Events - easiest way to validate, validation event occurs before a control loses focus

Convertions

- Implicit conversion

```
int I = 1234;
long L = I;
```

- Explicit casting
  Also known as casting, used when assigning a larger value to a variable than it can hold.

```
long L = 7123123123;
int I = (int) L;
```

Boxing / Unboxing

- Boxing is the implicit conversion of value type to reference type
- Following is an example to manually box a integer

```
int I = 1234;
object O;
O = I;
```

- Unboxing is the conversion of boxed variable back to a value type. To unbox a variable, you must perform an explicit cast on the object to convert it to the appropriate type.
- Only boxed variables can be Unboxed.

Parse

Parse is used to create a numeric value from a string. Parse method is extremely useful when developing user interfaces.

```
string S;
int I = int.parse(S);
```

Constants, Readonly and Enums

- Constants allow you to refer to frequently used values by friendly names.
- Const key word is used to state a variable as constant.
- Constants can be of any value type.

```
public const double PI = 3.14;
```

- Readonly is the preferred way to defined constants.

```
public readonly double PI = 3.14;
```

- Enumerations allow you to work with sets of related constants and to associate easy-to-remember names with those sets.
- Enums are user-defined types
- The default data type for enums is int
- It is not necessary to supply values for the members of you enums. The default values start from 0.
- Enums members can convert to the value that they represent and can be used as constants.
- Enums must be of a numeric integral type.

```
public enum DaysOfWeek
{    Monday    = 1,
     Tuesday   = 2,
     Wednesday = 3,
     Thursday  = 4,
     Friday    = 5,
     Saturday  = 6,
     Sunday    = 7
}
```

```
public enum DaysOfWeek2
{    Monday,    // equals 0
     Tuesday,   // equals 1
     Wednesday, // equals 2
     Thursday,  // equals 3
     Friday,    // equals 4
     Saturday,  // equals 5
     Sunday,    // equals 6
}
```

```
MessageBox.Show(((int) DaysOfWeek.Friday * 2).toString()); // output is 10
```

Arrays, Collections, ArrayList

- Arrays are a way to manage groups of similarly typed values or objects.
- You refer an array using an index.
- All arrays are 0 based.
- There are 2 types of arrays
  - Rectangular arrays
  - Jagged arrays
    - `int[] ia = new int[10];`
    - `int[] ia = new int {3,7,1,5};`
    - `int[] ia = new int[4] {1,2,3,4};`

- ArrayLists are collections and dynamic arrays.
- Allow you to dynamically add and remove items from a simple list.
- As items are removed from a collection, the index numbers are reassigned to occupy any available spaces.

- Collections allow you to manage groups of objects, which can be of the same or different types.
- The following are the collection classes
  - BitArray - array of bits (0,1)
  - CollectionBase - serves as a base for implementing you own collection class
  - HashTable - represents a collection of key-value pairs based on the hash code of the key. They are very fast.
  - Queue - manages a group of objects FIFO
  - SortedList - organizes a group of objects, allows access using key or index
  - Stack - manages a group of objects FILO

```
foreach (object O in myList)
{    if (O.GetType() == typeof(string)
     {    MessageBox.Show(O.ToString());
     }
}
```

readOnly, writeOnly properties

```
private readonly int I;
public int propI          // properties
{    get { return I;  }
}

private writeonly int I;
public int propI          // properties
{    set { I = value; }
}
```

Events / Event Handler / Event Arguments

- Event represents a message that is sent to other parts of the application
- An event can be handled by more than one method, and a single method can handle more than one event
- Event Handlers allow your application to respond to user input.
- 2 arguments are always passed to an event handler by the object that raises the event. `(Object sender, EventArgs e)`

Encapsulation

- Encapsulation is the concept that implementation of an object is independent of its interface.
- In other words, as an application with an object through its interface, as long as the interface remains constant, the application can contain to interact with the component, even if implementation of the interface was completely rewritten between versions.

Polymorphism

- Polymorphism is the ability of different classes/objects to provide different implementations of the same public interfaces.
- There are 2 types of polymorphism
  - Interface polymorphism
  - Inheritance polymorphism

Single polymorphism is achieved by method overriding.

Multiple polymorphism is when an abstract class uses another abstract class. Using multiple polymorphism leads to a more scalable and extensible design.

Inheritence

- Inheritence allows you to declare a new class that retains all the members and functionality of a previously defined class.
- Derived / Base class
- Any derived class may be implicitly converted to its base class.
- When cast to its base class, any members implemented in the derived class will be inaccessible. [Only base class members will be available]
- Classes can only inherit a single base class, but can implement multiple interfaces

sealed class

- if you want to create a class that cannot serve as a base class for other classes, create this class as a sealed class. If you do not want the class to be inherited.

```
public sealed class BCLASS
{
}
```

overriding

- Only methods can be overridden [variables (fields), events cannot be overridden]
- Keyword *override* should be used
- When a method is overridden, the new member is called in place of the base class method
- The type of the object, not the variable, determines which method is called
- To override a base class method, the method should be marked as *virtual*

```
public class Car
{     public virtual void GoForward(int speed)
      {
      }
}

public class SportsCar : Car
{     public override void GoForward(int speed)
      {
      }
}
```

hiding

- The keyword *new* is used to hide a base class member
- To access the base class member, use base.GoForward()
- Overloading vs Hiding
  - Any method that has the same name as a base class method but a different signature is treated as an overload of that method (this will not hide the base class method).
  - The new keyword is used to hide a base class member and provide a new implementation.

```
public class Car
{   public int GoForward(int speed)
    {
    }
}

public class SportsCar : Car
{   public new string GoForward(int speed)
    {
    }
}
```

example:

```
public class Animal
{   public Animal()
          {    Console.WriteLine("Animal - Constructor"); }

          public void Greet()
          {    Console.WriteLine("Animal - Greet");       }

          public void Talk()
          {    Console.WriteLine("Animal - Talk");        }

          public virtual void Sing()
          {    Console.WriteLine("Animal - Sing");        }
}

public class Dog : Animal
{   public Dog()
          {    Console.WriteLine("Dog - Constructor");    }

          public new void Talk()
          {    Console.WriteLine("Dog  - Talk");          }

          public override void Sing()
          {    Console.WriteLine("Dog - Sing");           }
}


Animal a = new Dog();     //output:  Animal – Constructor
                          //output:  Dog – Constructor
a.Greet();                //output:  Animal - Greet
a.Talk();                 //output:  Animal - Talk
a.Sing();                 //output:  Dog - Sing

Dog d = new Dog();        //output:  Animal – Constructor
                          //output:  Dog – Constructor
d.Greet();                //output:  Animal - Greet
d.Talk();                 //output:  Dog - Talk
d.Sing();                 //output:  Dog - Sing
```

Abstract class

- Abstract classes are classes that contain one or more abstract methods. An abstract method is a method that is declared, but contains no implementation.
- To represent base class, and not to create objects of these class types, create the class as abstract.
- Abstract classes cannot be instantiated. Abstract classes are classes that must be inherited.
- An abstract class can contain either abstract methods or non-abstract (concrete) methods.
- Abstract members do not have any implementation in the abstract class, but the same has to be provided in its derived class.
- Abstract classes are frequently either fully or partially implemented, or not at all implemented.
- Defining an abstract class with abstract members has the same effect to defining an interface.
- When a class inherits from an abstract class, it must provide an implementation for every abstract member.
- An abstract member declaration is similar to the interface member declaration. Only the member type, access level, required parameters, and return type are specified. No details regarding implementation of the members are defined in the method declaration.

```
// abstract class cannot be sealed
public ~~sealed~~ abstract class Car
{     public abstract void GoForward(int I);
      public abstract int CheckSpeed();

      public abstract string Color
      {     get;
            set;
      }

      ~~private~~ ~~static~~ abstract ~~virtual~~ int m();
      // abstract member cannot be private / static / virtual
         (as it is implicitly virtual)
}
```

Overloading

- Overloading allows you to create multiple members (methods) with the same name. Each member that shares a name must have a different signature.
- Signature is a combination of access modifier, return value, and parameter list.

overloaded operator

- Use the operator keyword to signify that it is an operator, followed by the operator that you want to overload.
- Overloaded operators must be public and static

```
public static type operator op (arg1 [,arg2])
{
}
```

Interface

An interface is a construction similar to an abstract class but with no implementation code. It is only composed of method declarations, indexers, events and constants. An interface is a contract defining that any class that implements an interface must implement all the method definitions given in the interface.

- Interface is a contract for behaviour.
- Interfaces, like classes, define a set of properties, methods, and events. But unlike classes, interfaces do not provide implementation.
- A class that implements an interface must implement every aspect of that interface exactly as it is defined. Implementation of the interface is left entirely to the implementing class or structure.
- Any object that implements an interface can interact with any object that requires that interface.
- Interfaces are defined with using the keyword interface.

- Ex., Ishape defined CalculateArea, which will have a different implementation for CIRCLE class and SQUARE class.
- Members must be defined with method signature, but without the access modifiers.
- The interface access modifier decides the member's access modifiers i.e., if you have a private access level for interface then all the members by default will have private access level.
- You can also add properties to your interface, but a definition for properties should define getters or setters or both and as well specify the return type.
- Colon : is used to designate that a class/struct implements a specific interface.
- A Class can implement multiple interfaces.
- There are 2 ways to implement an interface
  - By implementing the interface with the same name, signature, and access level of the member. This way this member will be available to both the class and the interface.
  - By explicitly implementing the interface member using the fully qualified member name. This way the member will be available only to the interface.

```
// default is public, if access modifier is not specified
interface Iinterface1
{     //implicity public final static constant
      public static final const int i = 0;

      //all methods are implicitly abstract / public
      public abstract int AddNumbers(int Num1, int Num2);
      public abstract int MultiplyNumbers(int Num1, int Num2);

      // readonly property
      int I { get; }

}

private interface Iinterface2
{     private void add (int a, intb);
      private string greetings (string s);
}

public class CLASS1 : Iinterface1, Iinterface2
{
}
```

	Interface	**Abstract**
Inheritence	A class may implement several interfaces.	A class may extend only one abstract class
Implementation	An interface cannot provide any code at all.	An abstract class can provide complete code, default code, and/or just stubs that have to be overridden.
Constants	Static final constants only, can use them without qualification in classes that implement the interface. (Presumed public static final)	Both instance and static constants are possible.
Third party convenience	An interface implementation may be added to any existing third party class.	A third party class must be rewritten to extend only from the abstract class.
is-a vs -able or can-do	Interfaces are often used to describe the peripheral abilities of a class, not its central identity.	Abstract class defines the core identity of its descendants.

Additional functionality	If you add a new method to an interface, you must track down all implementations of that interface in the universe and provide them with a concrete implementation of that method.	If you add a new method to an abstract class, you have the option of providing a default implementation of it. Then all existing code will continue to work without change.

```
byte      sbyte     1 [8]     -128 to 127            sbyte sbi  = 12;
          byte      1         0 to 255               byte  bi   = 12;

short     short     2 [16]    -32k to 32k            short  si  = 12345;
          ushort    2         0 to 65k               ushort usi = 62345;

int       int       4 [32]    -2b to 2b              int  ni    = 1234567890;
          unit      4         0 to 4b                uint uni   = 323467890U;

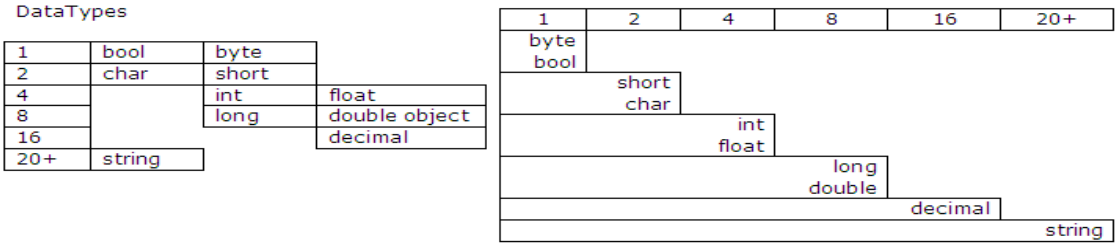
long      long      8 [64]    -10^20 to 10^20        long  li   = 123456789012L;
          ulong     8         0 to 2*10^20           ulong uli  = 123456789012UL;

float     float     4 [32]    1.5*10^-45 to 3.4*10^38  float fi   = 1.2F;
                              [accuracy 6 to 7]

double    double    8 [64]    5.0*10^-324 to 1.7*10^308 double di  = 1.2;
                              [accuracy 15 to 16]

decimal   decimal   16 [128]  1.0*10^-28 to 7.9*10^28  decimal mi   = 1.2M;
                              [accuracy 28 to 29 places]
```

float and double will have rounding issues 25.0 could be 25.0000001
Decimal operations are 50 times slower than int operations
Double operations are 3 times slower than int operations.



Constants

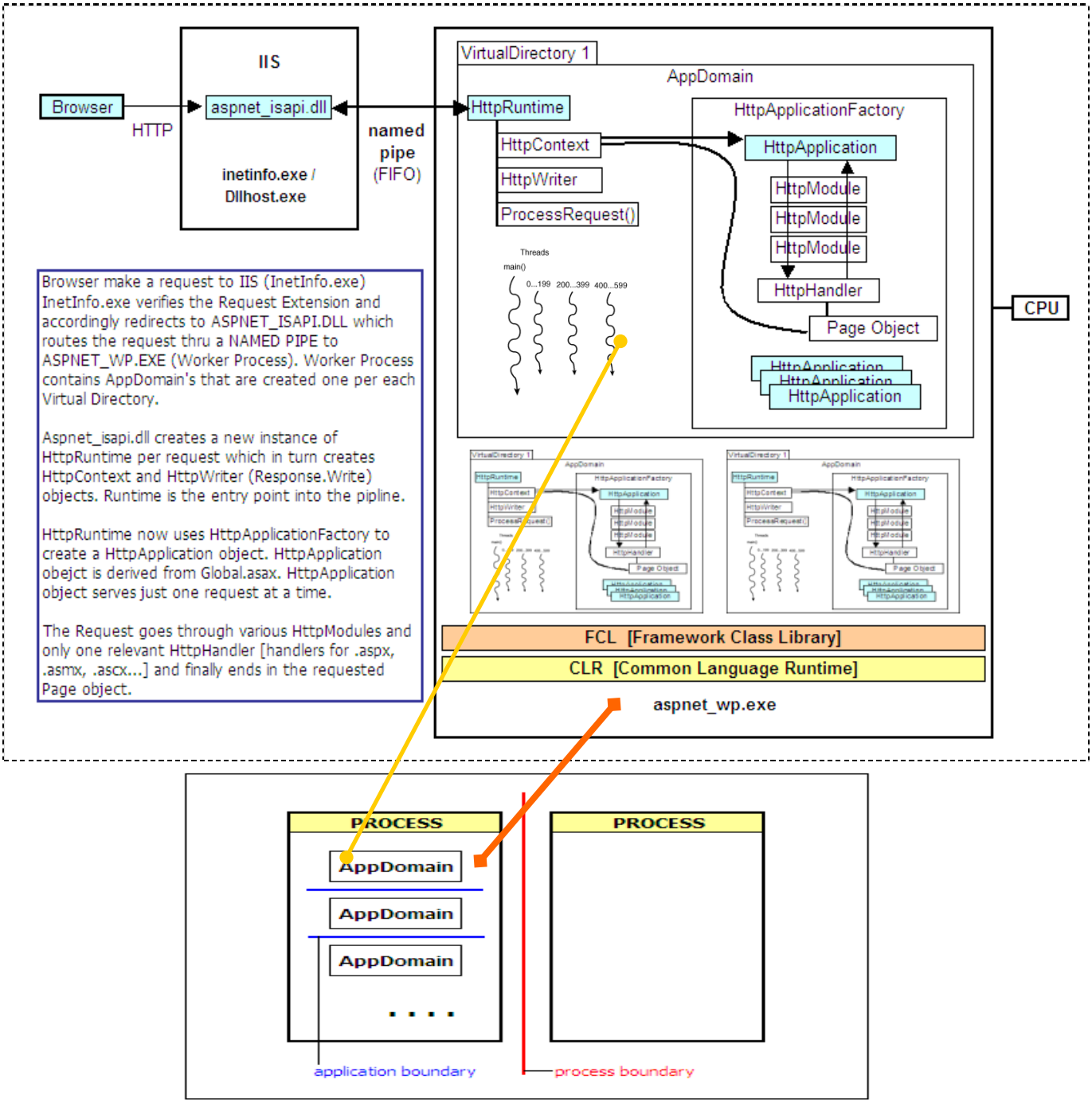
```
public const float PI_VALUE = 3.14;
public readonly float PI_VALUE = 3.14; [preferred way]
```

Arguments/Parameters

```
abc (arg1, arg2);          function  abc (int param1, int param2);
```

arguments by default are passed by value
to pass arguments by reference use ref and out keywords

```
calculate (int i, ref int j, out int k);
```



- A single copy of worker process runs all the time and hosts all the active web applications.
- Web Garden – multiple CPU
  Web Farm – many computers

AppDomain

- An AppDomain is created within the worker process whenever a client addresses a virtual directory for the first time.
- AppDomain is the smallest execution unit for a .NET application.
- Each AppDomain can contain one or more threads.

- An application pool is a blanket term that identifies a worker process and a virtual directory.
- HttpApplicationFactory maintains a pool of HttpApplication objects.
- HttpApplication is a global.asax derived object.
- HttpApplication object is used to process a single request at a time.

- Each asp.net is allowed a maximum number of recompiles (15 as default) before the whole application is restarted.
- When a page is changed, the assembly is recompiled and asp.net tries to delete the old assembly if asp.net finds that the old assembly is still locked then the old assembly is renamed with a .DELETE postfix. This .DELETE old assembly is scavenged during the next SWEEP MODE or whenever the application is restarted or an application file (global.asax, web.config) changes.
- Server form – default value for method=POST, HTML form – default value for method=GET

- For service side events two hidden form variables are created _EVENTTARGET (contains the name of the control that caused the postback) and _EVENTARGUMENT (contains any argument that might be useful to carry out the call).
- A master page is a distinct file referenced at the application level as well as page level that contain the static layout of the page.

A directive controls how and ASP.NET page is compiled.

```
<@ Page . . . >
```

Page.IsPostBack – checks if the page is requested the first time or not.

SmartNavigation

- Persists element focus and scroll position between postbacks.
- Eliminates page flash by using double buffering technique.
- Prevents each postback from being saved in the browser history.
- In web.config <pages SmartNavigation=true />

Stages of ASP.NET page processing

OnInit()	Initialize - Initialize settings for the incoming Web request. • Creates an instance of ASP.NET page • Starts instantiation and initializing of Page and Controls • Appropriate place for ***attaching a delegate*** for event handling
LoadViewState ()	Load view state - At the end of this phase, the ViewState property of a control is automatically populated. A control can override the default implementation of the LoadViewState method to customize state restoration.
LoadPostData ()	LoadPostdata - Process postback data - Process incoming form data and update properties accordingly.
OnLoad()	Load - Perform actions common to all requests, such as setting up a database query. At this point, server controls in the tree are created and initialized, the state is restored, and form controls reflect client-side data. • At this stage Page Initialization is complete and Page is loaded into memory. • User code initialization / controls are loaded in the Page object. • ViewState is available at this point. • Always use this method for loading the controls dynamically. • Best time ***to initialize controls, configure properties***
RaisePostDataChangedEvent()	Send postback - change notifications Raise change events in response to state changes between the current and previous postbacks.
RaisePostBackEvent()	Handle postback events - Handle the client-side event that caused the postback and raise appropriate events on the server.
OnPreRender()	Prerender - Perform any updates before the output is rendered. Any changes made to the state of the control in the prerender phase can be saved, while changes made in the rendering phase are lost. • Pre-render / this is the last chance to modify the Pages output • The brief moment before the page is displayed to the user
SaveViewState()	Save state - The ViewState property of a control is automatically persisted to a string object after this stage. This string object is sent to the client and back as a hidden variable. For improving efficiency, a control can override the SaveViewState method to modify the ViewState property.
Render()	Render - Generate output to be rendered to the client.
Dispose()	Dispose - Perform any final cleanup before the control is torn down. References to expensive resources such as database connections must be released in this phase.
OnUnLoad()	Unload - Perform any final cleanup before the control is torn down. Control authors generally perform cleanup in Dispose and do not handle this event. • Page clean up. Page has been rendered and is ready for discarding. • When the page starts unloading

- Asp.net applications are compiled pieces of code.
- An asp.net page isn't compiled into native machine code until it is actually requested by a browser.
- Reentrant Form is am html <form> that posts to the same page that contains it.
- Call context are hidden fields that carry session state.
- <FORM> tag is the only element authorized to transmit client side data to the server.
- Web Server <form> default method="POST"
- Html <form> default method="GET"
- Objects like REQUEST, RESPONSE, SERVER form the HTTP CONTEXT for the call.
- Directives let you import namespaces, load assemblies, register new controls, etc.,
- <@ Page> sets up the environment in which the page will run. Page directive can be used only with an .aspx page.<@ Page|Control="" CodeBehind="test.cs" Src="test.cs"> Page and Control attributes are mutually exclusive. AspCompat="true|false" for STA [Single-Threaded Apartment] execution.
- Inherits attribute give the link info at runtime,
- CodeBehind attributes give the link info during design time.
- Inherits + Src attribute combination is used for on demand / dynamic compilation
- runat attribute promotes the server-side tag to the rank of component instance.
- Custom controls are created as assemblies [.dll]. User controls are source files [.ascx]
- When asp.net page is parsed all directives attributes are extracted and stored in a Dictionary.
- <@ Assembly > links an assembly to the current page. <@ Assembly Name|Src="">
- By default 9 assemblies are linked and also all the assemblies that lie in the BIN directory, this is made possible through <add assembly="*" > of machine.config.
- <@ Import > links the specified namespace to the page to allow use shorter class names instead of using fully qualified name.
- <@ Reference > is used to establish a dynamic link between the current page and the specified page. <@ Reference Page|Control="" >
- <% inline code %> <%= inline expression %> The inline expression is basically a shortcut for Response.Write().
- Only controls that are a part of the <form> can persist their state across page requests.
- When page is loaded, asp.net runtime parses the source code and creates instances of all controls marked with the runat attribute.
- Class is a Reference Type that defines a blue print of an idea (template). It contains Data (Constants and Fields) and defines its behavior through Methods, Properties, Constructs and Events.
- Objects are created from classes and have physical memory allocated to them. Objects are created on a Heap.
- Object pointers and are stored on Stack.
- Structs are Value Types similar to classes but uses Stack memory. Value Types cannot be inherited.
- Property provides access to the characteristics of the class / object. Property is not a storage location.
- Namespaces allow you to organize classes into logical groups. This will also help in avoiding naming conflicts. System namespace is the root namespace for all base classes defined in FCL.
- Response.Output.Write() allows formatted output compared to Response.Write()
- ASP.NET supports only one language for a page.
- A solution groups one or more projects.
- A directive can be placed anywhere in a page but is usually placed at the top.
- Application variables must be initialized in C#.
- src - will allow dynamic compilation of the class at runtime
- A protocol is a set of rules that describe how two or more items communicate over a medium, such as the Internet.
- Applications that run under the CLR are called **managed code** because the CLR takes care of many of the tasks that would have formerly been handled in the application's executable itself. Managed code solves the Windows programming problems of component registration and versioning (sometimes called DLL hell) because the assembly contains all the versioning and type information that the CLR needs to run the application. The CLR handles registration dynamically at run time,

rather than statically through the system registry as is done with applications based on the Component Object Model (COM).

- State variables are ApplicationState, SessionState and ViewState variables.
- Application variables must be initialized in C#.
- To read one web form's ViewState from another, you must firt set the EnableViewStateMac attribute in the web form's Page directive to false.
  <%@Page ..... EnableViewStateMac="false" .....%>
- Transfer and .Execute works with web forms only, trying to navigate to an HTML page using one of these methods results in a run-time error.
- Visual studio does not support CodeBehind if it is implemented through the Src attribute.
- ISAPI extensions (.aspx, .cs, .asmx …) are run-time modules that handle web resources.
- .aspx files are assigned to ASPNET_ISAPI.DLL module of IIS.
- IIS5.0 INETINFO.EXE hosts Aspnet_isapi.dll.
- Aspnet_isapi.dll does not process the .aspx files but acts as a dispatcher and it routes the request toward asp.net worker process (aspnet_wp.exe)
- Resource mappings are stored in the IIS metabase.
- Configuration can be carried out like: multiple CPU's can handle 1 WORKER PROCESS / 1 CPU runs 1 aspnet_wp.exe (worker process).
- Process Recycling: <processModel> element of the machine.config file defined threshold values for all these parameters. The aspnet_isapi.dll checks the overall state of the current worker process before forwarding any request to it. If the process breaks one of these measures of good performance, a new worker process is started to serve the next request. The old process continues running as long as there are requests pending in its own queue. After that, when it ceases to be invoked, it goes into idle node and is then shut down. In this way, memory leaks and run-time anomalies are promptly detected and overcome.
- Assemblies generated for the asp.net page are cached in the temporary asp.net files folder WINDOWS\Microsoft.NET\Framework\v1.1.4322\Temporary ASP.NET Files
- An Assembly contains the IL code and metadata.
- Assemblies are the deployment units in the .NET framework.
- The code that runs within the CLR is called managed code.
- Namespace groups logically and functionally related classes.
- a portable executable file cannot be executed if it does not contain an assembly manifest. The runtime uses the information in the assembly manifest to load the assemblies into the runtime.
- CLR can terminate an AppDomain without stopping the window process
- It is much cheaper to create, monitor and maintain an AppDomain than a Process.
- An application boundary / AppDomain defines the scope of an application
- A windows process runs only one application.
- Windows achieves isolation thru process boundaries.
- AppDomain is the basic unit of isolation for running application in CLR and an AppDomain achieves isolation through application domain boundaries
- CLR allows several AppDomains to run within a single window process
- AppDomain contains its own set of code, data and configuration settings
- Windows process is an application that's running and had been allocated RAM memory.
- CLR runtime also provides a common debug engine, which enables the debugging process.
- Metadata is the info that describes the code and the types (classes) that the code contains.
- Identity objects represent the user on whose behalf the code is running.
- WindowsIdentity represents local or domain user accounts.
- FormsIdentity is used in the implementation of form-based authentication.
- GenericIdentity represents generic users (windows and non-windows users too).

MapPath converts virtual paths to physical paths.

Basic application structure
Every application contains a special directory /bin and two special files Web.config & Global.asax.
/bin directory contains custom components and controls (in the form of assemblies). Any component or control added to the /bin directory is automatically visible to all pages executing within the application.
Web.config specifies configuration information for the entire application.
Global.asax
Global.asax contains subroutines that handle applicationwide events and objects declared with application scope.

Shared or Strong Names
A name that consists of an assembly's identity—its simple text name, version number, and culture information (if provided)—strengthened by a public key and a digital signature generated over the assembly.

Managed Vs Unmanaged Code
Code that runs within the CLR is called managed code. Code that targets the runtime is known as managed code, while code that does not target the runtime is known as unmanaged code. Code that you develop with a language compiler that targets the runtime is called managed code. To enable the runtime to provide services to managed code, language compilers must emit metadata that describes the types, members, and references in your code. Metadata is stored with the code; every loadable common language runtime portable executable (PE) file contains metadata. The runtime uses metadata to locate and load classes, lay out instances in memory, resolve method invocations, generate native code, enforce security, and set run-time context boundaries.

Process and Application
A process is an instance of a running application. An application is an executable on the hard drive or network. There can be numerous processes launched of the same application (5 copies of Word running), but 1 process can run just 1 application.

Locking
Optimistic locking locks the record being updated only during the call to Update.
Pessimistic locking locks the record as soon as Edit is called and keeps it locked until the Update call completes or you move to a new record

System.Threading.ReaderWriterLock is the perfect tool to lock the application cache during an update

ASHX Files
ASHX files contain **HTTP handlers** - software modules that handle raw HTTP requests received by ASP.NET. ASHX files provide developers with a convenient way to deploy HTTP handlers without customizing CONFIG files or modifying the IIS metabase.

Polymorphism
Polymorphism is the ability for classes to provide different implementations of methods that are called by the same name. Polymorphism implies the existence of an entity in different forms. Polymorphism allows classes to provide different implementations of a method that can be called in the same way.
Polymorphism can be of three types-inheritance polymorphism, interface polymorphism and polymorphism through abstract classes.

Inheritance, polymorphism, encapsulation and operator overloading
Operators that cannot be overloaded are
Conditional logical operator &&, ||
Array indexing operator []
Cast operator ()
Assignment operator =
And the list continues
=, ., ?, :, ->, new, is, sizeof, typeof

Unsafe Code
Using pointers for advanced COM programming which requires structures with pointers in it.
Or calling Win32 API's of Platform SDK may require pointers.

Indexers
A class member that permits instance of a class or structure to be stored in the same way as arrays. Indexers are similar to properties, except that the get and set accessors of indexers take parameters, while property accessors do not.

-Security
- -Forms cookie based authentication [asp.net]
- -Windows authentication [asp.net]
- -Passport authentication [asp.net]
- -Client certificates [IIS]

ASPCompat
AspCompat is an aid in migrating ASP pages to ASPX pages. It defaults to false but should be set to true in any ASPX file that creates apartment-threaded COM objects--that is, COM objects registered ThreadingModel=Apartment.

Allocating memory
When a process is initialized the CLR reserves a contiguous address space called as managed heap. In Unmanaged memory, the pointers to memory are maintained in linked–list data structures. The new operator checks whether the memory required by the object is available on the heap.

Locating Assemblies
Strong Name Assemblies
GAC is the first place that the CLR looks for a strong-named assembly. If the assembly cannot be found in GAC, then the CLR will look at the <codeBase> elements of the machine.config file. Even if the assembly is not found then CLR resorts to *probing* (performs a search through a series of directories relative to or below the root directory (APPBASE) of the application.).
Weak Assemblies
GAC is never searched for weak assemblies.

Publisher Policy Files
A publisher policy file is placed in GAC along with an assembly to inform all applications that utilize the assembly that the applications should use the latest version of the assembly. This behoviour can be overridden in app.config by setting the apply attribute of the publisherPolicy element to No.

Early Binding / Late Binding
When you declare a variable using the generic data type Object, you are late binding to the object. Late binding requires a great deal of overhead, and it adversely affects the performance of an application. Because C# doesn't know anything about the members of a late-bound object, the compiler has no way of knowing whether you're using a member correctly—or even if the member you're referencing exists. This can result in a runtime exception or some other unexpected behavior.

When an object is early bound (declared as a specific type of object), C# is able to gather the necessary ID information at compile time. This results in considerably faster calls to object members at runtime. In addition, C# can validate a member call at compile time, reducing the chance of errors in your code.

The following are important reasons to use early binding:
1. Speed, 2. Objects, their properties, and their methods appear in IntelliSense drop-down lists. 3.The compiler can check for syntax and reference errors in your code so that many problems are found at compile time, rather than at runtime.

```
Object o = new Car();       //causes object reference to be late-bound
Car c = new Car();  //causes object reference to be early-bound which results in faster execution
```

To explicitly release an object, set the object variable equal to null, like this:
objMyObject = null;

When a client requests an ASPX page from the Web server, the request travels through many steps before ultimately ending up as HTML displayed by the client's browser. First, the request is processed by IIS and routed to the appropriate ISAPI extension. The ISAPI extension for ASP.NET (aspnet_isapi.dll) routes the request to the ASP.NET worker process.
At this point, the request begins to interact with classes that you are used to dealing with. The request is passed to an HttpApplication. Usually this is the class created in the codebehind file for Global.asax. The HttpApplication then passes the request through any number of HTTP Modules. These classes implement the IHttpModule interface and have a chance to modify the request (or even halt the processing of it) before it gets passed on to the next module. ASP.NET provides some standard modules which provide functionality you're probably familiar with, including FormsAuthenticationModule, PassportAuthenticationModule, WindowsAuthenticationModule, and SessionStateModule, all of which provide exactly the functionality that their names imply.
Ultimately, the request ends up at an IHttpHandler, the most common of which is System.Web.UI.Page. Inside the IHttpHandler.ProcessRequest method, Page raises appropriate events (like Init, Load, and Render), handles ViewState, and provides the programming model for ASP.NET.

Page Redirection or Transfer

- Response.Redirect ()
    - Is client side redirection?
    - Can connect to any. aspx pages / non-.aspx resources on any server
    - Can pass info through URL
    - creates a 302 response header (object moved) status code + target url
- Server.Transfer ()
    - is server side redirection.
    - Can be used for. aspx pages only
    - .aspx page must be on the same server and should not contain URL
    - to preserve the original page data - Server.Transfer("a2.aspx",True) but in this situation keep the EnableViewStateMac=false on the destination page
    - In .transfer from 1.aspx to 2.aspx, the client url will show 1.aspx even though the page data will be from 2.aspx
- Server.Execute ()
    - Executes an .aspx within the current requested page (Similar to a method call)
    - Need to keep the EnableViewStateMac=false on the called page

Intrinsic objects

- Intrinsic objects of the page are Server, Application, Session, Cache, Request, Response, and Trace.
- Intrinsic objects allow ASP.NET to enable low-level access to the web application framework.

HttpContext

HttpContext.Handler / HttpContext.Items can be used to access one pages variables from another page. Context object gives access to the current HTTP request and response. CONTEXT object exposes a key-value collection via the ITEMS property in which you can add values that will be available for the life of the current request (like REQUEST scope of coldfusion).

ASP.NET Introduction

To convert a HTML page to an ASP.NET page, the following 4 modifications are required

- The file name should end with an .aspx extension
- Add Runat="Server" attribute to each of the form tags
- Instead of using the "Name" attribute use the "Id" attribute.
- Convert the <form method="post" action="xyz.aspx"> to <form Runat="Server">
  Only one <form runat="server"> is allowed per .aspx page

HtmlGenericControl

Any html tag (including future html controls) can be represented through HtmlGenericControl control and in case of any new attribute; the attribute can be passed in the control without causing any error.

Every control has a visible and enabled attribute.

IsPostBack

Page.IsPostBack property with the load event to execute code only when a page is first loaded.

Directive

A directive controls how an ASP.NET page is compiled.

```
<%@ Page Trace="True" %>
<%@ Import Namespace="System.Web.Mail" %>
```

Code Declaration Blocks

```
<Script language="C#" runat="server">
    . . .
</Script>
```

Code Render Blocks

```
<% strSomeText = "hello world" %>
<%= strSomeText %>
```

Server Side Comments

```
<%-- server-side comment --%>
```

Server Side Comments

```
<!-- #INCLUDE file="include.aspx" -->
<!-- #INCLUDE virtual="/myDirectory/include.aspx" -->
```

Rich controls

Calendar
AdRotator

Params

params collection also represents QueryStrings, ServerVariables, and Cookies.

Page navigation - todo

Response.Redirect()
Server.Transfer()

Adding Controls to a Page

Control.Add();

File Upload

```
<form EncType="multipart/form-data" Runat="Server">
    <input Id="tbFileUp" Type="File" Runat="Server" />
    <asp:Button Text="Upload File" OnClick="Button_Click" Runat="Server" />
</form>
```

```
private void Button_Click(object sender, System.EventArgs e)
{    tbFileUp.PostedFile.SaveAs("c:\sample.txt");

}
```

Asp.net page / Component / HttpContext
Instrinsic objects like Application, Session, Cache or Trace are available as properties of a Page class. If you want to access these objects in a component, you must first get a reference to the HttpContext object.

`System.Web.HttpContext.Current` returns an object that represents the current `HttpContext`.

IE Webcontrols
- TreeView
- Toolbar
- TabStrip

To do chapter 7 – aspnet unleashed

Web.config
Web.config specifies configuration information for the entire application.

HTTP handlers
- Both HTTP handlers and HTTP modules enable you to gain low-level access to HTTP request and responses.
- Typical uses for handlers include implementations of custom authentication schemes and custom filters. Ex. You can create a handler that authenticates requests for image files, or you can create a handler that automatically transfers request for one file to another file.
- HTTP handlers perform many of the same functions as an ISAPI extension.
- ```
  IHttpHandler()
  { IsReusable;

    ProcessRequest()
    { }
  }
  ```
- ```
  <configuration>
    <system.web>
         <httpHandlers>
              <add verb="*" path="*" type="ProductsHandler,ProductsHandler" />
         </httpHandlers>
    <system.web>
  </configuration>
  ```

HTTP module

- HTTP moduleis similar to a handler in that it enables you to gain low-level access to HTTP requests and reponses but unlike a handler, enables you to participate in the processing of every request.
- Http module can be used to perform custom logic on each page request.
- ASP.NET includes several standard modules for managing state and implementing authentication schemes. The output cache, session state, forms authentication, and windows suthentication are implemented as modules.
- ```
  IHttpModule()
  { Init()
    { }

    Dispose()
    { }
  }
  ```
- ```
  <configuration>
    <system.web>
         <httpModules>
              <add name=”AuthModule” type=”myModules.AuthModule,AuthModule” />
         </httpModules>
    <system.web>
  </configuration>
  ```

User Control

- A user control is an asp.net page that has been converted into a control. It is a chunk of UI + processing logic packaged together to make it work as a pluggable component.
- To build a user control create an .ascx [content file] file.
- User control need be included in the project in which the controls are used. A copy of the control must exist in each web application project in which the control is used.
- To refer a user control

```
<%@ Register TagPrefix="PP" TagName="NN" Src="Header.ascx" %>
<PP:NN ID="ctlHeader" Runat="Server" comanyName="Suhit Co." />
```

- If you want the user control to participate in viewstate include it in the <form> tag.
- User controls have to be hosted on a web form; they cannot be independently requested.
- User controls cannot be added to the toolbox.
- User controls support flow layout only.
- User control code is initialized after the web form loads.
- User controls Load () runs after the web forms Load ().
- <%@ Page .... %> directive is NOT ALLOWED in the web user control
- Values can be passed into a user control thru attributes or also can be set programmatically like

```
uc1.MyTitle="suhit co.";
```

myUC.ascx

```
<script language="C#" runat="server">
     public String MyTitle ="";

     public void Page_Load(Object Sender, EventArgs e)
     {if (Page.IsPostBack == false)
          {  lblTitle.Text = MyTitle.ToString();
          }
     }
</script>
<asp:Label id="lblTitle" runat="server" />
```

myPage.aspx

```
<%-- declare any namespaces --%>
<%@ Register TagPrefix="PP" Tagname="NN" src="myuc.ascx"%>
<PP:NN Id="uc1" MyTitle="abc" Runat="Server" />
```

To load a user control programatically

- <%@ Reference="Usercontrol01.ascx" %>
- `Page.LoadControl()` is used to dynamically load a control programmatically. You need to cast this loaded control before accessing any of properties.

Exposing Properties / Methods / Events in user controls

- If a value is declared public in a user control then it is can be accessed within any page that contains the user control.
- All public variables declared in the user control file are exposed as properties of the user control.
- Page_Load() contained in a user control is completely distinct from the Page_Load() in the parent page.

Assembly

- The primary uit of .NET application is an assembly.
- An assembly is a self-describinh collection of code, resources, metadata and manifest.
- The assembly manifest describes the assembly and one or more modules, and the modules contain the source code for the application.
- Each assembly has one and only one assembly manifest.
- Assemblies are the fundamental units of application development and deployment.
- Assemblies are the smallest unit for versioning and configuration security.
- Assemblies are the smallest unit to which .NET grants permissions.
- An Assembly contains
    - Manifest [version no., hash of the file, name, version, public key info]
    - Metadata [version, security identity, resources]
    - IL code
    - Resources
- Assembly attribute
  `[assembly: AssemblyVersion("1.1.*")]`
- Types of assemblies
    - Static assembly
        - A static assembly is created when you compile the program using the compilers and are stored on the hard disk in the form of a portable executable (.exe or .dll) file.
    - Dynamic assembly
        - A dynamic assembly is created at runtime when an application requires them. Reflection API allows you to find type info and create objects dynamically at runtime.
    - Private assembly
        - A private assembly is installed in the installation directory [/bin] of an application and is accessible to that application only.
    - Shared assembly / Global Assembly
        - A shared / global assembly is shared by multiple application [can be used across applications] and must be loaded into the GAC (Global Assembly Cache - .NET equivalent of the system registry). A shared assembly has a strong name [assembly name, version, culture info, digital signature, public key]. All .NET System classes are shared assemblies.
- Portable Executable is a Static Assembly.
- CLR cannot execute PE if the assembly does not contain the Manifest.
- Shared or Strong Names
  A strong name consists of an assembly name, version, culture info, digital signature and public info.
- Namespace contains logically- and functionally- related classes and divides an assembly into a logical grouping of types.
- If the assembly resides in the applications **\bin** directory, web pages in the application do not need the <%@Assembly … %> directory.

.NET framework tools

- **Ngen.exe** [Native Image Generator Tool] – pre-compiles code into processor-specific native code.
- **Sn.exe** [Strong Name Tool] – used to give assemblies strong names for unique identification. Allows verification, key pair and signature generation

```
c:\> sn –k  MyKey.snk                → creates a key pair
c:\> sn –r  abc.dll    MyKey.snk     → re-assigns a strong name key to assembly
using System.Reflection;
[assembly: AssemblyKeyFile ("MyKey.snk")]
```

- **Gacutil.exe** [Global Assembly Cache Tool] – manages the contents of GAC. Is used to install/uninstall the assembly in the GAC

```
c:\> gacutil  /i <assembly>
```

- **Caspol.exe** [Code Access Security Policy Tool] – manages security policies. Is used to view and modify security information.

- **PermView.exe** [xyz Tool] –to view security (permission) settings required by a component.
- **Al.exe** Assembly Linker Tool [AL Tool] enables you to create a single manifest file by combining one or more modules and resources files. It is basically used for modules, AL Tool generates a file with an assembly manifest from the modules or resources file.

```
c:\> al /t.exe /out:abc.exe abc.dll
```

- **Secutil.exe** is used to manage strong name public key information or Authenticode certificates signatures. [to extract X.509 certificate from components]
- **Certmgr.exe** [xyz Tool] – manages X.509 certificates.
- **Tlbimp.exe** [Type Library Importer Tool] – [COM --> ASSM] The Type Library Importer converts the type definitions found within a COM type library into equivalent CLR definitions.
- **Tlbexp.exe** [Type Library Exporter Tool] – [allows ASSM to be used by COM/VB6] is used to export assembly info to a type library so that COM/unmanaged code can call this assembly
- **Regasm.exe** [Assembly Registration Tool] – [allows ASSM to be used by COM/VB6] works similar like Tlexp.exe but also registers the assembly by adding the necessary entries to the registry.
- **Disco.exe** is used to discover the URLs of web services that are running on a server
- Whe you compile managed code; the compiler converts the source code to MSIL code. Ilasm.exe tool is then used to generate a portable executable file from MSIL.
- **ILASM.exe** [IL Assembler Tool] - is used to generate a portable executable file from MSIL code
- **ILDASM.exe** [IL Disassembler Tool] - is used to view the metadata and dissembled code of PE file
- **Regsvr32.exe** [xyz Tool] – is used to register COM components in the windows registry
- **Fuslogvw.exe** [xyz Tool] – provides info regarding an application's inability to bind to an assembly
- **Wsdl.exe** [xyz Tool] – used to create a WSDL file for your component
- **Dotnetfx.exe** [xyz Tool] – used to repair corrupted .net framework

Configuration and Security

- A configuration file is an XML document that contains predefined elements.
- <codebase> element deermines the location of the assembly.
- .NET provides 3 sets of configuration files
    - Machine configuration file
        - Machine.config
        - Located at %runtime installation path%\config\
        - Settings here affect all applications.
    - Application configuration files
        - Web.config, ClassName.exe.config (App.exe)
        - Settings required for configuring an individual application.
    - Security configuration files
        - Contains security permissions for a hierarchy of code groups.
        - They define enterprise-level, machine-level, and user-level security policies.
- .NET provides Mscorcfg.msc (Microsoft Configuration Tool) or Caspol.exe (Code Access Security Policy Tool) tools to configure these security files.
- Mscorcfg.msc is a Microsoft management console MMC snap-in that enables you to manage and configure assemblies located in GAC.
- CASPOL.EXE allows you to grant and modify permissions granted to code groups at the user policy, machine-policy, and enterprise policy levels.

Garbage collection

Introduction
Application Roots
Implementation
    Phase I Mark
    Phase II Compact
Finalization
Performance optimizations
    Weak References
    Generations

Introduction

- GC relieves the programmer of the burden of manual memory management.
- Garbage collector is a low-priority thread under normal circumstances and always runs in the background.
- GC performs periodic checks on the managed heap to identify objects that are no longer required by the program and removes them from memory.
- GC also manages circular references (previously a common form of memory leak).

Application Roots

Every application has sets of roots. Roots point to the storage location on the managed heap. Each root either refers to an object on the managed heap or is set to null. An application's roots consist of global and static object pointers, local variables, and reference object parameters on a thread stack. JIT and CLR maintain the list of application roots.

The garbage collector uses this list to create a graph of objects on the managed heap that are reachable from the root list. When GC starts running, it navigates the application root list and checks for directly/ indirectly referenced objects and flag these objects as reachable. The unreachable objects on the managed heap are removed and the reachable objects are compacted and are copied to a new contiguous location, after this the GC updates the pointers in application root list.

Implementation – Mark / Compact

GC is done using tracing collection and CLR specifically implements the MARK/COMPACT collector for this purpose.

- Phase I: Mark - When the garbage collector starts running, it makes the assumption that all objects in the heap are garbage. In other words, it assumes that none of the application's roots refer to any objects in the heap. It starts walking the roots and building a graph of all objects reachable from the roots. Once all the roots have been checked, the garbage collector's graph contains the set of all objects that are somehow reachable from the application's roots; any objects that are not in the graph are not accessible by the application, and are therefore considered garbage.
- Phase II: Compact - Move all the live objects to the bottom of the heap, leaving free space at the top. The garbage collector now walks through the heap linearly, shifting the non-garbage objects down in memory (compacting, and all the non-garbage objects).

Finalization

.NET Framework's garbage collection implicitly keeps track of the lifetime of the objects that an application creates, but fails when it comes to the unmanaged resources (i.e. a file, a window or a network connection) that objects encapsulate.

The unmanaged resources must be explicitly released once the application has finished using them. .NET Framework provides the Object.Finalize method: a method that the garbage collector must run on the object to clean up its unmanaged resources, prior to reclaiming the memory used up by the object. Since Finalize method does nothing, by default, this method must be overridden if explicit cleanup is required.

Finalize method is called once when Garbage collection gets around to cleaning up an object. The potential existence of finalizers complicates the job of garbage collection in .NET by adding some extra

steps before freeing an object. Whenever a new object, having a Finalize method, is allocated on the heap a pointer to the object is placed in an internal data structure called Finalization queue.

When an object is not reachable, the garbage collector considers the object garbage.

The garbage collector scans the finalization queue looking for pointers to these objects. When a pointer is found, the pointer is removed from the finalization queue and appended to another internal data structure called Freachable queue, making the object no longer a part of the garbage. At this point, the garbage collector has finished identifying garbage. The garbage collector compacts the reclaimable memory and the special runtime thread empties the freachable queue, executing each object's Finalize method.

The next time the garbage collector is invoked, it sees that the finalized objects are truly garbage and the memory for those objects is then, simply freed.

Thus when an object requires finalization, it dies, then lives (resurrects) and finally dies again. It is recommended to avoid using Finalize method, unless required. Finalize methods increase memory pressure by not letting the memory and the resources used by that object to be released, until two garbage collections. Since you do not have control on the order in which the finalize methods are executed, it may lead to unpredictable results

~Destructor is simply a shortcut for declaring a `Finalize()` method that chains up to its base class. It is not legal to call a destructor explicitly; the garbage collector will call the destructor.

Dispose() and ~Destructor()
Class instances often encapsulate control over resources that are not managed by the runtime, such as file handles, network connections, window handles (HWND), database connections, and so on. Therefore, you should provide both an explicit and an implicit way to free those resources. Dispose() and finalize() are called finalizers.

Dispose() is one such method that provides Explicit control and is provided by the IDisposable Interface. The consumer of the object should call this method when it is done using the object. Dispose() can be called even if other references to the object are alive. If you have resources that need to be reclaimed as quickly as possible, provide a Dispose() method that gets called explicitly.

If you do handle precious unmanaged resources (such as file handles) that you want to close and dispose of as quickly as possible, you ought to implement the IDisposable interface. The IDisposable interface requires its implementers to define one method, named Dispose(), to perform whatever cleanup you consider to be crucial. The availability of Dispose() is a way for your clients to say, "Don't wait for the destructor to be called; do it right now."

If you provide a Dispose() method, you should stop the garbage collector from calling your object's destructor. To stop the garbage collector, you call the static method, GC.SuppressFinalize(), passing in this reference for your object. Your destructor can then call your Dispose() method.

```
using System;
class Testing : IDisposable
{     bool is_disposed = false;

      protected virtual void Dispose(bool disposing)
      {     if (!is_disposed) // only dispose once!
            {     if (disposing)
                  {     Console.WriteLine("Not in destructor, OK to reference other objects");
                  }

                  // perform cleanup for this object
                  Console.WriteLine("Disposing...");
            }

            this.is_disposed = true;
      }
```

```
      public void Dispose()
      {     Dispose(true);

            // tell the GC not to finalize
            GC.SuppressFinalize(this);
      }

      ~Testing()
      {     Dispose(false);
            Console.WriteLine("In destructor.");
      }
}
```

Using statement
Because you cannot be certain that your user will call Dispose() reliably, and because finalization is nondeterministic (meaning you can't control when the garbage collector will run), C# provides a using statement, which ensures that Dispose() will be called at the earliest possible time. The idiom is to declare which objects you are using and then to create a scope for these objects with curly braces. When the close brace is reached, the Dispose() method will be called on the object automatically:

```
using System.Drawing;
class Tester
{     public static void Main()
      {     using (Font theFont = new Font("Arial", 10.0f))
            {     // use theFont
            } // compiler will call Dispose on theFont

            Font anotherFont = new Font("Courier",12.0f);
            using (anotherFont)
            {     // use anotherFont
            } // compiler calls Dispose on anotherFont
      }
}
```

Destructor is another method that provides Implicit Control on an object. Destructors call the Finalize() method. The garbage collector calls this method at some point after there are no longer any valid references to the object. Note that even if you provide explicit control by way of Dispose, you should provide implicit cleanup using the destructor. Destructor provides a backup to prevent resources from permanently leaking if the programmer fails to call Dispose. Destructors are the methods that contain the cleanup code that is executed before the object is garbage collected. Destructors ensures that even if the client does not call the Dispose() explicitly , the resources used by the object are released from memory on GC. GC during GC process calls the finalize() before releasing memory.

- Destructors cannot be used with structs. They are only used with classes.
- Execution of the destructor for the instance may occur at any time after the instance becomes eligible for destruction.
- Destructors are invoked automatically, and cannot be invoked explicitly.

Weak / Strong References
Weak references are a means of performance enhancement, used to reduce the pressure placed on the managed heap by large objects.

When a root points to an object it's called a strong reference to the object and the object cannot be collected because the application's code can reach the object.

When a object is marked for garbage collection then it means it is weakly referenced. That is an object not has a reference pointer on the stack.

The managed heap contains two internal data structures whose sole purpose is to manage weak references: the short weak reference table and the long weak reference table.

Weak references are of two types:

- A *short weak reference* doesn't track resurrection. i.e. the object which has a short weak reference to itself is collected immediately without running its finalization method.
- A *long weak reference* tracks resurrection. i.e. the garbage collector collects object pointed to by the long weak reference table only after determining that the object's storage is reclaimable. If the object has a Finalize method, the Finalize method has been called and the object was not resurrected.

Generations

Garbage collector divides the objects on the managed heap into 3 generations 0, 1, 2. Ideally

Generation 0 will be in smaller size and contains recently created objects,
Generation 1 will be in medium size and
Generation 2 will be larger.

When Generation 0 is full and it does not have enough space to occupy other objects but still the program wants to allocate some more memory for some other objects, then the garbage collector will be given the REALTIME priority and will come in to picture. Now the garbage collector will come and check all the objects in the Generation 0 level. If an object's scope and lifetime goes off then the system will automatically mark it for garbage collection. Here in the process the object is just marked and not collected. Garbage collector will only collect the object and free the memory. Garbage collector will come and start examining all the objects in the level Generation Zero right from the beginning. If it finds any object marked for garbage collection, it will simply remove those objects from the memory. It will look which are all the objects survive after the sweep (collection). Those objects will be moved to Generation 1 and now the Generation 0 is empty for filling new objects. If Generation One does not have space for objects from Generation Zero, then the process happened in Generation 0 will happen in Generation 1 as well. This is the same case with Generation 2 also. If all the generations are filled with the referred objects then MemoryOutofRangeException will be thrown.

Controls

HTML controls
- Simple html tags,
- They do not maintain their state across postback

HTML Server controls
- html tags with runat=”server”
- Can be accessed on the web server
- Provided for migration old code to .NET
- ServerClick / ServerChange are the default events

Web Server controls
- provide higher level of abstraction
- preferred controls
- have built-in automatic browser detection capabilities
- Support event bubbling
- for asp:button if the CommandName is provided then the Button is Command Button. Command button is useful when you want to pass some info to the EventHandler

Basic Web Controls

```
<asp:Label >
<asp:TextBox TextMode=”Single|MultiLine|Password” AutoPostBack=”True” >
```

Request Validation (enabled by default) is added to handle cross-site scripting attacks. You cannot enter html tags like <b> or Expressions like <script> (in the text field). Request Validation is implemented behind the scenes by a private class named the *CrossSiteScriptingValidation* class.
- to disable Request Validation

```
<%@ ValidateRequest=”False” %>          page level setting
<Pages ValidateRequest=”False” /> in web.config
```

AutoPostBack property requires client-side JavaScript

```
<asp:Button >          <asp:ImageButton >          <asp:LinkButton >
<asp:RadioButton >     <asp:RadioButtonList >
      <asp:ListItem Text=”Red” />
<asp:CheckBox >        <asp:CheckboxList >
<asp:DropDownList>
<asp:ListBox >
<asp:HyperLink >
<asp:Panel >
<asp:PlaceHolder >

CommandButton
<asp:Button      CommandName=”ADD”
                 CommandArgument=”5”
                 OnCommand=”Button_Command” ... />
```

Validation controls

can also perform validation using client script.
- RequiredFieldValidator - value cannot be empty
- RegularExpressionValidator - validate against a RE
- RangeValidator - data should be within a range
- CompareValidator - compare data against a control value
- CustomValidator - custom logic to validate data
- ValidationSummary

Data controls
- DataGrid
- DataList
- Repeater

Rich web controls
- Calendar
- AdRotator

Mobile controls

- EventHandling
    - Intrinsic events – Click() / Command() +
    - Event Arguments
    - AutoPostBack
    - Bubbled Events
- Client side event handling [JavaScript code]
    - btnSubmit.Attributes.Add(“onMouseOver”, “SomeClientJSCode();”)
    - Validation controls
- client side validation
    - WebUIValidation.js
    - RegisterClientScriptBlock() / IsClientScriptBlockRegistered()
    - Startup…..
    - Array….
    - Hidden…..
    - Validation check fires onFocusOut() of the control
- server side validation
    - Page.IsValid() / Page.Validate()

User controls

Composite controls

Custom controls

There can be only one <form runat=”server”> control on a web page.

User Controls
Created as a text file with an .ascx extension (with an optional code-behind file).
A separate copy of the control is required in each application
Cannot be added to the Toolbox in Visual Studio
Good for static layout

Custom Controls
Custom controls are compiled code components (DLL) that execute on the server.
Only a single copy of the control is required, in the global assembly cache
Can be added to the Toolbox in Visual Studio
Good for dynamic layout

ASP.NET Validation

- Asp.net provides VALIDATION CONTROLS on both SERVER and CLIENT side
- Asp.net ensures that validations are performed on the server side even if they were already performed on the client side.
- If the client side validation fails the server side validation is never perofrmed
- BaseValidator is the abstract class
- Page.IsValid
- Validation is performed after Page Initialization, that is after the Page_Load() is raised but before the event-handling code is called.

Client-side validation

- The validation controls make use of a JavaScript script library that is automatically installed on your server when you install the .NET framework. This library is located in a file named ***aspnet_client/wwwroot/WebUIValidation.js***
- Validation takes place when the focus leaves the element

aspnet_regiis

aspnet_regiis is a command line tool to install/uninstall the script library.

to disable validation

`CausesValidation="False"` property of a control can be used to disable validation.
`<%@ Page ClientTarget="downlevel" %>` directive disables client-side form validation.

Custom validation

- Is used to perform any other complex validation on both SERVER side and CLIENT side
- To validate on the server side double click the custom validator and write the code to validate. In ServerValidate event procedure.
- To validate on the client side write a javascript function, this function gets 2 parameters (src, args) and call this function in the control's ClientValidationFunction.
- args contains the values to validate.
  - args.IsValid
  - args.Value
- Client side vaidation is optional, and if you provide it you should maintain similar validation code in both places.

```
<script language="javascript">
      function callABC(src, args)
      {     var temp = "from function callABC";
                temp += "\nSource and Arguments are always passed";
                temp += "\nargs.Value = " + args.Value;
                temp += "\nargs.IsValid = " + args.IsValid;

            alert(temp);

            args.IsValid = args.Value >= 8;
            alert("args.IsValid = " +args.IsValid);
      }
</script>


<form id="Form1" method="post" runat="server">
```

RequiredFieldValidator

```
<asp:TextBox id="tbName" runat="server"> </asp:TextBox>
<asp:RequiredFieldValidator id="RequiredFieldValidator1" runat="server"
        ControlToValidate="tbName" ErrorMessage="Name is required"
        EnableClientScript=”True” Enable=”True” InitialValie=””
        Display=”Static|Dynamic|None”>
        Show this text here beside this control </asp:RequiredFieldValidator>
```

CompareValidator

```
<asp:TextBox id="tbNum1" runat="server"> </asp:TextBox> is equal to
<asp:TextBox id="tbNum2" runat="server"> </asp:TextBox>
<asp:CompareValidator id="CompareValidator1" runat="server"
        ErrorMessage="Numbers are not equal" ValueToCompare="="
        ControlToCompare="tbNum1" ControlToValidate="tbNum2"> </asp:CompareValidator>
```

RangeValidator

```
<asp:TextBox id="tbRange" runat="server"> </asp:TextBox>
<asp:RangeValidator id="RangeValidator1" runat="server"
       ErrorMessage="Number not in 10-30 range" ControlToValidate="tbRange"
       MaximumValue="30" MinimumValue="10"> </asp:RangeValidator>
```

RegularExpressionValidator

```
<asp:TextBox id="tbRegExp" runat="server"> </asp:TextBox>
<asp:RegularExpressionValidator id="RegularExpressionValidator1" runat="server"
        ErrorMessage="not a valid email id [tested with Regular Expression]"
        ValidationExpression="\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([.]\w+)*"
        ControlToValidate="tbRegExp"> </asp:RegularExpressionValidator>
```

CustomValidator

```
<asp:TextBox id="tbCV" runat="server"> </asp:TextBox>
<asp:CustomValidator id="CustomValidator1" runat="server"
        ErrorMessage="length should be a minimum of 8 letters"
        ControlToValidate="tbCV" ClientValidationFunction="callABC"
        OnServerValidate=”callServerABC”> </asp:CustomValidator>

<asp:Button id="Button1" runat="server" Text="Submit"></asp:Button>
<asp:Button id="Button2" runat="server" Text="Cancel" CausesValidation=”False”> </asp:Button>
```

ValidationSummary

```
<asp:ValidationSummary id="ValidationSummary1" runat="server" ShowMessageBox="True"
        ShowSummary=”True” DisplayMode=”BulletList|List|SingleParagraph”>
</asp:ValidationSummary>

</form>
```

RegisterClientScriptBlock(key, script); **IsClientScriptBlockRegistered(key);**

Allows you to include script blocks in the page. The client-side script is emitted just after the opening tag of the Page object's <form runat= server> element. The script block is emitted as the object that renders the output is defined, so you must include both tags of the <script> element.

```
RegisterClientScriptBlock( "showSaveMessage",
                          "<script language='JavaScript'>
                               function showAlert() {
                                    alert('hello world');
                               }
                          </script>");
```

```
<form method="post" ...>
   <script language='JavaScript'>
      function showAlert() {
         alert('hello world');
      }
   </script>
   ...
</form>
```

IsClientScriptBlockRegistered(key) and IsStartupScriptRegistered(key), determines if a client startup script has been registered with the Page object.You can ensure that the script common to all instances of the server control on the page is generated only once per page load, rather than once per control instance on the page.

RegisterStartupScript(key, script); **IsStartupScriptRegistered(key);**

Similar to the RegisterClientScriptBlock method, this method emits the script just before the closing tag of the Page object's `<form runat="server">` element after all form fields.

```
<form method="post" ...>
   ...
   <script language='JavaScript'>
      function showAlert() {
         alert('hello world');
      }
   </script>
</form>
```

RegisterClientScriptBlock **enables a custom control to register a block of client-side script that the control returns to a browser**. Why does it exist? So the same script block doesn't get returned multiple times if the page contains multiple instances of a control that emits client-side script
RegisterStartupScript is for returning blocks of **client-script *not* packaged in functions**—in other words, code that's to execute when the page is loaded. The latter positions script blocks near the end of the document so elements on the page that the script interacts are loaded before the script runs.

RegisterArrayDeclaration(arrayName, arrayValue);

If you need to create a client-side JavaScript **Array** object with some set values, use this method to add a value to a specific array.

```
RegisterArrayDeclaration("FavoriteFolks", "'Scott'")
RegisterArrayDeclaration("FavoriteFolks ", "'Jisun'")

<script language="javascript">
   <!--
      var FavoriteFolks =  new Array('Scott', 'Jisun');
   // -->
</script>
```

RegisterHiddenField(hiddenFieldName, hiddenFieldValue);

```
RegisterHiddenField("foo", "bar");

<form name="_ctl0" method="post" action="test.aspx" id="_ctl0">
   <input type="hidden" name="foo" value="bar" />
   ...
</form>
```

The actual function name is assigned to the ASP.NET button control via the Add method of the Attributes property of the object.

```
btnSubmit.Attributes.Add("onclick","showAlert()");
```

Validation check fires onFocusOut() of the control.

Web Form's RenderChildren() contains three lines of code:

- Page.OnFormRender(...)
  - RegisterHiddenField() for hidden variables.
  - _VIEWSTATE (hidden formm field, base-64 encoded view state)
  - RegisterClientScriptBlock() script blocks.
- MyBase.RenderChildren(...)
  - RegisterArrayDeclaration() for Array objects
  - RegisterStartupScript() script blocks
- Page.OnFormPostRender(...)
  - the closing form tag </form>

You can programmatically add a client-side attribute using the Attributes collection.
To achieve the following:

```
<input type="text" onfocus="this.style.backgroundColor='yellow';"
                   onblur="this.style.backgroundColor='white';" />
```

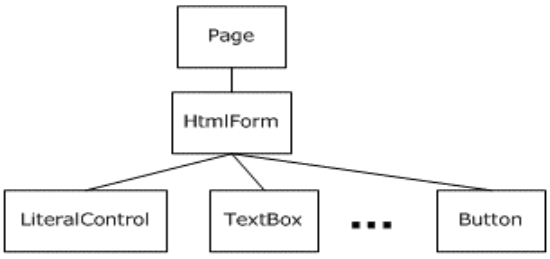
Do the following:

```
TextBoxControl.Attributes("onfocus") = "this.style.backgroundColor='yellow';"
TextBoxControl.Attributes("onblur")  = "this.style.backgroundColor='white';"
```

Imagine we have an ASP.NET Web page with markup like:

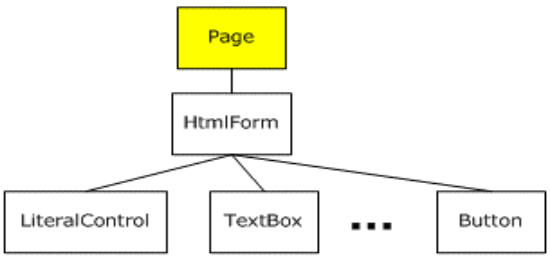
```
<form runat="server">
  Your name: <asp:TextBox runat="server"
... />
   ...
   <asp:Button runat="server" ... />
</form>
```

It would have a control hierarchy similar to the one shown to the right.



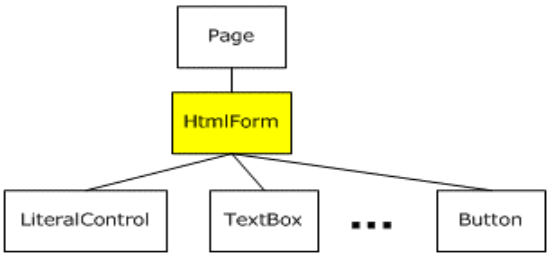
When the page is rendered, the rendering begins at the root of the control hierarchy and recursively proceeds through each of the controls.

Each control is responsible for emitting its own HTML markup based upon its properties and other settings.

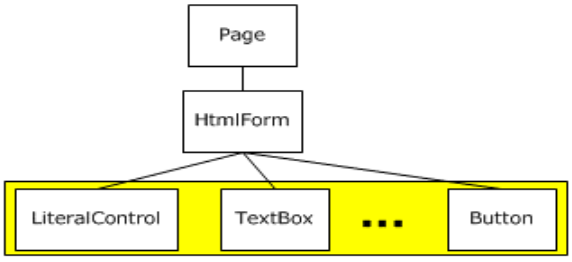


When the `HtmlForm` begins rendering, the opening `<form>` tag is rendered and then the `Page.OnFormRender()` method is called, which emits any script blocks registered via `RegisterClientScriptBlock()`, any hidden form fields added via `RegisterHiddenField()`, and the `__VIEWSTATE` hidden form field. The rendered HTML at this point might look like:

```
<form method="post" ...>
  <script language="JavaScript">
      ...
   </script>
   <input type="hidden" name="__VIEWSTATE"
value="..." />
```



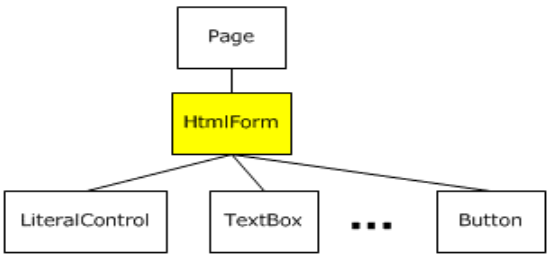
The `HtmlForm`'s `RenderChildren()` method is called, rendering the remaining controls in the control hierarchy. These children controls of `HtmlForm` add their HTML markup to the markup returned to the visitor's browser. The TextBox Web control, for example, is rendered as an `<input type="text" ... />`, while the Button Web control is rendered as an `<input type="button" ... />`.



After `RenderChildren()` finishes running, the `HtmlForm` calls the `Page.OnFormPostRender()`, which adds any `Array` object included via `RegisterArrayDeclaration()`, as well as any script blocks added via `RegisterStartupScript()`. Finally, the closing `</form>` tag is added. This closing markup might look like:

```
   <script language="JavaScript">
       ...
   </script>
</form>
```

Putting everything together, we have the complete HTML markup for the page.

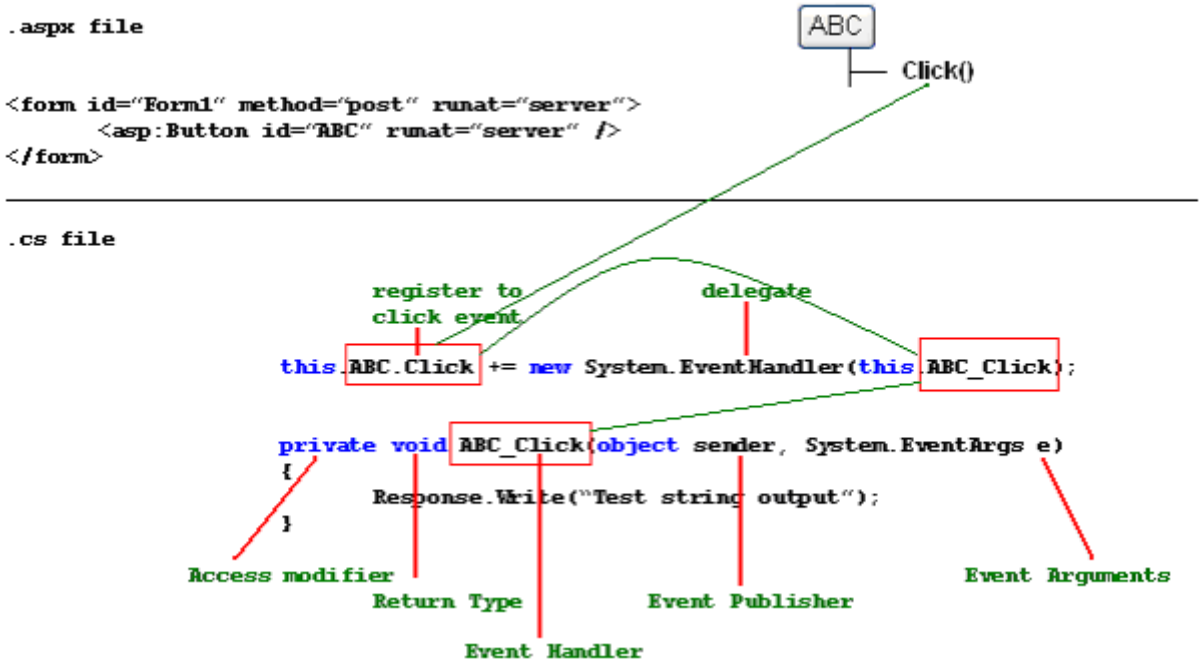


Events

- Intrinsic events
- Event arguments
- AutoPostBack
- Bubbled events
- Server control events
    - o Postback events — These events cause the Web page to be sent back to the server for immediate processing. [causes the page to send the request to the server, but their event procedure is processed last in order of events handled]
    - o Cached events — These events are saved in the page's view state to be processed when a postback event occurs. [these events are collected while the page is displayed and then processed once the page sends a request to the server]
    - o Validation events — These events are handled on the page without posting back or caching. [occur before the page is returned to the server]. ControlToValidate and Text are the 2 common properties.

event handling
- overiding virtual, protected methods of base class
- using delegates
- using AutoEventWireup

AutoEventWireup event-handling is not as flexible as event handling through delegates because the event handlers are required to have specific names. Therefore the AutoEventWireup is set to false in VS.NET.

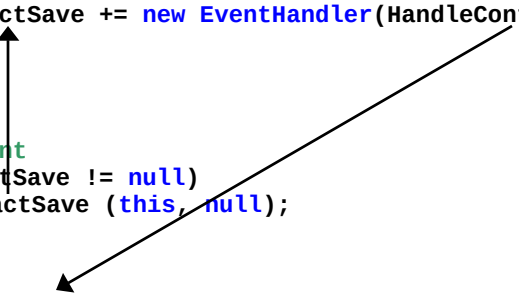


```
public class Contact
{     //define the event
      public event EventHandler EventContactSave;

      //constructor
      public Contact ()
      {     this.EventContactSave += new EventHandler(HandleContactSave);
      }


      public void OnSave()
      {     //raise the event
            if (EventContactSave != null)
            {     EventContactSave (this, null);
            }
      }

      public void HandleContactSave(Object sender, EventArgs e)
      {
      }
}
```



```
//define the delegate
public delegate void DelXYZ (Object sender, EventArgs e);

// define the event
public event DelXYZ TestEvent;

// register the event with the delegate
this.TestEvent += new DelXYZ (ABC);

// notify the registered object about the event
public virtual void OnTestEvent(EventArgs e)
{     // if any object is registered with the event
      if (TestEvent != null)
      {     // raise the event | notify that object
            TestEvent (this, e);
      }
}

// some EventHandler
public void ABC (Object sender, EventArgs e)
{}
```

Delegates

- Delegate is essentially a type-safe function pointer. It allows you to pass a reference to the entry point for a method and invoke that method without making an explicit method call.
- Delegate is a special class [reference type] whose object is capable of storing references to methods with a particular prototype, i.e., it defines the signature for a method call.
- Delegates allow you to attach a single event handler to several events.
- OnInit() is the appropriate place for attaching a delegate for event handling.
- Delegates are Marshal-by-value objects.

```
public class CLASS1
{     //define the delegate
      private delegate string MyDelegate(string e);

      //define the event
      public event MyDelegate TestEvent;

      public CLASS1()
      {     //register event with delegate
            MyDelegate md += new MyDelegate(METHOD1);
            this.TestEvent += new MyDelegate(METHOD2);
      }

      private string METHOD1(string s)
      {     return “Hello ” + s;
      }

      public string METHOD2(string s)
      {     string msg = md(“Ketan Jetty”);
      }
}

public class Form1 : Form
{
      private Button button1;
      private Button button2;

      public Form1()
      {     this.button1 = new Button();
            this.button2 = new Button();
            this.button1.Click += new System.EventHandler(this.button1_Click);
            this.button2.Click += new System.EventHandler(this.button2_Click);
      }

      private void button1_Click(object sender, System.EventArgs e)
      {
      }

      private void button2_Click(object sender, System.EventArgs e)
      {
      }
}
```

```
//define the delegate
public delegate void DelXYZ (Object sender, EventArgs e);

// define the event
public event DelXYZ TestEvent;

// register the event with the delegate
this.TestEvent += new DelXYZ (ABC);

// notify the registered object about the event
public virtual void OnTestEvent(EventArgs e)
{     // if any object is registered with the event
      if (TestEvent != null)
      {     // raise the event | notify that object
            TestEvent (this, e);
      }
}

// some EventHandler
public void ABC (Object sender, EventArgs e)
{}
```

Steps to PUBLISH an event

- define a type derived from EventArgs
- define a delegate type that specifies the prototype of the event handler
- define an event based on the delegate type
- define a protected, virtual metod raises the event
- when an event occurs invoke the previous method

Steps to HANDLE an event

- implement an event handler with signature specified by the delegate object of event
- create a delegate object specified for the event. This delegate should refer to the event handler method.
- attach the event handler using += operator

- delegates are MBV objects
- a delegate is a class that holds a reference to the method that is called when an event is fired
- delegates enable the methods to be abstract

```
Delegate int SomeDelegate(string s, boolean b);
Private int SomeFunction(string str, boolean bln)
{}
SomeDelegate sd = new SomeDelegate(SomeFunction);
sd(“abc”,true);
```

Error / Exceptions

Exceptions are unusual occurrences that happen within the logic of an application.

Hnadled Exceptions vs UnHandled Exceptions

handled exceptions - exceptions handled by try-catch block
unhandled exceptions - exceptions not handled by try-catch
- exceptions not caught by try/catch blocks
- asp.net stops processing a web page after encountering an unhandled exception
- can be trapped at Page lever or Application Level

FCL provides 2 types of exceptions
ApplicationException - represents exceptions thrown by application
SystemException - represents exceptions thrown by CLR
- these classes exist to differentiate exceptions in application from exceptions in CLR.

try-catch-catch{}-finally

```
try
{   //code
}
catch (FormatException fe)
{   //special error handlers should be before generic exception handler
    //will catch only ApplicationExceptions
}
catch (Exception e)
{   //catch all CLS complaint exceptions
    //will catch only ApplicationExceptions
}
catch
{   //catches all other exception including NON-CLS complaint exceptions
    //will also catch SystemException
}
finally
{   //this block is always called
    //you cannot have more than one finally block for a try block
    //transfer statements like goto, break, continue are not allowed here
}
```

Note:
1. transfer-controls statements such as goto, break, continue, if exist in the try or catch block, transfer will happen only after the finally block execution [if exists].
2. if no matching catch block is found the unhandled exception is propagated to the caller

custom error pages

- Handling at page level.
```
<%@ Page language="c#" errorPage="specificErrorPage.htm" %>
```

- Handling at global level – in web.config
```
<customErrors mode="On|Off|RemoteOnly" defaultRedirect="errGeneric.aspx">
  <error statusCode="403" redirect="errForbiddenPage.htm" />
  <error statusCode="404" redirect="errPageNotFound.htm" />
</customErrors>
```
problem with this redirection is that you will lose the exception information during the redirection process.

error handling at global.asax, application level, page level, page event

when an unhandled exception occurs the following events are fired in successive order

Page.Error - handled by Page_Error() [code written in page]
Application.Error - handled by Application_Error() [code written in Global.asax]

throw() - to throw/raise an exception,
- throw statement is an expensive statement.

custom exceptions

1. always derive a custom exception class from System.ApplicationException
2. naming convention used can be myOwnCustomException
3. implement 3 constructors with signatures as given
   myOwnCustomException()
   myOwnCustomException(string msg) : base(string msg)
   myOwnCustomException(string msg, Exception inner) : base(string msg, Exception inner)
4. important properties of exception class
   - ex.Source
   - ex.Message
   - ex.InnerException
   - ex.StackTrace

Note: operations involving floating-points never produce exceptions

using

```
using(...)
{     // use allocated resources
      // Dispose() is called automatically for objects referenced in parenthesis of the using method
}
```

The best exception handling is to

1. handle the error in the Application.Error event i.e, in the Global.asax page's Application_Error()
2. log the error details
3. mail the error details to admin
4. do the Server.Transfer for HttpUnHandledException to a error display page
5. and let the other errors be handled through web.config <customErrors> tag

error / event logging in code in Application_Error() of GLOBAL.ASAX

```
void Application_Error(Object sender, EventArgs e)
{     // this is a better place to log error events
      String errorMessage = Server.GetLastError().Message;
      String sourceName = "mySource";
      String logName = "myLog";

      //create event log if it does not exist
      if (!EventLog.SourceExists(sourceName))
      {     EventLog.CreateEventSource(sourceName, logName);
      }

      //create an event log and assign its source
      EventLog myLog = new EventLog();
      myLog.Source = sourceName;
      myLog.WriteEntry(errorMessage, EventLogEntryType.Error);

      //clear the error | so that it doesn't trigger subsequent error events
        //or appear to the user in the browser.
      Server.ClearError();
}
```

Authorization / Authentication / Impersonation

Authentication refers to the process of obtaining credentials from a user and verifying their identity. The following are the various authentication modes

- **None**
  Users will not be authenticated at all. Pages can simply be delivered to all comers
- **Windows** [uses Windows authentication module]
  - *anonymous* No authentication, everyone is given access to asp.net application
  - *basic* Users must provide windows username/password, info is sent in clear text across network
  - *digest* Users must provide windows username/password, but info is hashed and sent across network
  - *integrated* Users must provide windows username/password, but info does not travel across network, Kerbos or Challenge/Response protocol is used for user authentication
- **Forms** [uses Forms authentication module]
  Uses an HTML form to request credentials from the user. If the credentials are acceptable, the application sends back an identity key. Credentials are then stored in a cookie.
- **Passport** [uses Passport authentication module]
  Microsoft's centralized authentication and profile service for member sites.
- **IIS authentication**
  IIS supports several alternative methods for authentication. These methods interact with the authentication choices within ASP.NET. IIS authentication can also protect resources that are not controlled by ASP.NET.
- **Custom**
  It is very difficult to design a custom secure authentication scheme. This is not recommended for anyone who doesn't have an advanced understanding of cryptography.

Authorization refers to granting rights based on that identity to use various resources.

Impersonation

- <identity impersonate="false"/> Default setting. ASP.NET will always run with its own privileges. All requests will run in the context either the ASPNET account or the system account. This is true whether you're using anonymous access or authenticating users in some fashion.
- <identity impersonate="true"/> ASP.NET takes on the identity passed to it by IIS. If you're allowing anonymous access in IIS, it means that ASP.NET impersonates the IUSR_ComputerName account that IIS itself uses.

Role-based security allows you to authorize access based on user identity or group membership. IsInRole(), WindowsPrincipal, WindowsIdentity

```
<authentication mode="Windows|Forms|Passport|None">
     <forms     name="name" loginUrl="url"
                protection="All|None|Encryption|Validation" timeout="30" path="/"
                requireSSL="true|false" slidingExpiration="true|false">

           <credentials passwordFormat="Clear|SHA1|MD5">
                <user name="username" password="password"/>
          </credentials>
     </forms>
     <passport redirectUrl="internal"/>
</authentication>
//in Web.config
<authentication mode=”Forms”>
       <forms loginUrl=”frmLogin.aspx” name=”315C15” timeout=”1” />
</authentication>
<authorization>
       <allow users=”*” /> <deny users=”?” />
</authorization>
```

Forms Authentication

- Forms authetication is a *ticket-based* (also called *token-based*) system.
- Forms authentication relies on browser cookies to determine the identity of the user
- ASP.NET classes for forms authentication are located in the `System.Web.Security`
- Forms authentication supports both session and persistant cookies
- Forms authentication uses the Authentication Ticket stored in the cookie to authenticate users each time they request a page.

Parts of forms authentication

- *FormsAuthenticationModule*
  The actual module used for forms authentication.
- *FormsAuthentication*
  - `FormsAuthentication.Authenticate()`
  - `FormsAuthorization.RedirectFromLoginPage()`
    - Creates a cookie [.ASPXAUTH] on the users browser that contains an authentication ticket
    - Encrypts the cookie (authentication ticket) using DES or TripleDES encryption
    - Redirects the user back to the originally requested page (using ReturnUrl query string)
  - `FormsAuthentication.SignOut()`
- *FormsIdentity*
  Represents the identity of the authenticated user / current user.
  - `IsAuthenticated`
  - `Name`
  - `Ticket` - represents the FormsAuthenticationTicket
- *FormsAuthenticatonTicket*
  - `Expired`
  - `IssueDate`
  - `Name`
  - `UserData`

Implementing Forms Authorization:

```
<configuration>
  <system.web>
    <!—- STEP 1: set up the authentication mode in web.config -->
    <authentication mode="Forms">
          <!—- STEP 3: create a login page -->
          <forms
               name = "ckApp1"           <!—- auth. cookie name ckApp1.ASPXAUTH -->
               loginUrl = "login.aspx"   <!—- login page -->
               timeout = "30"
               path = "/"
               protection = "All">

               <!—- credential storage can be at 1.web.xml 2.Xml file 3.Database -->
               <credentials passwordFormat="Clear">
                    <user name="ketan" password="ketan" />
                    <user name="annu" password="annu" />
               </credentials>
          </forms>
    </authentication>

    <!—- STEP 2: deny anonymous access -->
    <authorization>
        <deny users="?" />               <!-- deny anonymous users -->
    </authorization>
  </system.web>
</configuration>
```

Implementing Role-based Authentication

To assign different roles to authenticated users, you need to use the Application_AuthenticateRequest() of the Global.asax file as follows:

```
String[] arrRoles = {“Admins”,”Developers”};
Context.User = new GenericPrincipal (Context.User.Identity, arrRoles);
```

Forms Authentication and Web Farms

- By default, you cannot share the same authetication ticket cookie across multiple servers.
- If you want to use Forms authentication across multiple servers, you need to configure all the servers to share the same validation and decryption keys. This can be done thru the following settings of machineKey in the Machine.config

```
<machineKey     validationKey=”AutoGenerate, IsolateApps”
                decryptionKey=”AutoGenerate, IsolateApps”
                validation=”SHA1” />
```

- AutoGenerate modifier will auto generate a key for an application. Replace the AutoGenerate with a valid key; The key must be 40-128 characters.
- IsolateApps causes different keys to be generated for different applications. If you wan the same key to be used for all applications remove this modifier.

Cookie-less forms authentication

Cookie-less is required for persisting authentication ticket for users of mobile devices and cookies-disabled browsers. If FAT (forms authentication ticket/cookie) is not present, the FormsAuthenticationModule will automatically check the query string for FAT.

```
FixLinks()
{     UpdateHtmlAnchors();       // update anchors
      UpdateUrl();               // update urls
}
```

Errors and Debug tools

- Debugging is the process of locating and fixing coding errors.
- There are 3 types of coding errors
  - Syntax errors
    Syntax errors represent syntax the compile cannot interpret. These are caused due to incorrect keywords, missing semicolon, and others. Syntax error occurs when the compiler cannot compile the code.
  - Run-time errors
    Run-time errors occur when the application attempts to perform an operation that is not allowed, such as division-by-zero.
  - Logical errors
    Logical errors occur when the application compiles and executes correctly, but does not produce the expected results.

Page-Level error handling
Use try/catch for page level error handling

Application-Level error handling
- You can configure custom errors in the web.config

```
<configuration>
  <system.web>
     <compilation defaultLanguage="c#" debug="true" />
     <!—- remoteOnly: hides errors on remote browsers only  -->
     <customErrors mode=”On|Off|RemoteOnly” defaultRedirect=”genError.aspx”>
        <error statusCode=”404” redirect=”notFound.aspx />
     </customErrors>
  </system.web>
</configuration>
```

- You can create global error handler in the Global.asax
  use `Application_OnError` method to handle errors in global.asax

Break Mode
Brake mode allows you to halt program execution and execute code one line at a time.
VS.NET enters brake mode when
- An unhandled exception is thrown
- Execution reaches a STOP statement
- Execution reaches an enabled BREAK POINT

There are 4 types or break points
- Function breakpoints
  Enters a breakpoint when a specified location within a function is reached
- File breakpoints
  Enters a breakpoint when a specified location within a file is reached
- Address breakpoints
  Enters a breakpoint when a specified memory address is reached
- Data breakpoints
  Enters a breakpoint when the value of a variable changes (not available in VC# and VB.NET)

Breakpoint Properties Windows / Properties
Breakpoint properties windows has tree tabs
- Function
- File
- Address

In addition to the three tabs, the breakpoint properties window has two buttons
- Condition button
- Hit Count button

Debugging Window

- Output window
  Displays all of the command-line output from the application as the application is compiled and executed.
- Locals window
  Allows to monitor the values of all the variables in the current procedure. The locals window contain 3 columns NAME, VALUE, TYPE
- Autos window
  Is like an abbreviated form of the locals window. But only displays the variables in the current line and the previous line.
- Watch window
  Used to designate a variable to track.
- Quick Watch window
- Immediate window in the Command Window
  Used to execute procedures, evaluate expressions, or change variables values during debugging.
- Other windows
  Running documents, this, call stack, threads, modules, memory, disassembly, registers

ASP.NET Work Process

(todo – aspnet unleashed chapter 18)

The following settings in `<processmodel>` enable you to automatically restart the ASP.NET process in reponse to problems such as memory leaks, deadlocks, and access violations.

- memoryLimit
- requestQueueLimt
- shutdownTimeout

The following settings in `<processmodel>` enable you to automatically restart the ASP.NET process after a certain amount of time or after a certain number of requests.

- idleTimeout
- requestLimit
- timeout

Trace / Debugging

Debug and Trace classes allow you to produce and log informative messages about your application's conditions (trace a program) without having to interrupt application execution. Trace statements are compiled in the release build, whereas Debug statements are not.

Tracing messages can be sent to various destinations including Output Windows [default], text file, event log, or any custom-defined trace listener. TraceSwitches can be used to change the types of messages being generated without recompiling the application. Debug and Trace classes let you display alerts or write messages based on results within your application.

Tracing is a process of monitoring an executing program by collecting information about program execution. This is a technique for recording events, such as exceptions, in an application.

- <%@ Page Trace="true" %> enables tracing at page level
- `trace.axd` allows application-level tracing

Tracing can be done using 2 different techniques:

- TraceContext class [***System.Web***]
  TraceContext class is responsible for gathering execution details of a web request.
  - message in page output
  - trace.axd (trace viewer utility)
- Trace & Debug classes [***System.Diagnostics***] – also for conditional compilation
  TRACE and DEBUG classes belong to the System.Diagnostics namespace. They have members with the same name. All their members are static. They conditionally compiled, i.e, only when the DEBUG or TRACE symbols are defined.
  - output window [default]
  - text files
  - event log
  - custom trace listener

When you compile a program in the Release configuration, only the TRACE symbol is defined. Debug methods and properties are automatically stripped out of code compiled for release, whereas, Trace methods and properties are retained in release code

Writing trace methods

Write() / WriteLine()	– writes trace output unconditionally
WriteIf() / WriteLineIf()	– writes trace output if the condition is true
Warn()	-
Assert()	– write trace output and display a message box if condition is false Use Assert() to halt an application for an unexpected result. Assert outputs both messages and call stack, whereas WriteXX()/WriteLineXX will output only messages.
Fail()	– write trace output and display a message box if condition is false

Listeners Collection

- All output from the Trace and Debug classes is directed to the Listeners collection.
- This is a collection that organizes and exposes classes that are capable of receiving Trace output.
- Listeners are the classes responsible for forwarding, recording and displaying the messages generated by the Trace and Debug classes.
- These objects receive the tracing information, store it, and then route it to its final destination.
- The Trace Listener decides final destination of the tracing information.
- Listener collection is initialized with one member, which is an instance of *DefaultTraceListener* class.
- System.Diagnostics provides the following three implementations of Trace Listeners:
  - DefaultTraceListener
    This is created automatically and will receive Trace and Output even if no other listeners are attached. Writes to OUTPUT window.
  - TextWriterTraceListener
    Writes to any class that derives from the Stream class ex. CONSOLE or FILE.

- EventLogTraceListener
  Writes to windows EVENT LOG.

To create your custom listener class, inherit from TraceListener class and implement at least Write() and WriteLine().
DEBUG and TRACE share the same Listeners collection. So any listener object that is added to the Trace.Listeners collection is also added to the Debug.Listeners collection

- Logging trace/debug to event log

```
EventLog el = new EventLog("test log");
el.source = “Trace output”;   //will throw an error is not set
EventLogTraceListener eltl = new EventLogTraceListener(el);
```

- Logging trace/debug to a text file

```
FileStream fs = new FileStream("c:\\temp\\appLog.txt",FileMode.OpenOrCreate);
TextWriterTraceListener twtl = new TextWriterTraceListener(fs);
//or
//TextWriterTraceListener twtl = new TextWriterTraceListener("c:\\temp\\appLog.txt");

Trace.Listeners.Add(twtl);

Trace.WriteLine("trace test message");
Debug.WriteLine("debug test message");

Trace.Flush();
//or
//Trace.AutoFlush()     // AutoFlush should be set initially

fs.Close();
```

- using Trace / Debug - log message to event log and text file

```
Trace.Listeners.Add(new EventLogTraceListener(“el”));
Trace.Listeners.Add(new TextFileTraceListener(“c:\\temp\\test.log”));

Debug.Listeners.Add(new EventLogTraceListener(“eventLog”));
Debug.Listeners.Add(new TextFileTraceListener(“c:\\temp\\test.log”));
```

TraceSwitches
TraceSwitches are used to set the parameters that control the level of tracing that needs to be done on a program. Trace switches allow you to enable, disable, and filter tracing output. Typically, a deployed application is executed with its switches disabled. You need not recompile the application for this change to take effect, just restart it.

TraceSwitches are of 2 types
- BooleanSwitch
- TraceSwitch

TraceLevel enumeration

0	Off	
1	Error	[error messages]
2	Warning	[error + warning messages]
3	Info	[error + warning + info messages]
4	Verbose	[error + warning + info + verbose messages]

Code example

```
<!-- TraceSwitch / BooleanSwitch -->
<system.diagnostics>
  <switches>
        <!-- traceswitch | tracelevels (0=none, 1=errors, 2=warnings, 3=info, 4=verbose) -->
        <add name="tsGeneral" value="4" />
        <!--  logging the messages to the log file (0=disable, 1=enable) -->
        <add name="bsLogToFile" value="1" />
        <!--  logging the messages to the Event Log (0=disable, 1=enable) -->
        <add name="bsLogToEventLog" value="1" />
  </switches>
```

```
    </system.diagnostics>

    static TraceSwitch ts = new TraceSwitch("display name","description");
    Trace.WriteLineIf(ts.TraceError,"Trace – test error");
    Debug.WriteLineIf(ts.TraceError,"Debug – test error");
```

To record debug and trace messages on a deployed application, create a TextWriterTraceListener class and add it to the Debug or Trace class's Listeners collection.

```
Debug.Listeners.Add(new TextWriterTraceListener("Results.log"));
Debug.WriteLine("Starting tests.");
Debug.Flush();
```

Conditional Compilation

Conditional Compilation directives are a set of *preprocessing directives* that are used to skip sections of source files for compilation, to report errors and warnings.

```
#if, #else, #elif, #endif
#define, #undef
#warning, #error
#line
#region, #endregion

[Conditional (“DEBUG”)]
[Conditional (“TRACE”)]
#if !DEBUG && !TRACE
#error you should have either DEBUG or TRACE defined
#endif

#if DEBUG
      Debug.WriteLine(“test message 1”);
#else
      Trace.WriteLine(“test message 2”);
#endif

#undef TRACE
#undef DEBUG
```

Debugging

Is the process of finding the causes of errors in a program and locating them and fixing them?

Testing

- Testing is the process of executing a program with the intention of finding errors
- Test Plan is a document that guides the process of testing.
- Testing ensures correctness / robustness / reliability.

Executing Tests

- **INCREMENTAL TESTING (evolutionary testing)**
  The idea of incremental testing is to test the system as you build it. 3 levels of testing are involved in this testing phase.
  - ***unit testing***
    Involves performing basic tests at component level. Involves testing elementary units of the application (usually classes). It is the lowest level of testing.
    Test Drivers are programs specially written to test the classes/components, these are used during the dev phase only, but they are not part of the final product.
  - ***integration testing***
    Tests the integration of two or more units or the integration between subsystems. This verifies that the major subsystems of an application work well with each other.
    - *bottom-up approach*
      In this approach, testing progresses from the smallest subsystem and then gradually progresses up in the hierarchy to cover the whole system. This may require that you write a number of *test-driver* programs to test the integration between the subsystems.
    - *top-down approach*
      starts with the top-level system, to test top-level interfaces, and gradually comes down and tests smaller subsystems. You may be required to write *stubs* (dummy modules to mimic the interface of a module) for the modules that are not yet ready for testing.
    - *umbrella approach*
      focuses on testing the modules that have a high degree of user interaction. This approach enables you to release a GUI based application early and gradually increase functionality.
- **REGRESSION TESTING**
  This should be performed any time a program is modified, either to fix a bug or add a feature. Involves the process of repeating the unit and integration tests whenever a bug is fixed, to ensure that old bugs do not exist and that no new ones have been introduced.

- **Testing Data**
  - Normal data
  - Boundary conditions
  - Bad data

State Management

State Management is the process of maintaining state for a web page across roundtrips. ASP.NET provides the following types of State Management

- Server based state management
  - Application
  - Session
- Client based state management
  - Query string
  - Hidden form fields
  - Cookies
  - View state thru StateBag
- Richer Database bound controls
  - Repeater, DataList and DataGrid control

Application

- Application variables are available throughout the application to all pages and all objects.
- After an item is added to the application state, it remains there until the application is shut down.
- An item added to the application state remains there until the application is shutdown, the Global.asax file is modified or the item is explicitly removed.
- Static Objects – gets a collection of all instances of all objects declared in global.asax using the <object> scope set application
- All write methods to the Application object implicitly apply a writing lock.
- *Use Locks only if you want to shield a group of instruction from concurrent writings*
- It is the global repository of data
- The HttpApplicationState class represents application state.
- Application [“myItem”] = “hello”;
- Whe a request is made, an application instance is created for this request and then the init() event is raised.
- Application lock/unlock, Synchronization
  - Concurrent access to the application state can result in deadlocks, race conditions and access violations.
  - You can use LOCK / UNLOCK to write to application state
  - LOCK is not mandatory to read application state
  - If you forget to UNLOCK after the LOCK, the lock is automatically released when the current request finishes processing, and error occurs, or the request times out.
  - Lock() method locks the application state object as a whole. You cannot selectively lock items in the application state. Because locking application state blocks all other access to any item, using application state unwisely in a high volume web site can have a serious impact on the web site performance.
  - Collections such as ArrayLists and HashTables are not designed to be accessed by multiple threads at the same time. You can create thread-safe versions of the objects by using the Synchronized() method.

```
SynTempArrayList = ArrayList.Synchronized(“tempArrayList”);
```

Global.asax

- Global.asax file contains application state events handlers and application objects.
- The Global.asax file does not have the Page object as its default object, hence, within the Global.asax file, you must use Context to access all intrinsic and other properties that are available in the Page scope.
  Ex. Use Context.Cache instead of Page.Cache
- Modifying the Global.asax file restarts the application and any info stored in the application (or session) state is lost.
- Variables added to the init event survive only one application instance.

Web.Config

- ASP.NET are configured with the web.config.
- ASP.NET uses a hierarchical configuration system.
  Machine.config → Web.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
      <appSettings>
            <add key="odbcDSN" value="DSN=dsn_dbexam70315;Uid=sa;Pwd=pwd;" />
            <add key="oleDSN" value="Provider=SQLOLEDB;Data Source=VIRAT;
                        Initial Catalog=bharat;Uid=sa;Pwd=pwd;" />
            <add key="sqlDSN" value="Data Source=VIRAT;Initial Catalog=bharat;
                        User Id=sa;Password=pwd;" />
            <add key="cs01" value="configuration setting 01 in web.config" />
      </appSettings>

      <system.web>
            <compilation defaultLanguage="c#" debug="true" />

            <customErrors mode="RemoteOnly" />

            <authentication mode="Windows" />

            <authorization>
                  <allow users="*" /> <!-- Allow all users -->
                  <!-- <deny verbs=”POST” users=”?” roles=”” /> -->
            </authorization>

            <sessionState mode="InProc" stateConnectionString="tcpip=127.0.0.1:42424"
                  sqlConnectionString="data source=127.0.0.1;Trusted_Connection=yes"
                  cookieless="false" timeout="20" />
      </system.web>

      <!--
      you can use the <location> tag to modify the configuration settings for a particular page
      or complete directory. You can also use this tag to lock configuration settings
      (allowOverride) in the web.config so that they cannot be overridden by a web.config file
      located below it.
      -->
      <location path=”application1”>
      <system.web>
            <pages enableViewState=”false” />
      </system.web>

      <location path=”secret.aspx” allowOverride=”false” >
      <system.web>
            <authorization>
                  <deny users="?" />
            </authorization>
      </system.web>
</configuration>
```

Init [Page.Init()]

ASP.NET framework uses a pool of application instances to process each request to the server. When the request is made, an application instance is assigned to the request. Immediately after any of these application instances are created, the Init event is raised.

You can use the Init event to initialize any variables or objects that you'll need to use throughout the lifetime of a particular application instance. If you assign values to local variables, the variables retain their values across multiple requests.

Items added to the application state are guaranteed to survive across multiple application instances, whereas variables created in the Init() survive only one application instance.

Session

- When a user first requests a page from ASP.NET web site, the server automatically adds a session cookie (ASP.NET_SessionID) to the users browser. From this point onwards a user session starts.
- A session is have a default lifetime set in the config files (30 minutes).
- A session will have a *sliding expiration policy* i.e., the expiration time extends to the set lifetime from the last page-hit time.
- ASP.NET supports storing session information
    - In-process `(default)`
    - Out-of-process
        - In a state server
        - In a database table
- ASP.NET_SessionID is the name of the session cookie name.
- Session string is 15 bytes and is made of URL allowed characters only
- Session state is a 120 bit session id
- Session can be maintained thru COOKIES or MODIFIED URL
- Session dictionary contains object types, to read data back you need to cast the return values
  `String temp = (String) Session["MyData"]`
- `Session_OnStart` event is raised when a user requests the first page from a web site.
- `Session_OnEnd` signals the end of session and is supported only in InProc session mode. For this event to be fired a session has to exist.
- Any object can be added to a session state, but one requirement is that the object support serialization.
- You can detect the start of a new session by using the `Session.NewSession` property.
- Modifying the Global.asax file automatically drops all items from session state.

Session state is stored at

- In-process storage (default)
    - Stored in memory, this is the default for session state storage
    - *Advantage:*
        - In-process session state is by far the fastest solution. If you are storing only small amounts of volatile data in session state, it is recommended that you use the in-process provider.
    - *Limitations:*
        - When using the in-process session-state mode, session-state data is lost if aspnet_wp.exe or the application domain restarts.
        - If you enable Web garden mode in the `<processModel>` element of the application's Web.config file, do not use in-process session-state mode. Otherwise, random data loss can occur.
- Session state service
    - `<sessionState mode="StateServer" stateConnectionString="tcpip=127.0.0.1:42424" />`
    - *Advantage:*
        - First, it is not running in the same process as ASP.NET, so a crash of ASP.NET will not destroy session information.
        - Second, this allows you to share state information across a Web garden (multiple processors on the same computer) or even across a Web farm (multiple servers running the application).
    - The State Server simply stores session state in memory when in out-of-proc mode. In this mode the worker process talks directly to the State Server
- SQL Server
    - Like the State Service, SQL Server lets you share session state among the processors in a Web garden or the servers in a Web farm.
    - Also get the additional benefit of persistent storage.

- SQL mode, session states are stored in a SQL Server database and the worker process talks directly to SQL. The ASP.NET worker processes are then able to take advantage of this simple storage service by serializing and saving (using .NET serialization services) all objects within a client's Session collection at the end of each Web request

Session and Cache

- When the first value is added to session dictionary an item is inserted into Cache.
- For InProc session [session held in worker process] the physical container for the session state is cache object.
- Out-of-process sessios (State Server and SQL Server session) do not work with Cache object.

Session state collections

- Contents
- StaticObjects

Session is maintained through non-persistent cookie SessionID.

Cookieless Sessions

- To enable cookieless sessions
- ```
  <configuration>
    <system.web>
        <sessionState cookieless="true" />
   </system.web>
  </configuration>
  ```
- http://mysite.com/(nd4vqe2j4nnidkdiwj49kfo2)/myPage.aspx
- You cannot use absolute URLs when linking between pages in hyperlinks and Redirect().
- Cookieless sessions cause a redirection when a session starts or when an absolute URL is invoked
- ASPFilterSessionId header for cookieless session
- click breaks the session for cookieless session instead use ApplyAppPathModifier()
- <a runat="server" href="<% = Response.ApplyAppPathModifier("/test/abc.aspx")%>>click
- Always use ApplyAppPathModifier() when redirecting from HTTP to HTTPS
- You must always use relative paths only.
  ```
  <a href="myPage.aspx">click here</a>
  <a href="myDir/myPage.aspx">click here</a>
  <a href="../myPage.aspx">click here</a>
  <a href="/myPage.aspx">click here</a>
  <a href="http://abc.com/myPage.aspx">click here</a>
  ```

Cookies

- Cookies are little pieces of information that a server stores on a browser.
- A cookie cannot contain more than 4KB of data.
- Cookies are passed back and forth between a browser and server through HTTP headers.
- Cookie can contain only string values.
- By default, when a user first requests a page from an ASP.NET web site, the web site automatically adds a session cookie *_ASP*.NET_SessionID cookie to the users browser to track the user for the remainder of his/her visists.
- Information stored within a cookie can be read only by the Web server that originally sent the cookie, not by other Web servers.
  ```

Properties of a cookie.

- Domain
- Expires
- HasKeys enables you to detect whether a cookie is a cookie dictionary. A cookie dictionary is a single cookie that stores multiple key/value pairs.
- Name
- Path
- Secure
- Value
- Values

There are two types of cookies.

Session Cookies:

- These are temporary cookies and exist only in the browser's memory.
- These cookies do not have an expiry date.
- When the browser is shut down, any session cookies added to the browser are lost.
- Session cookies survivie for as long as the visitor operates the Web browser in a single "session".

Persistent Cookies:

- These cookies have an expiry date.
- These cookies are stored on a visitor's hard drive and are read by the visitor's browser each time the visitor visits the Web site that sent the cookie. They will remain there until the set date has expired or until the visitor has deleted the file.

ViewState

- ViewState is a state service that developers can use to track/maintain UI state of a page.
- Data entered in the form fileds is automatically preserved in a hidden form filed named _VIEWSTATE, which is automatically created in every web forms page. This is an old web-programming trick "roundtripping state" in a hidden form field, but this process is now baked right into the page-process by .NET

```
<input type="hidden" name="_VIEWSTATE" value="YTB3er+mWr4yTg" />
```

- ViewState is preserved only if the form is posted back to itself.
- ***call context:*** ViewState lets application build a call context which is used to manage information across the entire request lifetime, similar to the REQUEST scope of coldfusion) and also retain values across 2 successive requests for the same page
- ViewState is case-sensitive and is base64-encoded.
- It is not encrypted but it can be encrypted by setting EnableViewStatMAC="true" & setting the machineKey validation type to 3DES.
- You can enable or disable the ViewState by using the attribute EnableViewState="true|false".
- All Asp.net controls automatically preserve viewstate.
- Form controls, such as TextBox and RadioButtonList, retain their values between page posts even when viewstate is disabled. The values of these controls do not need to be preserved in the VIEWSTATE because they are actually being submitted to the server on each form post.
- ***StateBag*** is the class behind the ViewState. Viewstate string is deserialized and transformed into an instance of StateBag object. To add a value to the STATEBAG: `ViewState("SomeItem")="xyz";` You can take advantage of the ViewState within your code by adding values to the STATE BAG class. If you add values to this class, they are automatically added to the hidden _VIEWSTATE.
- ***LosFormatter*** class is used to serialize web controls state into string presentation inside hidden field inside the page Form. It is also used on post back to de-serialize the hidden field back into controls. This class is optimized for highly compact ASCII format serialization. This class supports serializing any object graph, but is optimized for those containing strings, arrays, and hashtables.
- Decodes the string at init()
- MAC (Machine Authentication Check)

Cache

Caching is the technique of storing frequently used items in memory so that they can be accessed more quickly. In other words this refers to storing information for later retrieval. Caching is a technique widely used in computing to increase performance by keeping frequently accessed or expensive data in memory.

- Cache class works like an application wide repository.
- Cache object is a global, application-specific repository of data with advanced capabilities such as priority, file dependencies, removal callbacks, and expiration dates.
- Cache object enables you to specify expiration policies and dependencies for cached items.
- Cache object support scavenging. Scavenging tool eliminates low-priority and seldom used cache objects.
- Cache class periodically checks the memory and if the memory reaches a guard level a scavenging mechanism starts and reclaims memory.
- Cache object is created per AppDomain basis
- You can access cache using
  - `Http.Cache  or`
  - `Page.Cache`
- Cache live side by side with Application object
- Cache[“abc”] = “mydata”; stays in cache indefinetly
- Setting a SlidingExpiration > 1 year will throw and error
- Cache Dependency
- By default, all items added to the cache have no expiration date. But Cache.Add() and Cache.Insert() allow you to choose 2 mutually exclusive expiration policies – AbsoluteExpiration -SlidingExpiration
- Cache class is a collection that is instantiated within the HttpRuntime Object
- The cache location is determined by the web form setting.
- Location settings on a user control have no effect.
- If the web form's cache duration is longer than the user control's, both the web form response and the user control response will expire using the web form settings.
- When the memory is running low, ASP.NET might remove data from the cache if the cached data is not used frequently or is unimportant. This operation is called *scavenging*.
- To set expiration policy on a cache use
  - `Cache.Insert()  or`
  - `Cache.Add()`
- To set an absolute expiration policy

```
Cache.Insert(“myItem”, ”Hello!”, null,
             DateTime.Now.AddMinutes(10), Cache.NoSlidingExpiration);
```

- To set a sliding expiration policy

```
Cache.Insert(“myItem”, ”Hello!”, null,
             Cache.NoAbsoluteExpiration, Timespan.FromMinutes(10));
```

Overview

- Output caching
  - Absolute expiration
  - Sliding expiration
- Fragment caching
- Data caching

Output caching

- Output caching enables you to cache the contents of an entire page.
- You can cache a page for an *absolute period of time* or implement a *sliding expiration* policy.
- [Caches entire page] – caches embedded control too.
- Attributes are *VaryByParam, VaryByHeader, VaryByCustom, VaryByControl, Duration, Location*
  - *VaryByParam*
    A string or a semicolon-separated list of strings (GET / POST) used to vary the output cache. When this attribute is set to multiple parameters, the output cache contains a different version of the requested document for each specified parameter. Possible values include none, *, and any valid query string or POST parameter name.

```
<%@ OutputCache Duration="60" VaryByParam="none|*|name|name;age" %>
```

    none - only one version of page cached (only raw GET). Indicates that different cached versions of the page should not be created when the page is requested with different parameters.
    *** - indicates that different cached versions of a page should be created whenever the page is requested with different parameters. N versions of page cached based on query string and/or POST body.
    name - n versions of page cached based on values of parameter 'name' in the query string or post body.
    name;age - n versions of page cached based on values of parameter combinations of 'name' and 'age' in the query string or post body.
  - *VaryByHeader*
    Maintains a separate cache entry for headers (one or a list of headers, eg. User-Agent or User-Agent; Accept-Language) When this attribute is set to multiple headers, the output cache contains a different version of the requested document for each specified header.

```
<% @OutputCache Duration="60" VaryByHeader="Accept-Language" %>
```

  - *VaryByControl*
    For user controls, maintain separate cache entry for properties of a user control
  - *VaryByCustom*
    Enables you to control the sensitivity of the page output caching. Can specify separate cache entries for browser types and version or provide a custom *GetVaryByCustomString*() method in HttpApplication derived class

```
<% @OutputCache Duration="60" VaryByCustom="Browser" %>
```

  - *Location*
    For setting the cache location use the location attribute

```
<% @OutputCache
        Location="Any|Client|Downstream|None|Server|ServerAndClient"%>
```

HttpCachePolicy

- Behind the scenes, the HttpCachePolicy class handles page output caching.
- To set the caching policy programmatically – for more fine grained control
- This class allows you to set output caching for a page programmatically.
- The HttpResponse object can be accessed through the Response property of the Page or HttpContext class.
- HttpCachePolicy represents a programming interface alternative to using the `<% @OutputCache` directive

```
private void Page_Load(object sender, System.EventArgs e)
{ lblGenerateTime.Text =DateTime.Now.ToLongTimeString();
  // to indicate an abosulte date and time when a page expires
  Response.Cache.SetExpires(DateTime.Now.AddSeconds(15));
  // to indicate how the page should be cached
  Response.Cache.SetCacheability(HttpCacheability.Public|Private|Server|NoCache);
  Response.Cache.SetValidUntilExpires(true);
}
```

Fragment caching

- Enables you to cache particular areas of pages [partial page]
- [Caching partial page] - Caches the portion of the page generated by the request. Some times it is not practical to cache the entire page, in such cases we can cache a portion of page
- *Fragment* caching refers to caching part of a page. ASP.NET enables fragment caching by allowing you to specify cache rules for Web forms *user controls*.
- Fragment caching is like output caching, where the output of a user control is cached.
- VaryByControl is supported only for user controls
- `<% @OutputCache %>` directive in .ascx page is used to indicate how the user control is to be cached.
- By default, the output of each user control is cached separately, if you want to share the same cached output across then use the shared attribute.

```
<% @OutputCache Duration="60" VaryByParam="*" Shared="true" %>
```

Limitations:

- If you enable caching for a user control, you can no longer programmatically access the control in its containing page.

```
<mycontrols:PageContents ID="uc1" CategoryId="2" RunAt="Server" />
uc1. CategoryId = 2;
```

- You should not attempt to use data-binding syntax with a cached user control. This will generate an error.

```
<mycontrols:PageContents CategoryId="<%# CategoryId%>" RunAt="Server" />
<mycontrols:PageContents CategoryId="2" RunAt="Server" />  //OK
```

Programmatic partial caching

- [PartialCaching(5)] – alternative to partial caching using VaryByControl
- User controls are cached by applying [PartialCaching (120)] attribute to it.

Data caching

- To cache DataSets and other types of data.
- [Application data caching] - Caches the objects programmatically.
- The caching of arbitrary data is known as *data caching* or *application data caching*. Any arbitary data such as strings, data sets, and so on can be cached. You can cache any object you like in ASP.NET by calling the Add() or Insert() methods of the Cache object. You can also set a sliding expiration time to enable the expiration time to slide whenever the data in the cache is accessed.
- Cache["CustomersDataSet"] = dsCustomers;
- Cache.Add() or Cache.Insert() method
- For data caching asp.net provides a cache object

Cache.Add Vs Cache.Insert

Adding existing cached object to memory by using Cache.Add () method throws an exception

`Cache.Add()`	method returns the cached object on adding
`Cache.Insert()`	method just replaces the old value with new value. does not return anything. has CacheDependency object as second parameter.

Cache File Dependencies

- You can associate the item with a file. If the file changes, the item is automatically removed from the cache.
- To create the file dependency

```
Cache.Insert("myItem","Hello!",new CahceDependency(MapPath("myFile.txt")));
```

Cache Callback()

Used for reloading an item into the cache when it expires.

Data Binding

Data binding refers to the process of dynamically assigning a value to a property of a control at runtime. In other words, making a link between the UI and Data (Data Model). You can bind the properties of a control to expressions, properties, methods, collections, or even properties of another control.

Simple Data Binding - connected a UI element to a single value from the Data Model
- <%# %>
- DataBind()

Complex Data Binding - bind a UI control to an entire collection of Data.

Filtering [select a part of the data from a larger data]
- Filtering with a DataView Object
- Filtering at DataBase Server

Transforming Data
- You transform data using lookups.
- Lookup is a technique for replacing one column data with another in the same table

Template - a set of HTML elements and controls that collectively specify a layout for a control
- Repeater <%# Container.DataItem(“au_lname”) %>
- DataList
- DataGrid [has a default look and feel]

DataBind() - DataBind() is a method of a Page object and of all server controls

Templates enable one to apply complicated formatting to each of the items displayed by a control

Repeater Templates - supports five types of templates

HeaderTemplate	Use this template for elements that need to be render once before your ItemTemplate section. This controls how the header of the repeater control is formatted.
ItemTemplate	Use this template for elements that are rendered once per row of data. [Used for data display] controls the formatting of each item displayed.
AlternatingItemTemplate	Use this template for elements that are rendered every other row of data. This allows you to alternate background colors. Controls how alternate items are formatted.
SeperatorTemplate	Use this template for elements to render between each row, such as line breaks. Displays a separator between each item displyed.
FooterTemplate	Use this template for elements that you want to render once after your ItemTemplate section. controlling how the footer of the repeater control is formatted.

DataList and DataGrid Templates - support two templates in addition to the above five.
- **SelectedItemTemplate** controls how a selected item is formatted.
- **EditItemTemplate** controls how an item selected for editing is formatted.

ADO.NET / Data Manipulation

- ADO.NET implements a disconnected data access model by default.
- Data access in ado.net relies on 2 entities
  - DataProvider
  - DataSet

DataProvider - creates and maintains links to the database. It consists of the following:

Connection object

- Provides connection to the database.
- Represents the actual connection to the database.
- You associate transactions with the connection object.
- Each connection requires approximately 40 kb of memory.
- Connections are pooled only when they are opend with connection strings that exactly match character by character.

Command object

- Executes a command against a datasource.
- Used to execute queries, stored procedure, or return complete tables.
- Contains reference to a db query statement or stored procedure.
- Command object provides fast and efficient way to interact with db

CommandType property

- Text - represents a SQL (for Ad Hoc queries)
- StoredProcedure - represents a stored procedure
- TableDirect - returns all columns + rows of the table mention in the Text
- *Methods*
  - ExecuteReader – for select, returns a resultset by way of DataReader object.
  - ExecuteNonQuery – for insert, update, delete, returns no. of records affected
  - ExecuteScalar – returns first row, first column value
  - ExecuteXmlReader – provides a forward-only, read-only data formatted in XML
- *Parameters*
  - Parameters are values that fill placeholders left in the command text.
  - Unnamed parameters ?
  - Named parameters @xxx

DataReader object

- Is a lightweight object that provides a forward-only, read-only, connected stream resultset in a fast and efficient manner.
- DataReader objects cannot be directly instantiated. There will be only one row in memory at a time, hence, a DataReader provides the lowest overhead in terms of performance, but it requires exclusive use of an open Connection for the lifetime of the DataReader.
- You have to exclusively call *.Close()* of the *Connection* to close the active connection or you can also set the *CommandBehaviour* property to *CloseConnection* to close the connection automatically (eliminating the need to explicitly call *.Close()*)
- `Getxxx()` methods return strongly types data from columns
- Retrieving data thru [index] is faster than retrieving data using [“emp_name”]
- DataReader is similar to a Recordset with CursorType = adOpenForwardOnly and LockType = adLockReadOnly

DataAdapter object

- Populates a disconnected DataSet or DataTable with data and performs updates.
- Uses Fill() to populate a DataSet/DataTable and Update() to transmit changes to database.
- Have four properties that represent db commands.
  SelectCommand, InsertCommand, DeleteCommand, UpdateCommand.
- DataAdapter automatically opens and closes a Connection.
- DataAdapter acts as a pipeline between dataset and a connection.
- SqlDataAdapter is designed to work frm SQL Server 7 onwards.

CommandBehaviour
When you call the ExecuteReader() method of the Command object you can pass an optional ComandBehaviour parameter. CommandBehaviour parameter allows you to gain a greater control over how the ExecuteReader() method retrieves data from a database.

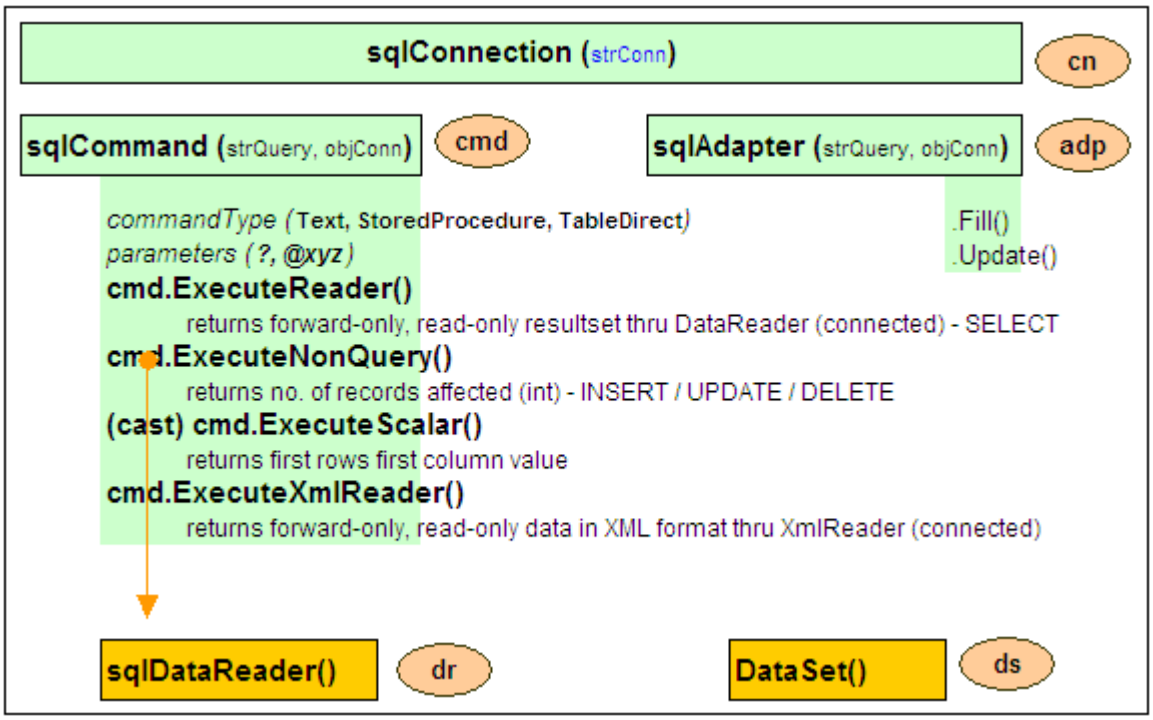
.CloseConnection()
.KeyInfo()
.SchemaOnly()
.SequentialAccess()
.SingleResult()
.SingleRow()

Drag and Dropping

- Drag and Dropping a DATABASE creates SqlConnection
- Drag and Dropping a TABLE, VIEW creates SqlAdapter
- Drag and Dropping a STORED PROCEDURE creates SqlCommand

Connection Strings

- `StrOdbcDSN = "uid=sa; pwd=pwd; dsn=dsnSQLServer;"`
- `strOracleDSN = "user id=sa; password=pwd; data source=dsOracleDB;"`
- `strSqlDSN = "user id=sa; password=pwd; data source=VIRAT; initial catalog=bharat;"`
- `strOleDSN = "uid=sa; pwd=pwd; data source=VIRAT; initial catalog=bharat; provider=SQLOLEDB;"`



Data Source	Identifies the server. [local machine / domain name / IP Addr.]
Initial Catalog	Data base name.
Integrated Security	Set to SSPI to make connection with user's Windows login
User ID	Name of user configured in SQL Server.
Password	Password matching SQL Server User ID.

ex.: string strConn = "Data Source=(local);Initial Catalog=Northwind;User ID=sa;Password=sqlpassword";

CONNECTION STRING

```
string strCon = "Data Source=VIRAT;Initial Catalog=bharat;User Id=sa;Password=pwd;";
```

QUERY STRING

```
string strQuery = "select * from emp;";
```

CONNECTION

```
SqlConnection con = null;
con = new SqlConnection();
con.connectionString = strCon;          // first way
con = new SqlConnection(strCon);        // second way
con.open();
```

COMMAND

```
SqlCommand cmd = null;
cmd = new SqlCommand();
cmd.CommandText = strQuery;
cmd.Connection = con;                   // first way
cmd = new SqlCommand(strQuery,con);     // second way
cmd = con.CreateCommand();
cmd.CommandText = strQuery;             // third way
```

```
SqlDataReader dr =                      cmd.ExecuteReader();
int rslt =                    (int)     cmd.ExecuteScalar();
int rslt =                              cmd.ExecuteNonQuery();
XmlTextReader xtr = (XmlTextReader)     cmd.ExecuteXmlReader();
```

```
con.close();
```

DATAADAPTER

always creates a new connection

```
SqlDataAdapter da = null;
SqlDataAdapter da = new SqlDataAdapter(strQuery, strCon);   // first way
SqlConnection con = new SqlConnection(strCon);
SqlDataAdapter da = new SqlDataAdapter(strQuery,con);       // second way
SqlConnection con = new SqlConnection(strCon);
SqlCommand cmd = con.CreateCommand();
cmd.CommandText = strQuery;
SqlDataAdapter da = new SqlDataAdapter(cmd);                // third way
```

```
Select * from authors;
Select * from Creditors;

do
{     while (myReader.Read())
      {
      }
} while (myReader.NextResult())
```

```
string strCon = "Data Source=VIRAT;Initial Catalog=bharat;User Id=sa;Password=pwd;";
string strQuery = "select * from emp;";

SqlConnection con = new SqlConnection(strCon);
con.Open();

SqlCommand cmd = new SqlCommand(strQuery, con);

SqlDataReader dr = cmd.ExecuteReader();

while (dr.Read())
{   Response.Write(dr[0] +" " +dr[1] +" " +dr[2] +" " +dr[3] +"<br>");
}

con.Close();
```

Using Parameters

```
string strCon = "Data Source=VIRAT;Initial Catalog=bharat;User Id=sa;Password=pwd;";
string strQuery = "select * from emp where empId = @pEmpId;";

SqlConnection con = new SqlConnection(strCon);
con.Open();

SqlCommand cmd = new SqlCommand(strQuery, con);
cmd.Parameters.Add("@pEmpId", 101);
//cmd.Parameters.Add("@pEmpId", SqlDbType.Integer).Value  = 101;

SqlDataReader dr = cmd.ExecuteReader();

while (dr.Read())
{   Response.Write(dr[0] +" " +dr[1] +" " +dr[2] +" " +dr[3] +"<br>");
}

con.Close();
```

```
string strCon = "Data Source=VIRAT;Initial Catalog=bharat;User Id=sa;Password=pwd;";

SqlConnection con = new SqlConnection(strCon);
con.Open();

SqlCommand cmd = new SqlCommand("SP_INSERT_EMP", con);
cmd.CommandType = CommandType.StoredProcedure;

cmd.Parameters.Add(new SqlParameter("pi_empName",SqlDbType.Varchar2,System.Data.ParameterDirection.Input)).Value = tbName.Text;
cmd.Parameters.Add(new SqlParameter("pi_empSal",SqlDbType.Int32,System.Data.ParameterDirection.Input)).Value = tbSal.Text;
cmd.Parameters.Add(new SqlParameter("prv_empId",SqlDbType.Int32,System.Data.ParameterDirection.ReturnValue));
cmd.Parameters.Add(new SqlParameter("po_sqlCode",SqlDbType.Int32,System.Data.ParameterDirection.Output));
cmd.Parameters.Add(new SqlParameter("po_sqlErrm",SqlDbType.Varchar2,System.Data.ParameterDirection.Output));

cmd.ExecuteScalar();

int empId  = (int) cmd.Parameters["po_empId"].Value;

con.Close();
```

Note: A returnValue can return only integer values.

```
string strCon = "Data Source=VIRAT;Initial Catalog=bharat;User Id=sa;Password=pwd;";

SqlConnection con = new SqlConnection(strCon);
con.Open();

SqlCommand cmd = new SqlCommand("SP_INSERT_EMP", con);
cmd.CommandType = CommandType.StoredProcedure;

SqlParameter param1 = new SqlParameter("pi_empName",SqlDbType.Varchar2);
param1.Direction = ParameterDirection.Input;
param1.Value = tbEmpName.Text;

SqlParameter param2 = new SqlParameter("pi_empSal",SqlDbType.Int32);
param2.Direction = ParameterDirection.Input;
param2.Value = tbEmpSal.Text;

SqlParameter param3 = new SqlParameter("po_sqlCode",SqlDbType.Int32);
param3.Direction = ParameterDirection.Output;

SqlParameter param4 = new SqlParameter("pi_sqlErr",SqlDbType.Varchar2);
param4.Direction = ParameterDirection.Output;

SqlParameter param5 = new SqlParameter("prv_empId",SqlDbType.Int32);
param5.Direction = ParameterDirection.Input;

cmd.ExecuteScalar();

int empId  = (int) cmd.Parameters["prv_empId"].Value;

con.Close();
```

T-SQL - ad hoc queries
- Stored procedures

@@IDENTITY
An identity column is a column whose value is assigned by the sql server whenever you add a new row to a table

An insert statement must specify explicit values for any columns that are not identity columns, are not nullable, and do not have a default value.

DiffGram is one of the two XML formats that you can use to render DataSet object contents to XML.

Data Error Handling
- SqlException
- SqlError

`System.Data.SqlClient` contains classes for connecting to aSQL server. This is designed for connection to SQL server 7.0 or higher. Low-level propriety TDS (Tabular Data Stream) protocol is used for optimized performance.

```
using (SqlConnection con = new SqlConnection("data source=(local); initial
                                        catalog=bharat; integrated security=true;"))
{
      SqlCommand cmdEmp = con.CreateCommand();
      cmdEmp.CommandText = "select * from emp";

      SqlCommand cmdDept = con.CreateCommand();
      cmdDept.CommandText = "select * from dept";

      SqlDataAdapter da = new SqlDataAdapter();
      DataSet ds = new DataSet();

      con.Open();

      da.SelectCommand = cmdEmp;
      da.Fill(dataset, "emp");

      da.SelectCommand = cmdDept;
      da.Fill(dataset, "dept");

      con.Close();
}
```

```
SqlCommand cmd = con.CreateCommand();
SqlDataAdapter da = new SqlDataAdapter();
DataSet ds = new DataSet();

con.Open();

cmd.CommandText = "Select * from emp";
da.SelectCommand = cmd;
da.Fill(dataset, "emp");

cmd.CommandText = "Select * from dept";
da.SelectCommand = cmd;
da.Fill(dataset, "dept");

con.Close();
```

```
Optimistic locking locks the record being updated only during the call to Update
Pessimistic locking locks the record as soon as Edit is called and keeps it locked
until the Update call completes or you move to a new record
```

Dataset

- Dataset is a disconnected, in-memory representation of database data [containing tables, keys, constraints, and relationships]
- DataSet objects
  - DataSet - is a self-contained, memory-resident representation of relational data
  - DataTable - represents a single table within a DataSet
  - DataRow - represents a table row
  - DataColumn - represents a table column
  - DataRelation - represents a relationship between 2 columns in different tables.
    - You can also enforce data integrity in the DataSet by using the UniqueConstraint and ForeignKeyConstraint objects.
    You use datarelation to retrieve parent and child rows. GetChildRows, GetParentRow
  - DataView - represents subset of data, can be sorted, and/or filtered
    .sort() and .rowsFilter()
- DataSets are of 2 types
  - Late-Bound Dataset [weakly typed]
    Standard datasets objects are inherently weakly typed or late-bound.
    - Slower
    - late bound
    - table/column names cannot be validated till runtime
    - dt.Rows[0][0] or dtEmp.Rows[0]["empId"]
  - Strongly Typed DataSet
    A strongly typed dataset implements strong typing between each member. A Typed DataSet is actually an instance of the new class that derives from DataSet class and provides strongly typed methods, events, and properties. This means you can access tables and columns by name, instead of using collection-based methods.
    - faster
    - this is early-bound
    - columns become properties of dataset
    - table/column names show up in intellisense lists during dedsign time.
    - can be created by deriving directly from a table or other data bearing source
    - dtEmp.Rows[0].empId
    - One way to create a strongly typed dataset object is to derive it directly from a table.
- The DataSet is similar to a Recordset with CursorLocation = adUseClient, CursorType = adOpenStatic, and LockType = adLockOptimistic
- You can strore many edits in a dataset and write them to the original database in a single operation.

DataSet schemas

- DataSet schema is an XML representation of the metadata that defines the structure of dataset.
- Dataset files are saved as .xsd (XML Schema Definition)
- Using dataset schema (xsd file) you can instantiate a strongly typed dataset object.
- The datset can also persist and reload its contents as XML and its schema as XML Schema definition language (XSD) schema.

Clone() vs Copy()

Clone - Copies the structure of the DataSet, including all DataTable schemas, relations, and constraints. Does not copy any data.

Copy - Copies both the structure and data for this DataSet.

- You can create schema elements by dragging a table, view, a stored procedure, a function, or a set of columns.
- Metadata is the info that describes data.
- Primary Keys / Unique Keys – to create these keys just drag the key tool to the element.
- Relationships
  - One-to-Many relationship – to create, drag & drop the relation tool to the child table. [~DB]
  - Nested relationship – to create, drag & drop the child table to the parent table. [~XML data]

- A Simple Type can be modified by Facet. Facets enable you to specify data restrictions such as min, max, or exact length.
- A schema file can have only 1 primary key.
- DataSet contains the following:
  - Elements [unit of information / standalone XML Entity]
    - Complex element [contains other elements]
  - Element groups [Compositor associates Element Groups]
  - Attributes [attributes further describes an element]
  - Attribute groups
  - Types
  - Facets [describes min/max length of Simple Type]

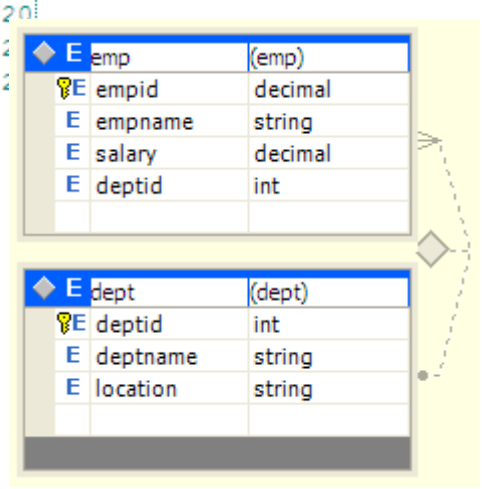
XmlDataDocument

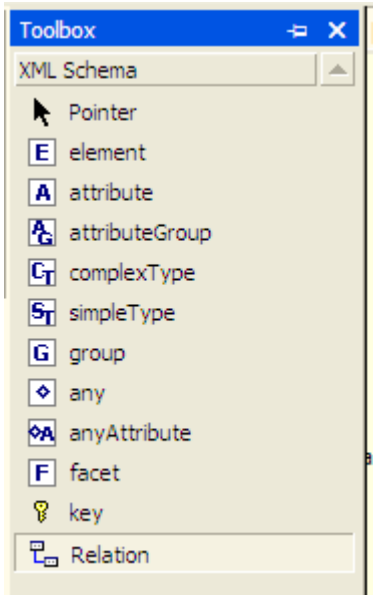
XmlDataDocument class allows you to work directly with in-memory representations of XML and synchronize them with DataSet.

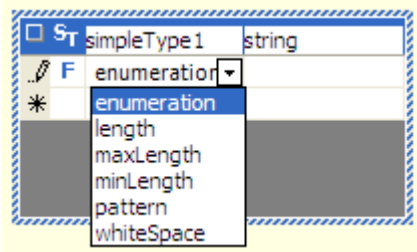
```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema  id="DataSet2" targetNamespace="http://tempuri.org/DataSet2.xsd"
             elementFormDefault="qualified"
             attributeFormDefault="qualified" xmlns="http://tempuri.org/DataSet2.xsd"
             xmlns:mstns=http://tempuri.org/DataSet2.xsd
                  xmlns:xs="http://www.w3.org/2001/XMLSchema"
             xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
      <xs:element name="DataSet2" msdata:IsDataSet="true">
            <xs:complexType>
                  <xs:choice maxOccurs="unbounded">
                        <xs:element name="emp">
                              <xs:complexType>
                                    <xs:sequence>
                                          <xs:element name="empid" type="xs:decimal" />
                                          <xs:element name="empname" type="xs:string"
minOccurs="0" />
                                          <xs:element name="salary" type="xs:decimal"
minOccurs="0" />
                                          <xs:element name="deptid" type="xs:int" minOccurs="0"
/>
                                    </xs:sequence>
                              </xs:complexType>
                        </xs:element>
                        <xs:element name="dept">
                              <xs:complexType>
                                    <xs:sequence>
                                          <xs:element name="deptid" type="xs:int" />
                                          <xs:element name="deptname" type="xs:string"
minOccurs="0" />
                                          <xs:element name="location" type="xs:string"
minOccurs="0" />
                                    </xs:sequence>
                              </xs:complexType>
                        </xs:element>
                  </xs:choice>
            </xs:complexType>
            <xs:unique name="DataSet2Key1" msdata:PrimaryKey="true">
                  <xs:selector xpath=".//mstns:emp" />
                  <xs:field xpath="mstns:empid" />
            </xs:unique>
            <xs:unique name="DataSet2Key2" msdata:PrimaryKey="true">
                  <xs:selector xpath=".//mstns:dept" />
                  <xs:field xpath="mstns:deptid" />
            </xs:unique>
            <xs:keyref name="deptemp" refer="DataSet2Key2" msdata:ConstraintOnly="true">
                  <xs:selector xpath=".//mstns:emp" />
                  <xs:field xpath="mstns:deptid" />
            </xs:keyref>
      </xs:element>
</xs:schema>
```

Dataset1.xsd

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema id="Dataset1" targetNamespace="http://tempuri.org/Dataset1.x
    attributeFormDefault="qualified" xmlns="http://tempuri.org/Dataset1
    xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schem
    <xs:element name="Dataset1" msdata:IsDataSet="true">
        <xs:complexType>
            <xs:choice maxOccurs="unbounded">
                <xs:element name="dept">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="deptid" type="xs:int" />
                            <xs:element name="deptname" type="xs:string
                            <xs:element name="location" type="xs:string
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="emp">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element name="empid" type="xs:decimal"
                            <xs:element name="empname" type="xs:string"
                            <xs:element name="salary" type="xs:decimal"
```







Transact SQL

- T-SQL is Microsoft's implementation of SQL.
- ad hoc queries / stored procedures
- SqlConnection object represents a connection to a database.
- SqlCommand object represents a single query that you can send to the server.
- DataSet object represents the results of one or more queries.
- SqlAdapter object acts as a pipeline between the SqlConnection and DataSet objects.

```
SELECT      field_list          INSERT into TABLE (field_list) VALUES (value_list)
FROM        table_list
WHERE       clause              UPDATE TABLE SET field=value WHERE field1=xyz
GROUP BY    clause
HAVING      clause              DELETE from TABLE WHERE field1=xyz
ORDER BY    field(s)
```

GROUP BY - think this as creating buckets.
SELECT country FROM customers GROUP BY country
- sets up buckets for each unique country
SELECT state, country FROM customers GROUP BY region, country
- sets up buckets for each unique combination of region and country

HAVING clause filters on the results whereas WHERE clause filters on the source input.

STORED PROCEDURES are compiled code stored on the SQL server.
SqlCommand lets us execute stored procedure.

@@IDENTITY column is a column whose value is assigned by SQL server itself whenever a new row is added to the table.

Deployment

xcopy - to depoly an assembly when the assembly is used by only one application and requires no additional setup steps to occur on the target computer

ftp -

windows installer [.msi]
- setup project
- web setup project
- merge module project
- cab project

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

File

There are 2 ways to access files
1. File - represents a disk file
- contains static methods to open, move, copy and delete files,
- performs security check whenever a static method is called which involves additional overhead
- also helps in creating FileStream objects

2. FileInfo - almost everything can of File class actions can be carried out
- does not contain static methods
- need to create and instance of this class before using it
- security check is performed only when the object is created and is not repeated for each method call.

Directory (static) / DirectoryInfo (instance)

Streams - a flow of raw data
Backing Stores - represents a place to put the data

TextReader [abstract]
    StreamReader
    StringReader

    BinaryReader

TextWriter [abstract]
    StreamWriter
    StringWriter

    BinaryWriter

Stream [abstract]
    FileStream
    MemoryStream
    NetworkStream
    CryptoStream
    BufferedStream

FileStream

This class treats file as raw, typeless stream of bytes.
This is the best option if you do not know the internal structure of the file.

StreamReader / StreamWriter
---
best for .txt [TEXT] files

BinaryReader / BinaryWriter

Globalization

Globalization is the process of creating an application that meets the needs of users from multiple cultures.

Localization is accommodating the cultural differences in an application.

Localization is a 3 step process consisting of
- globalization [identifying resources like text, dates, times, currency etc]
- localizability [verifying seperation of resources from code]
- localization [translating resources]

Culture

Cultures are indentified by culture codes and are of 3 types
- neutral culture code [specify general languages such as English, Spanish]
- specific culture code
- invariant culture

en - neutral culture
- does not specify subculture code
- does not provide sufficient information to localize an application

en-GB - specific culture
- provides enough information to localize an application

az-AZ-Cyrl - specific culture with 2 subculture codes

Invariant culture - does not have abbreviation
- has 2 purposes
- to interact with other software such as system services
- to store data in a culture independent format

System.Globalization.CultureInfo

CultureInfo.GetCultures() is a static method

.NET handles localization on a thread-by-thread basis.
- Thread.CurrentThread
- CurrentCulture - dictates formats for dates, currency, numbers, casing rules and string comparision rules.
- CurrentUICulture - for choosing resources for UI

Request.UserLanguages[0] gives the client browser culture info

english	name	age	city	save
french	nom	âge	ville	économiser
spahish	nombre	edad	ciudad	excepto

Resources01.resx - invariant culture resource file
Resources01.es-MX.resx - specific culture resource file
Resource01.es.resx - neutral culture resource file

mirroring - using <html dir=rtl>

string indexing

.NET uses the Request object's UserLanguages property to return a list of the user's language preferences.

To get the user's culture at run time:
1. Get the Request object's UserLanguages property.
2. Use the returned value with the CultureInfo class to create an object representing the user's current culture.

Globalization approaches
1. Detect and Redirect
2. Run-Time adjustments
3. Satellite assemblies

Setting culture in web.config
<globalization requestEncoding="utf-8" responseEncoding="utf-8" culture="ar-SA" />

mirroring
<body dir="rtl">

Threads / run-time / culture
you can set the culture dynamically at run time using the Thread class's CurrentCulture property,

The CurrentCulture property can't be set to a neutral culture because it uses region information to format currencies, among other things.

Encoding
The common language runtime (CLR) uses the UTF-16 character encoding.
ASP.NET uses the UTF-8 character encoding by default when interpreting requests and composing responses.

<globalization requestEncoding="utf-8" responseEncoding="utf-8" fileEncoding="utf-8" />

Help

Tool Tips

A ToolTip is a short, descriptive message that's displayed when the user places the mouse pointer over a control and leaves it there for a couple of seconds.

Only Internet Explorer displays the title attributes as ToolTips. Other browsers do not display ToolTips.

Displaying Help as Web Forms or HTML

```
<a href="#" onclick="window.open('helpTopic1.aspx','helpwin').focus()">
        click here for more help
</a>
```

Displaying HTML Help / Compiled Help Files

```
<a href="#" onclick="window.showHelp('helpTopic1.aspx')">click here for showHelp</a>
```

Configuration

If you change the <processModel> element in the machine.config file, you must restart IIS to have the changes take effect.

<location path="subDir1">
    settings contained in this element will apply only to pages stored in the subDir1 subdirectory of the application
</location>

allowOverride="false" attributes
- more speciifc config. files cannot override this setting

Configuration file is an xml document.
- Machine.config
- Application.config
- Security.config [mscorcfg.msc | caspol.exe]

check site www.hunterstone.com for configuration editor

A Matter Of Context by: dotScott

For passing values between asp.net forms in C#, there are two ways to do this besides the traditional request, application, session objects that were used in ASP. There is the Context.Items collection and the Context.Handler Context.Items is available throughout any of the pages within the given context (like the session object).

The syntax is simple for adding:
Context.Items.Add("userName", name.Text)

then retrieving the value is:
Context.Items("userName")

For the Context.Handler - this only allows passing values between two pages (not more than two- where context.items allows passing values through more than two pages) in the calling page (CallingPage)

```
private void Button1_Click(object sender, EventArgs e){
Server.Transfer("CalledPage.aspx");
}
```

Then in the called page:

```
CallingPage sourcepage = (CallingPage) System.Web.HttpContext.Current.Handler;
Label1.Text = sourcepage.Property1.Text;
```

The Context object is initialized at the start of each request and will last until the end of that request. So Context object provides the ease of getting the data from one page to another.

Context is an object of type System.Web.HttpContext. It is exposed as a property of the ASP.NET Page class. It's also available from user controls and your business objects (more on that later). Here's a partial list of the objects rolled up by HttpContext:

Object Description

Application	A key/value pair collection of values that is accessible by every user of the application. Application is of type System.Web.HttpApplicationState.
ApplicationInstance	The actual running application, which exposes some request processing events. These events are handled in Global.asax, or an HttpHandler or HttpModule.
Cache	The ASP.NET Cache object, which provides programmatic access to the cache. Rob Howard's ASP.NET Caching column provides a good introduction to caching.
Error	The first error (if any) encountered while processing the page. See Rob's Exception to the Rule, Part 1 for more information.
Items	A key-value pair collection that you can use to pass information between all of the components that participate in the processing of a single request. Items are of type System.Collections.IDictionary.
Request	Information about the HTTP request, including browser information, cookies, and values passed in a form or on the query string. Request is of type System.Web.HttpRequest.
Response	Settings and content for creating the HTTP response. Request is of type System.Web.HttpResponse.
Server	Server is a utility class with several useful helper methods, including Server.Execute(), Server.MapPath(), and Server.HtmlEncode(). Server is an object of type System.Web.HttpServerUtility.
Session	A key/value pair collection of values that are accessible by a single user of the application. Application is of type System.Web.HttpSessionState.
Trace	The ASP.NET Trace object, which provides access to tracing functionality. See Rob's tracing article for more information.
User	The security context of the current user, if authenticated. Context.User.Identity is the user's name. User is an object of type System.Security.Principal.IPrincipal.

Passing Values Between Web Forms Pages by Scott Brookhart

- Reponse.Redirect issues an HTTP 304 to the browser and causes the browser to go to that page."Response.Redirect() tells the browser that the page requested has moved to a new location. The browser then issues a request for the second page

  Response.Redirect introduces some additional headaches. First, it prevents good encapsulation of code. Second, you lose access to all of the properties in the Request object. Sure, there are workarounds, but they're difficult.

  Finally, Response.Redirect necessitates a round trip to the client, which, on high-volume sites, causes scalability problems. As you might suspect, Server.Transfer fixes all of these problems. It does this by performing the transfer on the server without requiring a roundtrip to the client.

- Server.Transfer starts executing another page without sending anything to the browser.Server.Transfer() is all server side. Effectively what happens is when ASP gets to Server.Transfer() command, it takes all the code in "somepage.asp" and pastes it onto the bottom of the current page, and processes the new page."

  Client is shown as it is on the requesting page only, but the all the content is of the requested page. Data can be persist across the pages using Context.Item collection, which is one of the best way to transfer data from one page to another keeping the page state alive.

  Client knows the physical location (page name and query string as well). Context.Items loses the persistence when navigate to destination page. In earlier versions of IIS, if we wanted to send a user to a new Web page, the only option we had was Response.Redirect. While this method does accomplish our goal, it has several important drawbacks. The biggest problem is that this method causes each page to be treated as a separate transaction. Besides making it difficult to maintain your transactional integrity.

.NET Distributed Applications

Distributed Applications enable communication between objects that run in different AppDomains and Processes.

- Remoting
- Web Services
- Window Services [background process]
- Serviced Components [COM+]

Remoting

.NET Remoting allows communication between objects running in same / different processes (AppDomains) located on the same / different computers at same / different geographic locations, these process can also be on the same / different OS.

To access objects and call methods on them, you need pointers or references to those objects. There are 2 ways to do this.

- COPY the server object to client
- Pass the server object REFERENCE to the client

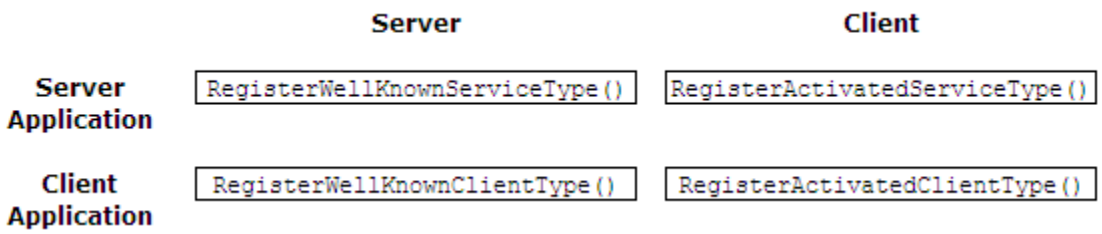
Remoting Architecture
When you create an instance of the server object using NEW keyword, your client receives a reference to the server object i.e., a PROXY object is created and passed to the client. This proxy object contains references to all the methods and properties of the server object.

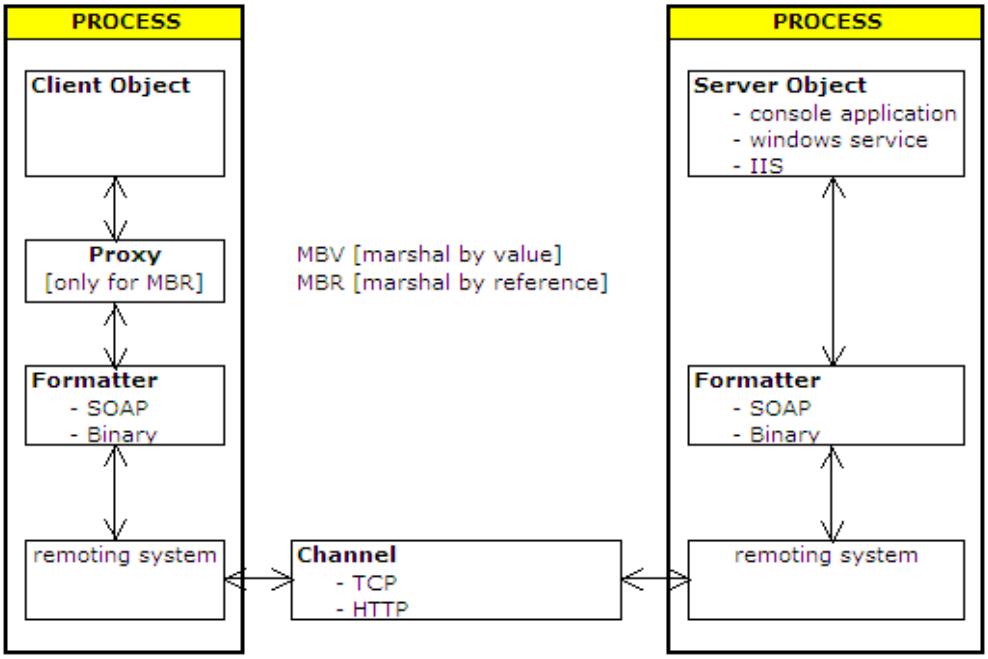
There are two categories of objects,

- Non-remotable objects and
  These objects cannot be either copied or referenced in different AppDomain.
- Remotable objects
  These objects can be copied or referenced in different AppDomain.

Remoting Elements

- Remoting Class / Object
  - Marshal-by-value objects
  - Marshal-byreference objects
    - Activation Modes
      - Server-activated mode
        - SingleCall activation mode
        - Singleton activation mode
      - Client-activated mode (created using the NEW keyword)
- Remoting Host
  - Console Application
  - Window Service
  - IIS
- Remoting Client
- Channels
  - Formatters
    - SOAP
    - BINARY
  - Transporter
    - TCP
    - HTTP





Activation modes
- Server-activated objects [well-known objects]
  - SingleCall activation mode
  - Singleton activation mode
- Client-activated objects

Remotable Object

Remotable objects are the objects that can be marshaled across the application domains. When you develop a remotable object you need to track the creation and initialization of the object, because the objects behoaviour depends on the way the object is created.

There are 2 types of Remotable Objects.

- *Marshal By Value objects*
  - These objects are COPIED and passed by value out of the AppDomain.
  - MBV objects should implement ***ISerializable*** interface or should be marked by SerializableAttribute ***[Serializable]***
  - When a client invokes a method on the MBV Object, a deep copy of the server object is made and this object is serialized and passed by value out of the AppDomain by transferring over the network to the client and restored there as an exact copy of the server-side object.
  - As MBV object is available locally to the client any method calls to the objects do not require a proxy or marshalling. MBV Objects do not provide the privilege of running the remote object in the server environment.
  - MBV objects provide faster access as they reduce network trips
  - Access is fast as marshalling does not take place

```
//Marshal-by-value
[Serializable()]
public class myMBVObject: ISerializable
{
```

- *Marshal By Reference objects*
    - These objects are referenced through a PROXY.
    - MBR objects should extend System.MarshalByRefObject
    - MBR objects are remote objects.
    - When a client invokes a method on the remote object a reference of the remote object is passed to the client.
    - Clients need to use a local PROXY (that holds a reference to the MBR object) to access these objects remotely.
    - .NET creates a proxy object in the client AppDomain that represents the server object and return a reference of that proxy to the caller/client. [**Proxy** is a fake copy of the server object that resides on the client side and behaves as if it was the server. It handles the communication between real server object and the client object (marshaling)]
    - Only MBR can be actuated remotely

```
//Marshal-by-reference
public class myMBRObject: MarshalByRefObject
{
```

Publish

If you need to call an object from a process outside the objects AppDomain, you should ***publish*** an object. When you publish an object, you either copy the object to another appDomain or create a proxy of the object in another appDomain. Whe you publish you cannot export static and private members.

Activation modes

Only MBR objects can be activated remotely, whereas MBV object is transferred to client. MBR objects can be activated in two modes.

- *Server-activated mode (well-known)*
    - Server Activated Objects are created only when the client makes the first call to the remote method. In other words, when the client asked for the creation of the remote object only the local proxy is created onthe client, the actual remote object on the server is instantiated on the first method call.
    - For SAOs the proxy object (on the client) and the actual object (on the server) are created at different times, for this reason ***only default constructors*** can be used for SAOs.
    - SAO's are created on the server only when you call a method in the server class. Objects are not created when you use the NEW keyword to create an instance of the server class.
    - SAO's are also called as Well-known objects
    - The server directly controls SAO's lifetime.
    - There are 2 types of activation modes:
        - *SingleCall* activation mode
            - An object is created and destroyed for each client request.
            - Ideally suited for load-balanced environments. [Maximum scalability]
            - Object cannot maintain state.
            - Ex. Display tracking info, retrieving inventory level for an item
        - *Singleton* activation mode
            - Can have only one instance of the object regardless of the number of clients.
            - These objects also have a default lifetime.
            - Object is created once on the server and is shared by all clients.
            - State can be maintained.
            - Ex. Chat application
- *Client-activated mode* (activated)
    - These are created on the server immediately upon the client's request. An instance of a CAO is cretated every time the client instiantiates one, either by the use of new or Activator. GetObject(...)
    - CAO's are created on the server when you create an instance using NEW keyword
    - The client appDomain defines the lifetimes of CAO's.
    - Any one of the constructors can be used to create CAO's.
    - Any overloaded onstructors can be used to create CAO's.

- When a client calls a server method, an activation request message to create an instance of the server class is sent to the remote server. The server now returns **ObjRef** object to the client application. Client now uses this ObjRef object to create a proxy [CAO] of the object on the client side. An instance of the CAO serves only the client that was responsible for its creation, and the CAO does not get discarded with each request. CAO can maintain state with each client that it is serving, but unlike SAO's, different CAO's cannot share a common state.
- Ex. Shopping carts, Purchase orders

Lifetime Lease

A lifetime lease is the period of time that a particular object shall be active in memory [duration for which the object resides in memory].

- Default lifetime lease is 5 minutes.
- RenewOnCallTime (default 2 minutes) increases the CurrentLeaseTime lease per every call.

Each AppDomain contains a LEASE MANAGER that administers the leases in its domain. The lease manager periodically reviews the leases for expiration. It the lease expires (CurrentLeaseTime reaches 0), the lease manager goes through a list of SPONSORS for that object and asks if they want to renew the lease. If none of the sponsors renew then this object is deleted.

SPONSORS are the objects responsible for dynamically renewing an objects lease. Sponsors are registered with the LEASE MANAGER by calling the ILease.Register()

- There are two ways to renew a lease:
  - A client application calls the ILease.Renew(). Ex. IL.Renew(TimeSpan.FromSeconds(60))
  - A sponsor renews a lease.

- Both Singleton SAOs and CAOs use lifetime leases to determine their lifespan.
- InitializeLifetimeService() of MarshalByRefObject should be overridden for custom lifetime lease

Channels

- Channels represents the objects that remote objects use to communicate with each other.
- A channel must exist before an object can be transferred.
- Channels transport messages across remoting boundaries such as appDomains, process and computers.
- To enable remote object to send messages, you need to register a ClientChannel (TcpClientChannel / HttpClientChannel) with the remoting system. Similarly, to enable remote object to receive messages, you need to register a ServerChannel (TcpServerChannel / HttpServerChannel) with the remoting infrastructure. You use ChannelServices.RegisterChannel() to register a remote object.
- ChannelServices class provides static methods that enable you to register channels, resolve URLs, and discover remote objects using the object URLs.
- The messages carried by channels pass through a chain of **channel sinks**.
  - A channel sink performs tasks such as formatting, transporting and stack building.
  - Within a channel sink chain, you can perform tasks, such as logging, applying filters, encrypting the message, and imposing security restrictions.
  - Message sink

    A message sink is created at the same time when proxy is created. It allows a client to establish a connection with the channel registered by the remote object and forward the messages to the channel.
    - Formatter Sink [the first in the sink chain must be a formatter sink]
      - formatter is an object that is responsible for encoding and serializing data into messages on one end, and deserializing and decoding messages into data on the other end.
      - formatter's code/decode, serialize/deserialise (marshalling) messages.
      - formats the messages into either SOAP or BINARY formats.
    - Transport Sink [the last in the sink chain should always be transport sink]
      - transports messages using HTTP or TCP protocols

- this is the combination of the following technologies that perform low-level tasks, such as a) opening a network connection b) formatting messages c) streaming messages and d) sending the message to the client.

- HttpChannel uses HTTP protocol for transport and by default uses SOAP formatter.
- TcpChannel uses TCP protocol for transpor uses BINARY formatter as it default formatter.
- *Marshalling* - When a client calls a remote object the .NET creates a message that contains parameters and other call related info. This way of bundling the info into a message is called marshalling. Marshalling manages the different representations of data across different execution environments. It performs the necessary conversions in data formats between managed and unmanaged code. Serializing/Deserilizing the message is also known as marshalling.
- There is no default constructor for the TcpServerChannel class.
- HttpChannels create 2 connections by default to connect to a given server. TcpChannels create as many connections as the number of client threads making the requests.

```
using System;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;
using System.Runtime.Remoting.Channels.Http;

HttpChannel hc = new HttpChannel(50001);        TcpChannel tc = new TcpChannel(50002);
ChannelServices.RegisterChannel(hc);            ChannelServices.RegisterChannel(tc);
```

.NET Remoting vs Web services

Use remoting for more efficient exchange of information when you control both ends of the application. Use Web services for open-protocol-based information exchange when you are just a client or a server with the other end belonging to someone else.

- .NET Remoting – when both end points (client + server) of a distributed application are in our control
- ASP.NET webservices – when one end point (server) is in out control

Configuration

- programmatic configuration
  - code recompilation for any new change
- declarative configuration
  allows storing the remoting settings in an XML-based configuration file. Settings can be configured at two levels.
  - Machine-Level
    C:\WINDOWS\Microsoft.NET\Framework\v1.1.4322\CONFIG\machine.config
  - Application-Level
    web.config
    serverConsoleApplication.exe.config
    [VS.NET copies App.config to class.exe.config]

Interface

For programmatic and declarative configuration, you have to copy the DLL to the client project, which is not advisable. Advisable solution is to create an *interface* of the remotable class and share it with the client project.

SOAPSUDs Tool is used to automatically create the interface for a remtable class/object. This tool works only with HTTP channel.

Asynchronous Remoting

Asynchronous remoting is implemented with the help of delegates types.
2 ways to control the async process

- using WaitOne() / WaitAll() with the IasyncResult.IsCompleted
- using CallBack function

Remoting task steps

1. Identify the AppDomain that will host the service.
2. Identify the Activation mode. [Server-Activated mode or Client-Activated mode]
3. Create a channel and a port.
4. Identify how the client obtains the metadata.
5. Configure the system using configuration file.
6. Finally publish the service so that it is accessible from outside the AppDomain.

- **remoting components**

```
using System;

namespace component
{     //Marshal-By-Value Remotable Object
        public class message_mbv
      {     public message_mbv()
            {     Console.WriteLine("message constructor is executed MBV");
            }

            public String sayHello()
            {     Console.WriteLine("sayHello() is executed MBV");
                  return "hello, welcome to remoting world MBV";
            }

            public String sayBye()
            {     Console.WriteLine("sayBye() is executed MBV");
                  return "good bye MBV";
            }
      }
}


using System;

namespace component
{     //Marshal-By-Reference Remotable Object
      public class message_mbr : MarshalByRefObject
      {     public message_mbr()
            {     Console.WriteLine("message constructor is executed MBR");
            }

            public String sayHello()
            {     Console.WriteLine("sayHello() is executed MBR");
                  return "hello, welcome to remoting world MBR";
            }

            public String sayBye()
            {     Console.WriteLine("sayBye() is executed MBR");
                  return "good bye MBR";
            }
      }
}
```

remoting server / listener

```
using System;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;
using System.Runtime.Remoting.Channels.Http;

namespace serverConsoleApplication1
{     class Class1
      {     [STAThread]
            static void Main(string[] args)
            {     //channel/////////////////////////////////////////////////
                        //tcp channel - register a tcp channel
                        TcpServerChannel tsc = new TcpServerChannel(50001);
                        ChannelServices.RegisterChannel(tsc);

                        //http channel - register a http channel
                        //HttpServerChannel hsc = new HttpServerChannel(50002);
                        //ChannelServices.RegisterChannel(hsc);
                  //...channel/////////////////////////////////////////////////
```

```
                //register a remote object (component) with the remoting framework
                //activation mode//////////////////////////////////////////
                      //CAO - Client Activated Object
      //RemotingConfiguration.RegisterActivatedServiceType
                                  (typeof(component.message_mbv));

                      //SAO - Server Activated Object
                      //RemotingConfiguration.RegisterWellKnownServiceType
                                  (typeof(component.message_mbr),"uri_sao_sc",
                                  WellKnownObjectMode.SingleCall);

                            RemotingConfiguration.RegisterWellKnownServiceType
                                  (typeof(component.message_mbr), "uri_sao_s",
                                  WellKnownObjectMode.Singleton);
                //...activated mode//////////////////////////////////////

                Console.WriteLine("server - started");
                Console.WriteLine("press <ENTER> to terminate server");
                Console.ReadLine();
          }
      }
}
```

remoting client

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;
using System.Runtime.Remoting.Channels.Http;
using component;

namespace clientConsoleApplication1
{     public class Form1 : System.Windows.Forms.Form
      {     private System.Windows.Forms.Button btnSayBye;
            private System.Windows.Forms.Button btnSayHello;
            private System.Windows.Forms.Label lblMessage;
            public component.message_mbr ro_mbr;
            public component.message_mbv ro_mbv;

            //…….
            private void btnSayHello_Click(object sender, System.EventArgs e)
            {     this.lblMessage.Text = ro_mbr.sayHello();
                  //this.lblMessage.Text = ro_mbv.sayHello();
            }

            private void btnSayBye_Click(object sender, System.EventArgs e)
            {     this.lblMessage.Text = ro_mbr.sayBye();
                  //this.lblMessage.Text = ro_mbv.sayBye();
            }

            private void Form1_Load(object sender, System.EventArgs e)
            {     TcpClientChannel tcc = new TcpClientChannel();
                  ChannelServices.RegisterChannel(tcc);

                  //HttpClientChannel hcc = new HttpClientChannel();
                  //ChannelServices.RegisterChannel(hcc);

                  //tcp
                  //CAO
                      //RemotingConfiguration.RegisterActivatedClientType
                            (typeof(component.message_mbv),"http://localhost:50001");
```

```
                    //SAO
                    //RemotingConfiguration.RegisterWellKnownClientType
                           (typeof(component.message_mbr),
                           "tcp://localhost:50001/uri_sao_sc");

                    RemotingConfiguration.RegisterWellKnownClientType
                           (typeof(component.message_mbr),
                           "tcp://localhost:50001/uri_sao_s");

              //http
              //CAO
              //RemotingConfiguration.RegisterActivatedClientType
                           (typeof(component.message_mbv),"http://localhost:50002");
              //SAO
                    //RemotingConfiguration.RegisterWellKnownClientType
                           (typeof(component.message_mbr),
                           "http://localhost:50002/uri_sao_sc");

                    //RemotingConfiguration.RegisterWellKnownClientType
                           (typeof(component.message_mbr),
                           "http://localhost:50002/uri_sao_s");

              ro_mbr = new component.message_mbr();
              //ro_mbv = new component.message_mbv();
        }
    }
}
```

App.config → serverConsoleApplication2.exe.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <system.runtime.remoting>
        <application>
            <service>
                <!-- SAO -->
                <!-— type=”QUALIFIED CLASS NAME, ASSEMBLY” -->
                <!-- <wellknown mode="Singleton"
                                type="component.message_mbr, component"
                                objectUri="uri_sao_s" /> -->
                <wellknown  mode="SingleCall"
                                type="component.message_mbr, component"
                                objectUri="uri_sao_sc" />

                <!-- CAO -->
                <!-- <activated type="component.message_mbr, component" /> -->
            </service>
            <channels>
                <!--  tcp         - sender / receiver
                      tcp server  - sender
                -->
                <!-- <channel ref="tcp server" port="50001" /> -->
                <channel ref="http" port="50002" />
            </channels>
        </application>
    </system.runtime.remoting>
</configuration>

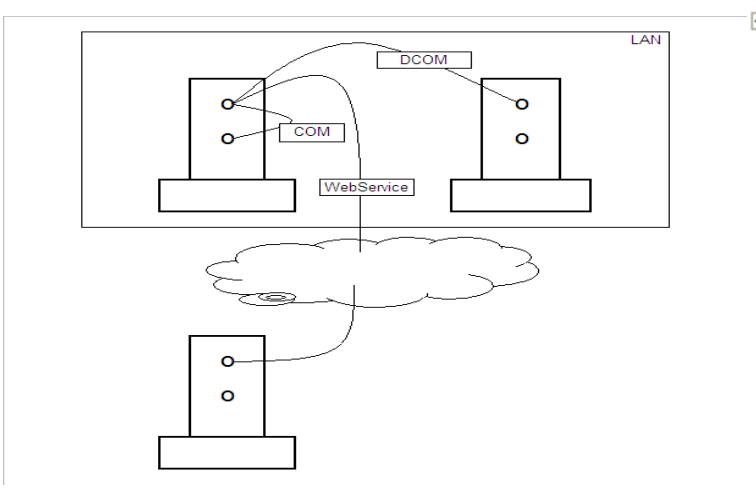
//load remote configuration
RemotingConfiguration.Configure("serverConsoleApplication2.exe.config");
```

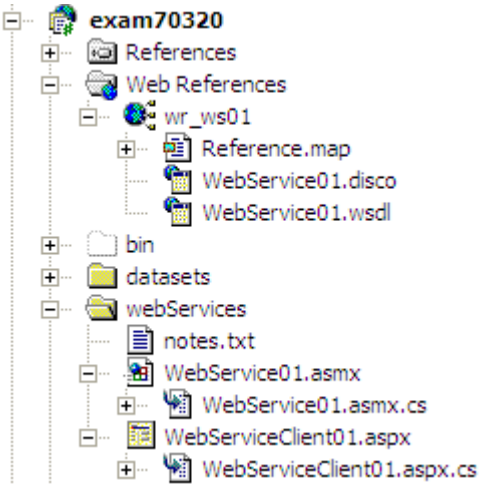
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
      <system.runtime.remoting>
            <application>
                  <!-- SAO -->
                  <!--
                  <client>
                        <wellknown  type="component.message_mbr, component"
                                          url="tcp://localhost:50001/uri_sao_s" />
                        <wellknown  type="component.message_mbr, component"
                                          url="tcp://localhost:50001/uri_sao_sc" />

                        <wellknown  type="component.message_mbr, component"
                                          url="http://localhost:50002/uri_sao_s" />
                        <wellknown  type="component.message_mbr, component"
                                          url="http://localhost:50002/uri_sao_sc" />
                  </client>
                  -->

                  <!-- CAO -->
                  <client url="http://localhost:50001">
                        <activated type="component.message_mbv, component" />
                  </client>
            </application>
      </system.runtime.remoting>
</configuration>

//load remote configuration
RemotingConfiguration.Configure("clientConsoleApplication2.exe.config");
```





Web services enable you to remotely access the properties and methods of classes across the network. A WebService page must have a <%@ WebService %> directive and it should have an **.asmx** extension.

WebService01.asmx
```
<%@ WebService  Language="c#" Codebehind="WebService01.asmx.cs"
                Class="exam70315.webServices.WebService01" %>
```

WebService01.asmx.cs
```
public class WebService01 : System.Web.Services.WebService
{
      [WebMethod]
      public string HelloWorld()
      {
            //test URL: http://localhost/exam70315/webServices/WebService01.asmx
            return "Hello World from WebService01";
      }
}
```

note: on compilation WebService01.DLL is created under the \bin directory

[WebMethod] attribute exposes the method to the world for access thru a web service. Its attributes are:

```
[WebMethod      BufferResponse      true|false      // when false always buffers in16KB chuncks
                CacheDuration       60
                Description
                EnableSession       false|true [default is false for performance reasons]
                MessageName         to create an alias for the method name
                TransactionOption   TransactionOption.None|Required| . . .
```

```
      Example: [WebMethod (CacheDuration=60, EnableSession="true",
                        TransactionOption="TransactionOption.Required")]
```

http://localhost/exam70315/webServices/WebService01.asmx when you access the web service directly from a web browser as shown above, the web service help page is displayed, which lists all the properties and methods of the web service. This is possible because of \WINNT\Microsoft.NET\Framework\[version]\CONFIG\DefaultWsdlHelpGenerator

Hosting the WebServie

To host the above webservice compile it and

1. Copy the .asmx to the destination directory and
2. Copy the .DLL to the \bin directory.

Consuming a WebService [using Web Reference]

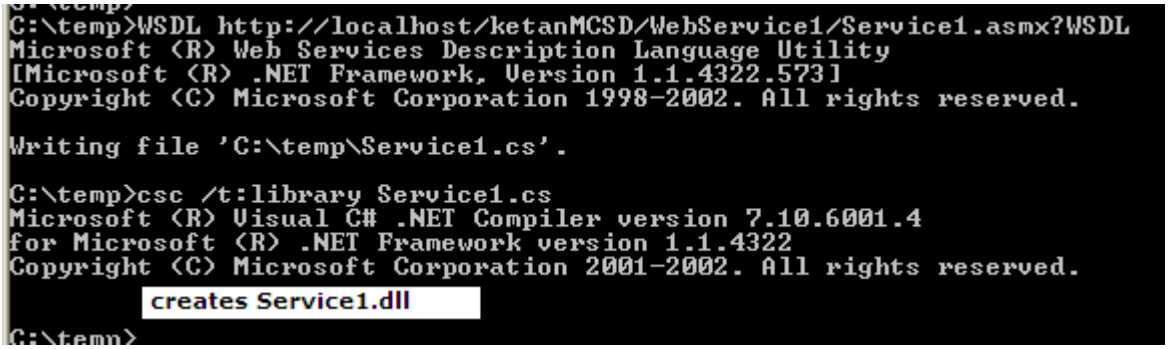
- By adding the web reference, .NET will connect to the web service, gets and parses the WSDL, and generate the proxy class, which you can use to program with.
- When a web reference is set software handles the details of discovery automatically for you.
- A web reference is of 2 types:
  - *Static URL:* This is the default setting. In this case the proxy class sets the URL property using a hard-coded URL when you create an instance of the class.
  - *Dynamic URL:* In this case the application obtains the URL at run time from the <appSettings> element of your application's configuration file (web.config).
- When you add a Web Reference the following files are are automatically created in the client application
  - .wsdl file
  - .disco file
  - .map

Proxy Class

- Proxy class is a local representation of the properties and methods of the web service.
- Proxy class marshals arguments between a webservice and client application.
- To use a proxy class in a client application you must use the fully qualified name of the proxy.
- You can access the web service from within an application by creation a web service proxy class.
- The proxy class contains methods that can make either synchronous or asynchronous calls to web services. Synchronous methods are represented by the name of the actual method call, and asynchronous methods are represented by the name of the method call preceded by *Begin* and *End*.

Manually creating a proxy class

- You can manually created a WSDL file like show below:
  - Using Discovery Tool
    ```
    C:\> disco http://live.capescience.com/wsdl/AirportWeather.wsdl
    Creates 2 files  AirportWeather.wsdl, Results.discomap
    ```
    A .wsdl file is am xml file that describes all the methods and properties of a web service.
  - Using Web Services Description Language Tool
    ```
    C:\> wsdl /l:CS /out:AirportWeatherProxy.cs AirportWeather.wsdl
    Creates a proxy file called AirportWeatherProxy.cs
    ```
- To generate a proxy class
  - First you must generate the source file for the proxy class using WSDL.EXE tool as given:
    `C:\> WSDL http://localhost/exam70315/webServices/WebService01.asmx?WSDL`
    [This creates a source file ***WebService01.cs*** in the current directory; this source file is the proxy class]
    [WSDL is not a physical file, its data is generated when a client requests it by appending ?WSDL to the ASMX page]
    NOTE: WebServices01.asms.cs is the code behind file whereas WebServices01.cs is the proxy class
  - Next, you need to compile the source file (proxy class) as given:
    `C:\> CSC /t:library WebService01.cs`
    [The will compile WebServices01.cs proxy class to WebService.DLL]



Serialization

- The process by which the objects and values are converted into a format suitable for transportation is called serialization.
- It is the process by which objects or values are converted into a format that can be persisted or transported.
- There are 2 types of serialization
  - Binary serialization
    - Converts the state of the objects by writing public & private (fields and properties), class name and assembly into a stream of bytes.
  - XML serialization
    - Converts public (fields and properties), parameters, and return values of methods into an XML stream.

Session and Proxy class

- The web service will interpret each call by the proxy class as the start of a new session. Since, by default, the session cookie won't be passed by the proxy class, session state won't work. If you want the web service to participate in a session you have to explicitly pass the session cookie to the proxy with each request.

Discovery documents

There are three types of discovery files and all of them help clients locate web service files.

- *.disco files,*
  - DISCO (Discovery of Web Services) is a Microsoft-specific technology used to locate web services in a particular directory on a server. The .NET framework generates this file when a client requests it (by appending ?DISCO to the ASMX's URL). A .disco file contains markup that specifies references to a web service's WSDL file and other DISCO documents.
- *.vsdisco files*
  - Dynamic Discovery is the process of searching the web services in the same folder or its subfolder where .vsdisco or .disco file resides. .vsdisco file is placed at the root of a server.
  - By default, dynamic discovery is disabled. .vsdisco files generate more overhead than .disco files.
  - If dynamic discovery is enabled all XML web services and discovery documents existing on the web server beneath the requested URL are discoverable, this is a security threat.
- *.map files*
- WSDL data and .DISCO information for an ASP.NET web service are not physical files. They are generated by appending ?WSDL and ?DISCO query strings to the .ASMX pages.

- In ASP.NET 1.1, by default the web service can only be accessed using SOAP.
- A web site and a web service can share the same application state.

Invoking an XML web service with SOAP

The SOAP (Simple Object Access Protocol) enables you to transmit more complex (DataSets, custom classes, and binary files) types of messages across a network.

The SOAP request:

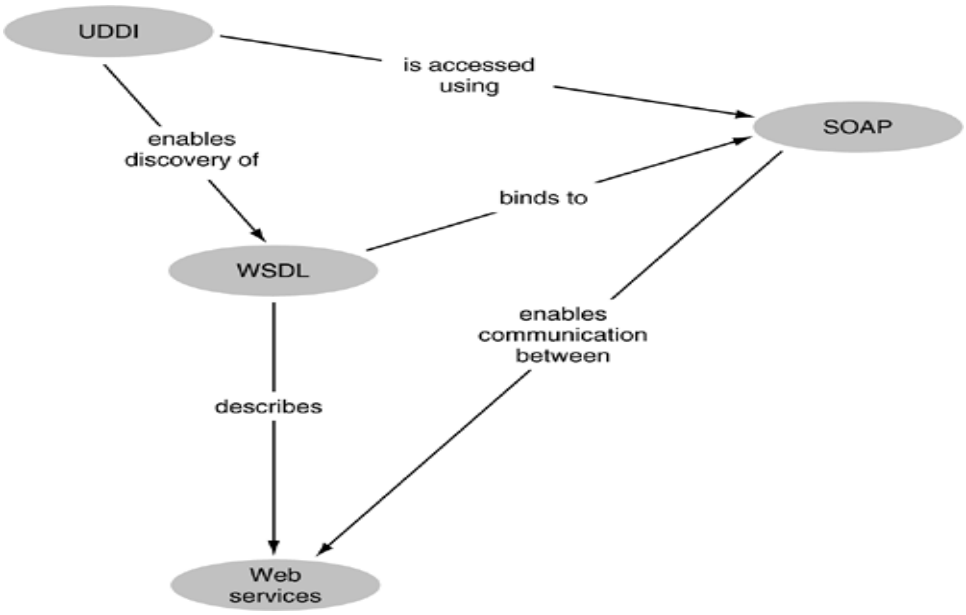
```
POST /MyFirstWS/Service1.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
SOAPAction: "http://tempuri.org/add"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope namespaces>
  <soap:Body>
    <add xmlns="http://tempuri.org/">
      <x>int</x>
      <y>int</y>
    </add>
  </soap:Body>
</soap:Envelope>
```

The SOAP response:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope namespaces>
  <soap:Body>
    <addResponse xmlns="http://tempuri.org/">
      <addResult>int</addResult>
    </addResponse>
  </soap:Body>
</soap:Envelope>
```


UDDI
is accessed using
SOAP
enables discovery of
binds to
WSDL
enables communication between
describes
Web services

DISCO [Discovery of Web Services | directory, inspection, description]

- is the process that clients use to locate documents that describe WS using WSDL.
- to find a web service and
- to discover a web service details.
- Encapsulates oo messages between web service servers and clients

SOAP [Simple Object Access Protocol]

- SOAP protocol is used for exchanging STRUCTURED and TYPED info between the application and internet.
- a way to translate Objects [methods + properties] into XML
- a way to encapsulate method calls as XML sent via HTTP
- tells how to package the data for transport

UDDI [Universal Description, Discovery and Integration]

- Directories provide a central place to store published information about WS.
- UDDI defines the guidelines for the publishing
- is like a phone book [combination of white and yellow pages].
- This protocol enables you to find web services by connecting to a directory
- Is the process of finding services by referring to a central directory
- UDDI Registries come in 2 forms PUBLIC and PRIVATE

WSDL [Web Services Description Language],

- provides info that enables you to know which operations to perform on a WS.
- Specifies the SOAP message schemas.
- This document mentions the SOAP message schemas [methods info]
- defines the public interface of a web service
- WSDL lets you retrieve information on the classes and methods that are supported by a particular web service
- You can create PROXY CLASS using wsdl.exe
- is a standard by which a web server tells it clients
  - what messages it expects and
  - which results it will return

- When you create the Web reference, for example, Visual Studio .NET reads the appropriate WSDL file to determine which classes and methods are available from the remote server. When you call a method on an object from that server, the .NET infrastructure translates your call and the results into SOAP messages and transmits them without any intervention on your part.
- Discovery is the process of finding the web services and determining their interfaces.
- Web Services SERVERS and CLIENTS communicate through XML messages transmitted over HTTP.
- Static discovery is used if the location of the discovery document [.disco] is known.
- Dynamic discovery is used when no static discovery file exists on the web service provider. The discovery information must be dynamically generated when the discovery request id made.
- By default, WSDL.exe will generate proxy classes that use SOAP protocol.
- **.vsdisco** is the dynamic discovery file. Dynamic discovery is used when no static disvoery file exists. The discovery information should be dynamically generated when the discovery request is made.
- Dynamic discovery looks for object interface methods that have the **WebMethod** attribute to know which method to publish.
- **.disco** static discovery is used when the location of the discovery document is known
- for optimal performace of web services set the <@ Page EnableSessionState="false" />, set session to false
- SoapSus.exe tool compiles clients that communicate with web service thru remoting.
- <definitions> is the root element of a wsdl file
- A Windows **Principal** is an object that holds the credentials (referred as Identity) of the current authenticated user.
- A Generic Principal is used to represent a user or role that is not part of win2000 or winNT
- Non-windows principals (Generic Principals) are used when web services use custom authentication protocol rather than windows-based authentication mechanisms.

- Forms and Passport authetication are not recommended for use with web services because they present a login dialog screen to which the clients are unable to respond.
- WSDL.exe will generate proxy classes that use the SOAP protocol
- The language used in an XML Schema is XSD (XML Schema Definition)
- SOAP is used with HTTP to invoke methods exposed by Web Services.
- ?DISCO parameter causes the web service to automatically generate the discovery document
- to access a method in web service *http://servername/apppath/webservicename.asmx/methodname?parameter=value*

Securing WebService

Approach 1: Authenticate user in each method call by passing the username and password. This will be cumbersome if the WebService has many methods.

Approach 2: Create a *Login()* method to accept username and password and return a session key that is valid for 30 minutes. This session key is passed in the SOAP neader to every other method call to authenticate the user.

```
[WebMethod]
[SoapHeader("AuthenticationHeader")]
public boolean MyWebMethod()
{
}
```

```
<soap:Header xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
     <AuthData xmlns="http://www.contoso.com">
          <AuthenticationHeader>abc</AuthenticationHeader>
     </AuthData >
</soap:Header>

// the above soap header is created using the following code
public class AuthData : SoapHeader
{   public string AuthenticationHeader;
}
```

WebService Behaviour

Behaviour is a component that you can associate with an element within an HTML page that extens the elements functionality. WebService Behaviour is used to directly access an XML WebService through IE.

HTML Pattern Machine

You can use HTML pattern matching to read information from an application even if the application does not directly support web services.

Windows Services

- Window services run as background processes. These applications do not need and do not have Graphical User Interfaces.
- You can use windows services to perform tasks such as monitoring the usage of a database, etc.
- You use the following methods of `ServiceProcess` class to create a windows service.
    - ServiceBase [provides OnStart, OnStop, OnPause, OnContinue, OnShutDown]
    - ServiceProcessInstaller [knows how to install your service in SCM]
    - ServiceInstaller [sets up the registry info about your service]
    - ServiceController
- Windows Service is an .exe. A window service is installed in the registry as an executable object.
- A window service architecture consists of 3 components
    - Service Application
        - WSA - Windows Service Application is the application that provides the desired functionality.
    - Service Controller Application
        - SCA enables you to control the behaviour of a service.
    - Service control Manager
        - SCM is a utility that is used to control the services that are installed on a computer. To start, stop, pause, and continue a windows service.
- You use the ServiceInstaller and ServiceProcessInstaller classes to install Windows Services
- SCM (Service Control Manager) manages all the winodws services.
- You use InstallUtil.exe tool to install / uninstall a windows service.

- Window Service States
  `Running, Paused, Stopped, Pending.`
- Window Service Events
  `Start, Pause, Stop, Continue.`

- ***Tasks to create a window service***
    1. Create a blank window service application (wsa)
    2. Add functionality by overriding the *OnStart()* and *OnStop()*
    3. In addition, to change properties override *OnPause()* and *OnContinue()*
       `[Properties like AutoLog, CanStop, CanShutDown, CanPauseAndContinue, CanHandlePowerEvent]`
    4. Write code in the wsa to handle various events
    5. Add installers to your wsa
    6. Install your wsa by using the installation tools [installutil.exe]

- You can view all the registered service applications in the Windows Registry under *HKEY_LOCAL_MACHINE* \ *SYSTEM* \ *CurrentControlSet* \ *Services*
- A window service execute in its own process space until a user stops it, or the computer is shut down.
- Window services are categorized based on the number of services that run in a process space.
    - Win32OwnProcess services - 1 service per process
    - Win32ShareProcess services - n service per process

Installing a windows service

- Installers install and register your service applications on a computer. In addition installers install and configure the resources, such as performance counters and event logs.
- You also use the predefined installers to install and configure the custom event logs, performance counters, and message queues. The following predefined installer classes are provided by .NET
    - `System.Diagnostics.EventLogInstaller`
    - `System.Diagnostics.PerformanceCounterInstaller`
    - `System.ServiceProcess.ServiceInstaller / System.ServiceProcess.ServiceProcessInstaller`
    - `System.Messaging.MessageQueueInstaller`
- Installutil <.exe filename> installs service application
- Installutil /u <.exe filename> uninstalls service application

Steps to attach a debugger to you windows service application

(start the Windows Service, then attach a debugger to the process)

1. Start your windows service application using the SCM
2. Choose Processes from the Debug menu
3. Select Show system Processes
4. Select the process for your service application and click attach. “Attach To Process” dialog box apprears.
5. Select CLR, click OK to specify a debugger, and close the Attach To Process dialog box.

- To debug a Windows Service, you need to install and start the service, and then attach a debugger to the process of the service

- There are 3 + 1 types of event logs available for logging the events
    - System log
    - Security log
    - Application log
    - Custom event log

Creating Event Log Code

```
private void CreateEventLog()
{    EventLog el = new EventLog();

     If (!EventLog.SourceExists(“Test Service”, “Test Log”))
     {    EventLog.CreateEventSource(“Test Service”, “Test Log”);
     }

     el.Source = “Test Service”;
     el.WriteEntry(“my message”, EventLogEntryType.Information);
}
```

Serviced Components

- Services Components are used to access COM+ services, such as automatic transaction management, object pooling, and just-in-time (JIT) activation.
- COM+ (Component Object Model) provides the infrastructure to create and deploy distributed multitier applications.
- Windows DNA (Windows Distributed internet Application) enables implementation of a 3-tier application architecture, which allows you to maintain business logic and data access code in separate components called business objects.
- Services of COM
  - Transactions
    - Atomicity
    - Consistency
    - Isolation
    - Durability
  - Queued Components
  - Security
    - Declarative security / Programmatic security
  - Loosely coupled events
    - Publishers /Subscribers
  - JIT activation
    - Exists in 3 states Exists and Activated, Exists and not activated, Nonexistent.
  - Object pooling
- PRESENTATION SERVICES APPLICATION SERVICES DATA SERVICES

  SYSTEM SERVICES
- Transactions group a set of tasks into a single execution unit.
- COM+ components automatically participate in transactions
  - BeginTransaction → CommitTransaction
    or
    AbortTransaction (RollbackTransaction)
- ACID (Atomicity, Consistency, Isolation, durability)
- COM+ component exists in 3 states; exists and activated; exists and not activated; nonexistent;
- System.EnterpriseServices
- After you created a serviced component, you need to host it in a COM+ application. For that you must assign a strong name to the assembly, register the assembly in the windows registry and install the type library definition in a COM+ application.
- COM+ application is like AppDomain
- Tlbexp.exe

- To debug a COM+ application (service component) you need to attach the debugger to the DLLHost.exe [not to the client process]
- For debuggin a Library application, you can attach the debugger to the client library application.
- You can identify an existing COM+ target application by NAME or GUID. You can register a COM class by adding a GUID attributes to it.

Console Applications

- Dont need/have graphical user interface
- Most of the dos based utilities are console apps. ex., fdisk, zip,...
- Run at command prompt
- Run in a DOS window
- Generally write to the standard out stream
- Can read from the standard in stram (generally keyboard)
- Can write to the standard error stream (generally the screen, but can be redirected)
- Can support redirection ex., `c:\> addnumber < inputfile.txt > result.txt`
- In DOS there are 3 predefined streams
    - In
    - Out
    - Error
- Console Object

  Properties: `In, Out, Error`

  Methods:
  ```
  OpenStandardIn(), OpenStandardOut(), OpenStandardError()
  Read(), ReadLine()
  Write(), WriteLine()
  SetIn(), SetOut(), SetError()
  ```

Transaction Attributes

- Required
- Required New
- Disabled
- Support
- Not Supported

XML [eXtensible Markup Language]

- XML is a series of nested items, including elements and attributes.
- You can access an XML file by using
  - DOM [Document Object Model]
    - DOM represents an XML document as a tree of nodes
  - XmlReader
    - XmlReader provides forward only, read only access to XML file [similar to cursor and resultset concept]

XmlDocument / XmlDataDocument

- XmlDocument object enables you to load either relational data or XML data and manipulate that data DOM. In other words it represents an entire XML file.
- XmlDataDocument
  - Allows connections between XmlDocument & DataSets
  - There are 3 ways to synchronize a DataSet and XmlDataDocument object
    - Start with XmlDataDocument and retrieve the DataSet from its DataSet property
    - Start with a DataSet object
    - Starting with XML Schema
  - Create a new dataset object with proper schema to match the xml document, but no data.
  - Create the XmlDataDocument object from the DataSet object.
  - Load the XML document into the XmlDataDocument.
- Any changes made to the XmlDataDocument are reflected in the DataSet and vice versa.
- XmlNode represents a single node in DOM.

XPath / XPathNavigator / XPathDocument

- XPath is like a query language and is used to query XML data
- XPath starts with the notation of current context. The current context defines the set of nodes that an XPath query will inspect.
  - **./** uses current node as current context
  - **/** uses the root of xml document as current context
  - **.//** uses the entire xml hierarchy starting with the current node as current context
  - **//** uses the entire xml document as current context [documentwide wildcard]
- XPath examples
  - /Books/Book/Author selects all Author elements [fast]
  - //Author selects all Author elements [slow]
  - /Books/*/Author selects all Author elements that are grandchildren of Book
  - * is the wildcard
  - /Books/Book/@Pages selects all Pages attributes from Book elements
  - @ is used for attributes
  - //@Pages selects all Pages attributes
  - //Books/@* selects all attributes of Book elements
  - /Books/Book/Publisher[.="oreily"] selects all Publisher nodes with value "oreily"
  - [] is the filter pattern and also indexing. .(DOT) opertor stands for current node.
  - /Books/Book[./Publisher="oreily"] selects all Publisher nodes with value "oreily"
  - /Books/Book[./@Pages>=1000] selects all Books having 1000 or more pages
  - /Books/Book[starts-with(Title,"A")] selects all Books whose title starts with A
  - /Books/Book[1] selects the first book node
  - /Books/Book[2]/Title[1] selects the first Title of the second Book.
  - (Books/Book/Author)[1] selects the first Author in the XML file
  - /Books/Book[last()] selects the last Book in the XML file
  - | is used as union.

- XPathNavigator allows random-access navigation of xml file. XPathNavigator can perform 2 distinct tasks.
    - Selecting a set of nodes with an XPath expression
    - Navigating the DOM representation of the XML expression.
- XPathDocument class provides a representation of the XML document that is optimized for query operations.
- XPathDocument and XPathNavigator objects are optimized for fast execution of XPath queries.
- You can use .Compile() of XPathNavigator to create a precompiled XPathExpression.

XSD Schema

- XSD schema - XML file can contain
    - Explicit schema info in the form of embedded schema
    - Implicit schema info in its structure. [Inferring an XML schema]
- The prime use of a schema file is to validate the corresponding XML file.
- An inline schema cannot contain an entry for the root element of the document. So Xml documents with inline schema will throw at least one validation error.
- A valid XML file is one whose structure conforms to a specification. The specification can be in the form of a. XML Schema
  b. DTD (Document Type Description)
- There are at least 4 ways to obtain XSD files
    - Use a file generated by SQL Server or Access
    - Create your own schema from scratch
    - Extract inline schema from an XML file using DataSet.ReadXmlSchema()
    - Infer schema information from an XML file using DataSet.InferXmlSchema()
- Prime use of schema file (.xsd) is to validate the corresponding xml file
- xsd.exe is XML schema definition tool is used to generate and xml schema from a DLL or EXE, generate dataset classes from and xml schema file.

DTD Vs Schema

- Entity declarations cannot be done in schema.
- Where as schema has XML like structure in comparison with DTDs.
- DTD has no mechanism for defining the content of elements in terms of data types.

XmlReader is abstract base class that provides non-cached, forward-only and read-only access. Basic implementation of XmlReader is XmlTextReader, XmlNodeReader and XmlValidatingReader. When you read XML data using XML Reader, nodes are not included for ATTRIBUTES but only for ELEMENTS and the text they contain.

XmlValidatingReader

XmlValidatingReader is used for validating an XML file. This does not stop on encountering errors but allows you to handle errors.

XmlReader VS SAX Reader

The big difference lies in the fact that the SAX model is a "push" model, where the parser pushes events to the application, notifying the application every time a new node has been read, while applications using XmlReader can pull nodes from the reader at will.

DiffGram
The DiffGram is one of the two XML formats that you can use to render DataSet object contents to XML.

- Diffgram is a before-and-after snapshot of a part of a SQL Server Table.
- DiffGrams can insert or delete or modify data.
- Diffgrams are properly formatted XML messages used to modify SQL Serve Database tables.
- .NET uses DiffGrams internally as means of serializing changes in a DataSet object.
- You cannot performa DiffGram update without a schema file that specifies the mapping between xml elements and data columns.

FOR XML

- FOR XML clause in the SQL is used to directly generate XML from SQL Server
- FOR XML clause allows you to retrieve the results as an XML rather than as a ResultSet.
- *… FOR XML RAW*
  returns one element (always named row) for each row of the resultset.
- *… FOR XML AUTO*
  nested tables in the resultset are represented as nested elements in the XML. Columns are still represented as attributes.
- *… FOR XML AUTO, ELEMENTS*
  represents columns as elements rather than as attributes
- *… For XML EXPLICIT*
  you must construct you query so as to create a resultset with the first column named TAG and the second column named PARENT.
    - Explicit mode allows you the finest control over the generated XML. But you should stick to RAW or AUTO mode wherever possible.

SqlCommand.ExecuteXmlReader()
ExecuteXmlReader() method of SqlCommand object enables you to retrieve and XML Reader directly from a SQL statement, provided that the SQL statement uses the FOR XML clause.

Filtering expressions needs to start with ./ operator to indicate that it is filtering nodes under the current node.

AVT (Attribute Value Templates)
An AVT, coded as an XPath expression enclosed in "curly braces" -- {and} characters -- takes content directly from the source tree, particularly attribute values, and plugs it directly into the result without the somewhat clunkier intermediate step of using xsl:attribute, etc.

XmlAttribute [XmlAttributeAttribute] class is used to translate a public field to an xml attribute when an object is serialized.

A Java Programmer Looks at C# Delegates

by Steven M. Lewis and Wilhelm Fitzpatrick

05/21/2003

Abstract

While C# has a set of capabilities similar to Java, it has added several new and interesting features. Delegation is the ability to treat a method as a first-class object. A C# delegate is used where Java developers would use an interface with a single method. In this article, the use of delegates in C# is discussed, and code is presented for a Java Delegate object that can perform a similar function. Download the source code here.

C#, pronounced C Sharp, is a language developed by Microsoft as a part of its .NET initiative. The language is extremely similar to Java. Were it not for legal difficulties between Microsoft and Sun, there is little question that Microsoft would have chosen Java to fill the role in its plans that is currently held by C#. The major features C# has in common with Java include garbage collection, a virtual machine, single inheritance, interfaces, packages (called namespaces), and the fact that all code is encapsulated within objects.

There are also a few significant differences between the two languages. Because the developers of C# had the advantage of carefully examining Java while developing their language, it is not surprising that some of the differences attempt to address significant problems that are difficult to deal with in Java. This article focuses on one item that Microsoft added to C#, why it was added, and how similar functionality could be implemented in Java.

Delegates in C#

C# introduces the concept of delegates, which represent methods that are callable without knowledge of the target object. Consider these situations:

Java Code

```
public class Class1 {
   ...
   public void show(String s) { .. }
}

public class Class2 {
   ...
   public void show(String s) { .. }
}
```

Here, two classes share a common method, show, performing a similar function, display of data. We would like to be able to call that method in the same way for Class1 and Class2. If the two classes share a common interface, we can simply treat instances of either class as instances of that interface. Unfortunately, if the classes do not share an interface, as in the above example, there is no easy way to make a uniform call. If the developer controls the code for both classes, it is possible to retrofit a common interface. When one or both classes are in library code, there is no easy fix.

A more complex situation is illustrated in the example below:

Java Code

```
public class Class1 {
   ...
   public void show(String s) { .. }
}

public class Class2 {
   ...
   public void display(String s) { .. }
}
```

These two classes have the methods show and display, which perform a similar function and have a similar signature. That is, they take similar arguments, return similar data, and could be used in a loop doing conceptually similar things. However, because the names of the two methods are different, no interface will recognize the two as performing the same action.

Java developers may address these issues through reflection (by generating an interface and implementing it with inner wrapper classes) or by constructing a dynamic proxy. All of these solutions are somewhat clumsy and require non-trivial amounts of code.

Consider how C# would address this issue. In the sample below, we define three classes that implement similar methods, two with different names and a third with a static implementing method:
C# Code

```
// define three classes with similar methods
// two instances with differing names, one static.

public class Class1 {
   public void show(String s) { Console.WriteLine(s); }
}

public class Class2 {
   public void display(String s) { Console.WriteLine(s); }
}

// allows static method as well
public class Class3 {
   public static void staticDisplay(String s) { Console.WriteLine(s); }
}
```

We will now define a new data type, doShow, which abstracts the common features of the methods in all three classes. That is, they all take a single string as an argument and return void. This is done using the C# delegate keyword.
C# Code

```
public delegate void doShow(String s);
```

Think of a delegate as an interface declaring exactly one method. An instantiation of a delegate is similar to an anonymous inner class that implements the interface through a one-line call to a single method (static or instance) with a compatible signature.

With this new data type in hand, we may now arrange to invoke all three methods via a common abstraction:
C# Code

```
public class TestDelegate
{
   // define a datatype as a method taking a string returning void
   public delegate void doShow(String s);

   public static void Main(string[] args)
   {
      // make an array of these methods
      doShow[] items = new doShow[3];

      items[0] = new doShow(new Class1().show);
      items[1] = new doShow(new Class2().display);
      items[2] = new doShow(Class3.staticDisplay);
```

```
        // call all items the same way
        for(int i = 0; i < items.Length; i++) {
            items[i]("Hello World");
        }
    }
}
```

The main function creates an array populated with newly instantiated doShow objects. These refer to the various instance and class methods defined above. Let's take a closer look at the instantiation of a delegate:
C# Code

```
items[1] = new doShow(new Class2().display);
```

Related Reading
Programming C#

Programming C#
By Jesse Liberty
Table of Contents
Index
Sample Chapter

Read Online--Safari Search this book on Safari:

Code Fragments only

In C#, a method is a first-class object in the same way a Class like String.class is a first-class object in Java. In C#, if we reference a method on an object (omitting the parentheses that would normally signal a method call), C# instead treats the method name like a field, returning the object representing that method. The constructor of a C# delegate expects to be called with just such a method object.

In Java terms, C# is dynamically creating an interface that declares a single method. When one considers how many interfaces (especially listeners and other event handlers) fit this description, the utility of this approach is apparent. C# uses delegates rather than Listener interfaces to implement most of its event handling. Threads in C# are constructed with delegates (System.Threading.ThreadStart), unlike Java, which uses the Runnable interface.

When an instance of a delegate is constructed, the actions the compiler takes are similar to the Java equivalent of building a wrapper class. This wrapper class exposes the interface defined by the declaration of the delegate, implementing the interface by calling the method that was passed to the delegate constructor.

The C# approach requires hooks into the compiler not available in Java. These hooks allow methods to be accessed at compile time via a convenient syntax in much the same way we would write String.class to access a known class at compile time. This approach allows the C# compiler to perform type checking on the arguments to a delegate invocation when compiling it, rather than throwing a type error at runtime.

Implementing Delegates in Java

The code presented in this article implements a significant portion of the functionality of C# delegates in Java. Two ways of accomplishing this will be presented. In the first case, the developer describes the method to be delegated by providing a list of the parameter and return classes. In the second case, the parameters are deduced by examining a suitable interface that declares a single method. The code presents a factory class, Delegator, capable of handling either case. The factory method,

build, returns an object implementing the Delegate interface. Where the Delegator has been constructed with an interface, the return is a Proxy implementing that interface.

Click here to find out more!

The Delegator Class

Two methods may be considered comparable if the argument lists of each method are assignable the same common list of classes and if the return types are assignable to a common type. As an example, if Foo is a superclass of Bar, the two methods

```
public String m1(Foo p1);
public Object m2(Bar p1);
```

both match a signature taking Bar and returning Object. We may express this signature in code by providing a Class object that represents the return type and an array of Class to represent the parameter types. We may also express this signature by providing an interface with a single method to be used as an exemplar.

A method may match the signature described by a Delegator in the weak sense that all arguments are assignable to the declared types rather than the stronger test required by a Java interface that the arguments be identical. Also note that methods are considered comparable even it they throw different exceptions. Delegation will convert all exceptions into a runtime DelegateInvokeException emulating C# behavior (all C# exceptions act like RuntimeExceptions).

Delegator provides a factory method, build, which associates the method template with a specific implementation. The implementation is a combination of either an instance method and a target instance or a static method. In either case, the method must be compatible with the requested signature. The object returned by the build method will satisfy the Delegate interface, which contains the method:

```
public Object invoke(Object[] args);
```

The returned object is a thin wrapper that invokes the method on the supplied object, converting any checked exceptions returned by the wrapped object into a runtime DelegateInvokeException. The code in Scenario 1 shows use of this basic type of Delegate object.

Scenario 1. Using a generic Delegate object
Java Code

```
class Class1 {
   public void show(String s) { System.out.println(s); }
}

class Class2 {
   public void display(String s) { System.out.println(s); }
}

// allows static method as well
class Class3 {
   public static void staticDisplay(String s) { System.out.println(s); }
}

public class TestDelegate  {
   public static final Class[] OUTPUT_ARGS = { String.class };
   public final Delegator DO_SHOW = new Delegator(OUTPUT_ARGS,Void.TYPE);

   public void main(String[] args)  {
      Delegate[] items = new Delegate[3];
```

```
      items[0] = DO_SHOW .build(new Class1(),"show,);
      items[1] = DO_SHOW.build (new Class2(),"display");
      items[2] = DO_SHOW.build(Class3.class, "staticDisplay");

      for(int i = 0; i < items.length; i++) {
         items[i].invoke("Hello World");
      }
   }
}
```

The code described above offers many of the advantages of C# delegates. Methods, either static or dynamic, can be treated in a uniform manner. The complexity in calling methods through reflection is reduced and the code is reusable, in the sense of requiring no additional classes in the user code. Note we are calling an alternate convenience version of invoke, where a method with one parameter can be called without creating an object array.

Scenario 2. Delegating via an interface

One advantage of C# delegates that is still missing is static type checking enforced by the compiler. In the example above, it is possible to call invoke on a returned Delegate with a Date object, and the error will not be discovered until run time. In order to get the full benefits of compiler support, it is necessary to construct the Delegator with an interface. The interface may be well-known or special-purpose, but should declare a single method. The signature of that method becomes the template used by the Delegator. In addition, the returned object will be a proxy that implements the requested interface.
Java Code

```
// interface to implement
public static interface IStringDisplay {
   public void doDisplay(String s);
}

public final Delegator ST_DEL  = new Delegator(IStringDisplay.class);

public void testDelegate()
{
   IStringDisplay[] items = new IStringDisplay[3];

   // build the delegates
   items[0] = (IStringDisplay) ST_DEL.build(new Class1(),"show");
   items[1] = (IStringDisplay) ST_DEL.build(new Class1()2,"display");
   items[2] = (IStringDisplay) ST_DEL.build(Class3.class,"staticDisplay");

   // call the delegates
   for(int i = 0; i < items.length; i++) {
      items[i].doDisplay("test");
   }
}
```

Note that while a cast is required to convert the value returned from the build method into an instance of the desired interface, invocations of the delegated method are now made through the interface with full static type checking.
Thread Delegates

One major use of delegates in C# is in threading. Rather than constructing a thread with an instance of Runnable, threads in C# are constructed by using a delegate, Thread.ThreadStart, with the appropriate signature. In Java, a similar problem exists where a developer wants to call a no-

argument method as the active portion of a Runnable. While this may be accomplished with an anonymous inner class, the construct is clumsy and reduces the readability of the code.

This important usage pattern is supported by the Delegator class, which implements convenience methods to create delegates implementing Runnable. This is done by providing a static, final instance variable holding a Delegator constructed using the well-known interface Runnable, implementing two static methods that build delegates using this Delegator and cast the returned object to the underlying interface. Similar code could be used any time it is necessary to build many delegates that all implement a particular interface.
Java Code

```
static final Delegator RUNNABLE_DELEGATE = new Delegator(Runnable.class);

public static Runnable buildRunnable(Object o,String methodName) {
   return((Runnable)RUNNABLE_DELEGATE.build(o,methodName));
}

public static Runnable buildRunnable(Class c,String methodName) {
   return((Runnable)RUNNABLE_DELEGATE.build(c,methodName));
}
```

The above code can be called with a line such as:

```
Runnable r = Delegator. buildRunnable(this,methodName);
```

Note that because a special-purpose method has been used, there is no need to cast the return value from buildRunnable. The cast is performed in the method implementation.
How It Works

The critical code is in the method build. This method constructs a DelegateProxy, exposed through an Delegate interface that is a wrapper around the method named in the call to build. If the Delegator was constructed by specifying an interface, the returned object is wrapped in a dynamic Proxy so that it will appear to the Java runtime as an instance of the requested interface.
Java Code

```
/**
  * @param target non-null target with a bindable method
  * @param MethodName  name of the  method
  * @return non-null IDelegate. If getInterface() is non-null the returned
  * Delegate will be a dynamic proxy implementing that interface
  */

public Delegate build(Object target,String methodName)
{
   Class myInterface        = getInterface();
   DelegateProxy theDelegate = new DelegateProxy(target,methodName,this);

   // build a dynamic proxy
   if(myInterface != null) {
      Class[] interfaces = { myInterface,Delegate.class };
      Delegate ret       = (Delegate)java.lang.reflect.Proxy.newProxyInstance(
                        target.getClass().getClassLoader(),
                        interfaces, theDelegate);
      return(ret);
   }

   return theDelegate;
}
```

The constructor DelegateProxy(target,methodName,this) uses reflection to find the best method in the target class matching the signature contained in the Delegator. When an interface has been specified, the DelegateProxy can be used as an InvocationHandler to construct a Proxy implementing the specified interface. The returned object may then be called using the Delegate's invoke method or, if an interface is specified in the Delegator, cast to that interface and used as an instance of that interface.
Usage

Sufficient information to build delegates is present once classes are loaded. Binding a delegate is a non-trivial operation requiring identification of an appropriate method in the target class. When a delegate is constructed with an instance method, the build call can occur any time after the target instance has been created. Delegates invoking static methods can be constructed at load time. It is usually a good idea to build delegatees as early as possible, caching them for later use. Actually performing method calls through delegates is relatively cheap.

Code Fragments only
Timing and Performance Costs

The most logical use of delegates involves messaging and event handling where the code is infrequently executed; i.e., not in a tight loop. In these situations, executing the code contained in the target method usually takes significantly longer than the process of finding and invoking the method. In addition, Hotspot and JDK 1.4 have significantly reduced the cost of method calls. I found that executing the loop in TestDelegate (three calls) for 10,000,000 iterations took 43 seconds on a 1.5GHz Athlon processor running JDK 1.4 under Windows 2000. This averages slightly more than one microsecond per call. This cost can be ignored in all but the tightest loops.
Discussion

This approach represents an implementation of the Adapter pattern (Gamma et al, Design Patterns). It differs from a Proxy in that an Adapter maps a number of different methods into identical calls, allowing multiple objects implementing methods with different names but compatible signatures to be used interchangeably. It will also work with or without a target interface to implement. The Delegator class simplifies and generalizes the steps in creating an Adapter. Delegates are simple to use, and a single delegate instance may be reused multiple times.

Where an interface is known or can be constructed, use of delegates provides a simple, readable way to coerce a method into implementing the actions in that interface. Common interfaces, such as Runnable and many event listeners, may easily be mapped to any matching public method. In these cases, the Java code is almost as simple as the code that could be written using C#'s built-in delegate construct.

In my own development, I find that delegates implementing interfaces are more useful than those using invoke. The buildRunnable method is especially useful. In Swing programming, where large numbers of Runnables are needed to pass control to the swing thread, the ability to turn methods into Runnables is particularly useful. Delegates allow me to largely eliminate the need for anonymous inner classes, improving the readability of my code.

UML [Unified Modeling Language]

UML is a language for visualizing, specifying, constructing and documenting the artifacts of a software intense system.

Collaboration	defines an interaction and is a society of roles and other elements that work together to provide some cooperative behavior that's bigger than sum of all elements.
UseCase	is a description of set of sequence of actions that system performs that yields an observable result of value to a particular actor.
Actor	is a coherent set of roles that user of use cases play when interacting with use cases.
StereoType	is extension of the vocabulary of the UML, which allows you to create new kinds of building blocks that are derived from existing ones but that are problem specific.

Building block of UML

- **Things**
  1. Structural
  2. Behavioral
  3. Grouping
  4. Annotation
- **Relationships**
  1. Dependency
  2. Association
  3. Generalization
  4. Realization
- **Diagrams**
  1. Class Diagram
     A structural diagram that shows a set of classes, interfaces, collaborations and their relationships
  2. Object Diagram
     A structural diagram that shows a set of objects and their relationships.
  3. Sequence Diagram
     A behavioral diagram that shows as interaction, emphasizing the time ordering of messages.
  4. Collaboration Diagram
     A behavioral diagram that shows an interaction, emphasizing the structural organization of the objects and send and receive messages.
  5. Statechart Diagram
     A behavioral diagram that shows a state machine, consisting of states, transitions, events and activities.
  6. Activity Diagram
     A behavioral diagram that shows a state machine, emphasizing the flow from activity to activity.
  7. Component Diagram
     A structural diagram that shows component and their relationships
  8. Deployment Diagram
     A structural diagram that shows a set of nodes and their relationships.

Steps in **RUP** (Rational Unified Processing)
It's a **process independent**, meaning that it is not tied to any particular **software development life cycle**.

Processes used are

- **Use case driven**
- **Architecture-centric**
- **Iterative and incremental** is a risk driven process, meaning that each new release is focused on attacking and reducing the most significant risks to the success of the project.

Basic process workflows are

- Business Modeling
- Requirements
- Analysis and design
- Implementation
- Test
- Deployment

Design Patterns

- ***Creational Patterns -*** *Creational design patterns abstract the instantiation process. They help* make a system independent of how its objects are created, composed, and represented.
  - **Factory** pattern
    is used to choose and return an instance of a class from a number of similar classes based on data you provide to the factory.
  - **Abstract Factory** pattern
    is used to return one of several groups of classes. In some cases it actually returns a Factory for that group of classes.
  - **Builder** pattern
    assembles a number of objects to make a new object, based on the data with which it is presented. Frequently, the choice of which way the objects are assembled is achieved using a Factory.
  - **Prototype** pattern
    copies or clones an existing class rather than creating a new instance when creating new instances is more expensive.
  - **Singleton** pattern
    is a pattern that insures there is one and only one instance of an object, and that it is possible to obtain global access to that one instance.
- ***Structural Pattern -*** *deals with the composition of classes and objects.*
  - **Adapter pattern**
    is used to change the interface of one class to that of another one.
  - **Bridge** pattern
    is intended to keep the interface to your client program constant while allowing you to change the actual kind of class you display or use. You can then change the interface and the underlying class separately.
  - **Composite** pattern
    is a collection of objects, any one of which may be either itself a Composite, or just a primitive object.
  - **Decorator** pattern
    is a class that surrounds a given class, adds new capabilities to it, and passes all the unchanged methods to the underlying class.
  - **Façade** pattern
    groups a complex object hierarchy and provides a new, simpler interface to access those data.
  - **Flyweight** pattern
    provides a way to limit the proliferation of small, similar class instances by moving some of the class data outside the class and passing it in during various execution methods.
  - **Proxy** pattern
    provides a simple place-holder class for a more complex class which is expensive to instantiate.
- ***Behavioral Pattern -*** *Behavioral patterns are those patterns that are most specifically concerned with communication between objects. In this chapter, we'll see that:*
  - **Observer** pattern
    defines the way a number of classes can be notified of a change,
  - **Mediator** pattern

defines how communication between classes can be simplified by using another class to keep all classes from having to know about each other.

- **Chain of Responsibility** pattern
  allows an even further decoupling between classes, by passing a request between classes until it is recognized.
- **Template** pattern
  provides an abstract definition of an algorithm, and
- **Interpreter** pattern
  provides a definition of how to include language elements in a program.
- **Strategy** pattern
  encapsulates an algorithm inside a class,
- **Visitor** pattern
  adds function to a class
- **State** pattern
  provides a memory for a class's instance variables.
- **Command** pattern
  provides a simple way to separate execution of a command from the interface environment that produced it, and
- **Iterator** pattern
  formalizes the way we move through a list of data within a class.

.NET Interview Questions

Does C# support multiple-inheritance?
No, use interfaces instead.

When you inherit a protected class-level variable, who is it available to?
Classes, in the same namespace.

Are private class-level variables inherited?
Yes, but they are not accessible. Although they are not visible or accessible via the class interface, they are inherited.

Describe the accessibility modifier "protected internal".
It is available to derived classes and classes within the same Assembly (and naturally from the base class it's declared in). ***"Within the same assembly"***

C# provides a default class constructor for me. I decide to write a constructor that takes a string as a parameter, but want to keep the constructor that has no parameter. How many constructors should I write?
Two. Once you write at least one constructor, C# cancels the freebie constructor, and now you have to write one yourself, even if there's no implementation in it.

What's the top .NET class that everything is derived from?
System.Object.

What does the term immutable mean?
Answer the question.

What's the difference between System.String and System.StringBuilder classes?
System.String is immutable. System.StringBuilder was designed with the purpose of having a mutable string where a variety of operations can be performed.

What's the advantage of using System.Text.StringBuilder over System.String?
StringBuilder is more efficient in cases where there is a large amount of string manipulation. Strings are immutable, so each time it's being operated on, a new instance is created.

Can you store multiple data types in System.Array?
No. Arrays are for storing similar data types only.

What's the difference between the System.Array.CopyTo() and System.Array.Clone()?
Clone() → creates a shallow copy of the Array. (copies only the structure)
CopyTo() → copies a section of one Array to another Array and performs type casting and boxing as required. Performs a deep copy of the Array. (structure + data)

How can you sort the elements of the array in descending order?
By calling Sort() and then Reverse() methods.

What class is underneath the SortedList class?
A Sorted HashTable.

What's the .NET class that allows the retrieval of a data element using a unique key?
HashTable.

Will the finally block get executed if an exception has not occurred?
Yes.

What's the C# equivalent of C++ catch (…), which was a catch-all statement for any possible exception?
A catch block that catches the exception of type System.Exception. You can also omit the parameter data type in this case and just write catch {}.

Can multiple catch blocks be executed?
No. Once the proper catch code fires off, the control is transferred to the finally block (if there are any), and then whatever follows the finally block.

Explain the three services model commonly know as a three-tier application.
MVC Model-View-Control. Presentation (UI), business (logic and underlying code) and data (from storage or other sources).

What is the role of the DataReader class in ADO.NET connections?
It returns a read-only dataset from the data source when the command is executed.
To do: Improve the answer.

How do you inherit from a class in C#?
Place a colon and then the name of the base class.

Can you prevent your class from being inherited by another class?
Yes. The keyword "sealed" will prevent the class from being inherited.

Can you allow a class to be inherited, but prevent the method from being over-ridden?
Yes. Just leave the class public and make the method sealed. **sealed m()**

What's an abstract class?
A class, which cannot be instantiated. An abstract class is a class that must be inherited and have the methods overridden. An abstract class is essentially a blueprint for a class without any implementation.

When do you absolutely have to declare a class as abstract?
When at least one of the methods in the class is abstract.
When the class itself is inherited from an abstract class, but not all base abstract methods have been overridden.

What's an interface class?
An Interface is a reference type and it contains only abstract members. Interface's members can be Events, Methods, Properties and Indexers. But the interface contains only declaration for its members. Any implementation must be placed in class that realizes them. The interface can't contain constants, data fields, constructors, destructors and static members. All the member declarations inside interface are implicitly public.

Why can't you specify the accessibility modifier for methods inside the interface?
They all must be public. Therefore, to prevent you from getting the false impression that you have any freedom of choice, you are not allowed to specify any accessibility, it's public by default.

Can you inherit multiple interfaces?
Yes, multiple inheritences is possible only through interfaces.

and if they have conflicting method names?
It's up to you to implement the method inside your own class, so implementation is left entirely up to you. This might cause a problem on a higher-level scale if similarly named methods from different interfaces expect different data, but as far as compiler cares you're okay.

What's the difference between an interface and abstract class?
In an interface class, all methods must be abstract. In an abstract class some methods can be concrete. In an interface class, no accessibility modifiers are allowed, which is ok in an abstract class.

What is the difference between a Struct and a Class?
Structs are value-type variables and are thus saved on the stack → additional overhead but faster retrieval. Another difference is that structs CANNOT inherit.

What's the implicit name of the parameter that gets passed into the set method/property of a class?
value. The data type of the value parameter is defined by whatever data type the property is declared as.

What does the keyword "virtual" declare for a method or property?
This means that the method or property can be overridden.

How is method overriding different from method overloading?
When overriding a method, you change the behavior of the method for the derived class. Overloading a method simply involves having another method with the same name within the class.

Can you declare an override method to be static if the original method is non-static?
No. The signature of the virtual method must remain the same, only the keyword ***virtual*** is changed to keyword ***override***.

Can you override private virtual methods?
No. Private methods are not accessible outside the class.
Original answer: No, moreover, you cannot access private methods in inherited classes, have to be protected in the base class to allow any sort of access.
To do: Can a private method even be declared a virtual?

What are the different ways a method can be overloaded?
Different parameter data types, different number of parameters, different order of parameters.

If a base class has a number of overloaded constructors, and an inherited class has a number of overloaded constructors; can you enforce a call from an inherited constructor to a specific base constructor?
Yes, just place a colon, and then keyword base (parameter list to invoke the appropriate constructor) in the overloaded constructor definition inside the inherited class.
To do: question is to complex. It can be stated better.

Why is it a bad idea to throw your own exceptions?
Well, if at that point you know that an error has occurred, then why not write the proper code to handle that error instead of passing a new Exception object to the catch block?
Throwing your own exceptions signifies some design flaws in the project.

What's a delegate?
A delegate can hold reference[s] to one more more functions and invoke them as and when needed.
A delegate object encapsulates a reference to a method.

What's a multicast delegate?
It's a delegate that points to and eventually fires off several methods. Muticast delegates point to more than one function at a time (that is, they're based off the System.MulticastDelegate type). A multicast delegate maintains a list of functions that will all be called when the delegate is invoked.

How is the DLL Hell problem solved in .NET?
Assembly versioning allows the application to specify not only the library it needs to run (which was available under Win32), but also the version of the assembly.

The problem of DLL HELL arises only when, the dll files of the two versions try to use the same memory location,instead of of choosing differemt location (as in VB6.0). DLL files of same name in a same directory, but different versions is termed as "DLL HELL".

What are the ways to deploy an assembly?
An MSI installer, a CAB archive, and XCOPY command.

What is a satellite assembly?
When you write a multilingual or multi-cultural application in .NET, and want to distribute the core application separately from the localized modules, the localized assemblies that modify the core application are called satellite assemblies.

What namespaces are necessary to create a localized application?
System.Globalization and System.Resources.

What's the difference between // comments, /* */ comments and /// comments?
Single-line comments, multi-line comments, and documentation comments.

How do you generate documentation from the C# file commented properly with a command-line compiler?
Compile it with the /doc switch.

Is XML case-sensitive?
Yes.

What debugging tools come with the .NET SDK?
CorDBG - command-line debugger. To use CorDbg, you must compile the original C# file using the /debug switch.
DbgCLR - graphic debugger. Visual Studio .NET uses the DbgCLR.

What does the "This" window show in the debugger?
It points to the object that's pointed to by this reference. Object's instance data is shown.
To do: Need a better answer.

What does assert() method do?
In debug compilation, assert takes in a Boolean condition as a parameter, and shows the error dialog if the condition is false. The program proceeds without any interruption if the condition is true.

What's the difference between the Debug class and Trace class?
Documentation looks the same. Use Debug class for debug builds, use Trace class for both debug and release builds.

Why are there five tracing levels in System.Diagnostics.TraceSwitcher?
The tracing dumps can be quite verbose. For applications that are constantly running you run the risk of overloading the machine and the hard drive. Five levels range from None to Verbose, allowing you to fine-tune the tracing activities.

Where is the output of TextWriterTraceListener redirected?
To the Console or a text file depending on the parameter passed to the constructor.

How do you debug an ASP.NET Web application?
Attach the aspnet_wp.exe process to the DbgClr debugger.

What are three test cases you should go through in unit testing?
Positive test cases (correct data, correct output).
Negative test cases (broken or missing data, proper handling).
Exception test cases (exceptions are thrown and caught properly).

Can you change the value of a variable while debugging a C# application?
Yes.If you are debugging via Visual Studio.NET, just go to Immediate window.

What are advantages and disadvantages of Microsoft-provided data provider classes in ADO.NET?
SQLServer.NET data provider is high-speed and robust, but requires SQL Server license purchased from Microsoft. OLE-DB.NET is universal for accessing other sources, like Oracle, DB2, Microsoft Access and Informix. OLE-DB.NET is a .NET layer on top of the OLE layer, so it's not as fastest and efficient as SqlServer.NET.

What is the wildcard character in SQL?
Let's say you want to query database with LIKE for all employees whose name starts with La. The wildcard character is %, the proper query with LIKE would involve 'La%'.

Explain ACID rule of thumb for transactions.
A transaction must be:
Atomic - it is one unit of work and does not dependent on previous and following transactions.
Consistent - data is either committed or roll back, no "in-between" case where something has been updated and something hasn't.
Isolated - no transaction sees the intermediate results of the current transaction.
Durable - the values persist if the data had been committed even if the system crashes right after.

What connections does Microsoft SQL Server support?
Windows Authentication (via Active Directory) and SQL Server authentication (via Microsoft SQL Server username and password).

Which one is trusted and which one is untrusted?
Windows Authentication is trusted because the username and password are checked with the Active Directory, the SQL Server authentication is untrusted, since SQL Server is the only verifier participating in the transaction.
To do: ask the question better.

Why would you use untrusted verificaion?
Web Services might use it, as well as non-Windows applications.
To do: answer better.

What does the Initial Catalog parameter define in the connection string?
The database name to connect to.

What is the data provider name to connect to an Access database?
Microsoft.Access.

What does the Dispose method do with the connection object?
Deletes it from the memory.
Call Dispose as soon as you are done using an object, especially if the object will remain in scope during a lengthy operation or if the object allocates significant resources of its own.

Destructors, Finalize()?
~MyClass(){}, this syntax is simply a shortcut for declaring a Finalize()

What is a pre-requisite for connection pooling?
Multiple processes must agree that they will share the same connection, where every parameter is the same, including the security settings. The connection string must be identical.

Where does the Web page belong in the .NET Framework class hierarchy?
System.Web.UI.Page

Where do you store the information about the user's locale?
System.Web.UI.Page.Culture

What's the difference between Codebehind="MyCode.aspx.cs" and Src="MyCode.aspx.cs"?
CodeBehind is relevant to Visual Studio.NET only.

What's a bubbled event?
When you have a complex control, like DataGrid, writing an event processing routine for each object (cell, button, row, etc.) is quite tedious. The controls can bubble up their eventhandlers, allowing the main DataGrid event handler to take care of its constituents.

Suppose you want a certain ASP.NET function executed onMouseOver over a certain button. Where do you add an event handler?
It's the Attributesproperty, the Add function inside that property. So btnSubmit.Attributes.Add("onMouseOver","someClientCode();")

What data type does the RangeValidator control support?
Integer, String and Date.

Explain the differences between Server-side and Client-side code?
Server-side code runs on the server. Client-side code runs in the clients' browser.

What type of code (server or client) is found in a Code-Behind class?
Server-side code.

Should validation (did the user enter a real date) occur server-side or client-side? Why?
Client-side. This reduces an additional request to the server to validate the users input.

What does the "EnableViewState" property do? Why would I want it on or off?
It enables the viewstate on the page. It allows the page to save the users input on a form. Turning it off will not store the users input.

What is the difference between Server.Transfer and Response.Redirect? Why would I choose one over the other?
Response.Redirect() is client side redirection. Can connect to any .aspx pages / non-.aspx resources on any server. [302 status header is created]
Server.Transfer() is server side redirection. Can be used for .aspx pages only. .aspx page must be on the same serverand should not contain URL.
Additional info: Server.Execute() executes an .aspx within the current requested page.

Can you explain the difference between an ADO.NET Dataset and an ADO Recordset?
A DataSet can represent an entire relational database in memory, complete with tables, relations, and views. Data in a DataSet is bulk-loaded, rather than being loaded on demand. There's no concept of cursor types in a DataSet. DataSets have no current record pointer You can use For Each loops to move through the data. You can store many edits in a DataSet, and write them to the original data source in a single operation.
TODO: RecordSet

Can you give an example of what might be best suited to place in the Application_Start and Session_Start subroutines?
This is where you can set the specific variables for the Application and Session objects.

If I'm developing an application that must accommodate multiple security levels though secure login and my ASP.NET web application is spanned across three web-servers (using round-robin load balancing) what would be the best approach to maintain login-in state for the users?
Maintain the login state security through a database.

Can you explain what inheritance is and an example of when you might use it?
When you want to inherit (use the functionality of) another class. Base Class Employee. A Manager class could be derived from the Employee base class.

Whats an assembly?
Assemblies are the building blocks of the .NET framework.

Describe the difference between inline and code behind.
Inline code written along side the html in a page. Code-behind is code written in a separate file and referenced by the .aspx page.

Explain what a diffgram is, and a good use for one?
The DiffGram is one of the two XML formats that you can use to render DataSet object contents to XML. For reading database data to an XML file to be sent to a Web Service.

Whats MSIL, and why should my developers need an appreciation of it if at all?
MSIL is the Microsoft Intermediate Language. All .NET compatible languages will get converted to MSIL.

Which method do you invoke on the DataAdapter control to load your generated dataset with data?
The .Fill() method

Can you edit data in the Repeater control?
No, it just reads the information from its data source

Which template must you provide, in order to display data in a Repeater control?
ItemTemplate

How can you provide an alternating color scheme in a Repeater control?
Use the AlternatingItemTemplate

What property must you set, and what method must you call in your code, in order to bind the data from some data source to the Repeater control?
You must set the DataSource property and call the DataBind method.

What base class do all Web Forms inherit from?
The Page class.

Name two properties common in every validation control?
ControlToValidate property and Text property.

What tags do you need to add within the asp:datagrid tags to bind columns manually?
Set AutoGenerateColumns Property to false on the datagrid tag

What tag do you use to add a hyperlink column to the DataGrid?

What is the transport protocol you use to call a Web service?
SOAP is the preferred protocol.

A Web service can only be written in .NET?
False

Where on the Internet would you look for Web services?
http://www.uddi.org

Which property on a Combo Box do you set with a column name, prior to setting the DataSource, to display data in the combo box?
DataTextField property

Which control would you use if you needed to make sure the values in two different controls matched?
CompareValidator Control

To test a Web service you must create a windows application or Web application to consume this service?
False, the webservice comes with a test page and it provides HTTP-GET method to test.

How many classes can a single .NET DLL contain?
It can contain many classes.

What do you mean by authentication and authorization?
Authentication is the process of validating a user on the credentials (username and password) and authorization performs after authentication. After Authentication a user will be verified for performing the various tasks, It access is limited it is known as authorization.

How do you create a permanent cookie?
Permanent cookies are the ones that are most useful. Permanent cookies are available until a specified expiration date, and are stored on the hard disk. The location of cookies differs with each browser, but this doesn't matter, as this is all handled by your browser and the server. If you want to create a permanent cookie called Name with a value of Nigel, which expires in one month, you'd use the following code
Response.Cookies ("Name") = "Nigel"
Response.Cookies ("Name"). Expires = DateAdd ("m", 1, Now ())

What tag do you use to add a hyperlink column to the DataGrid?
< asp:HyperLinkColumn > </ asp:HyperLinkColumn>

Which method do you use to redirect the user to another page without performing a round trip to the client?
Server.transfer

Can a user browsing my Web site read my Web.config or Global.asax files?
No. The <HTTPHANDLERS>section of Machine.config, which holds the master configuration settings for ASP.NET, contains entries that map ASAX files, CONFIG files, and selected other file types to an HTTP handler named HttpForbiddenHandler, which fails attempts to retrieve the associated file. You can modify it by editing Machine.config or including an section in a local Web.config file.

What's the difference between Page.RegisterClientScriptBlock and Page.RegisterStartupScript?
RegisterClientScriptBlock is for returning blocks of client-side script containing functions. RegisterStartupScript is for returning blocks of client-script not packaged in functions-in other words, code that's to execute when the page is loaded. The latter positions script blocks near the end of the

document so elements on the page that the script interacts are loaded before the script runs.<%@ Reference Control="MyControl.ascx" %>

What are different types of directives in .NET?

@Page: Defines page-specific attributes used by the ASP.NET page parser and compiler. Can be included only in .aspx files
<%@ Page AspCompat="TRUE" language="C#" %>

@Control: Defines control-specific attributes used by the ASP.NET page parser and compiler. Can be included only in .ascx files.
<%@ Control Language="VB" EnableViewState="false" %>

@Import: Explicitly imports a namespace into a page or user control. The Import directive cannot have more than one namespace attribute. To import multiple namespaces, use multiple @Import directives.
<% @ Import Namespace="System.web" %>

@Implements: Indicates that the current page or user control implements the specified .NET framework interface.
<%@ Implements Interface="System.Web.UI.IPostBackEventHandler" %>

@Register: Associates aliases with namespaces and class names for concise notation in custom server control syntax.
<%@ Register Tagprefix="Acme" Tagname="AdRotator" Src="AdRotator.ascx" %>

@Assembly: Links an assembly to the current page during compilation, making all the assembly's classes and interfaces available for use on the page.
<%@ Assembly Name="MyAssembly" %><%@ Assembly Src="MySource.vb" %>

@OutputCache: Declaratively controls the output caching policies of an ASP.NET page or a user control contained in a page

```
<%@ OutputCache     Duration="#ofseconds"
                    Location="Any | Client | Downstream | Server | None"
                    Shared="True | False"
                    VaryByControl="controlname"
                    VaryByCustom="browser | customstring"
                    VaryByHeader="headers"
                    VaryByParam="parametername" %>
```

@Reference: Declaratively indicates that another user control or page source file should be dynamically compiled and linked against the page in which this directive is declared.

What is the transport protocol you use to call a Web service SOAP ?

HTTP Protocol

Is it necessary to lock application state before accessing it?

Only if you're performing a multistep update and want the update to be treated as an atomic operation. Here's an example:

```
Application.Lock ();
Application["ItemsSold"] = (int) Application["ItemsSold"] + 1;
Application["ItemsLeft"] = (int) Application["ItemsLeft"] - 1;
Application.UnLock ();
```

By locking application state before updating it and unlocking it afterwards, you ensure that another request being processed on another thread doesn't read application state at exactly the wrong time and see an inconsistent view of it.

If I update session state, should I lock it, too? Are concurrent accesses by multiple requests executing on multiple threads a concern with session state?

Concurrent accesses aren't an issue with session state, for two reasons. One, it's unlikely that two requests from the same user will overlap. Two, if they do overlap, ASP.NET locks down session state during request processing so that two threads can't touch it at once. Session state is locked down when the HttpApplication instance that's processing the request fires an AcquireRequestState event and unlocked when it fires a ReleaseRequestState event.

Do ASP.NET forms authentication cookies provide any protection against replay attacks? Do they, for example, include the client's IP address or anything else that would distinguish the real client from an attacker?
No. If an authentication cookie is stolen, it can be used by an attacker. It's up to you to prevent this from happening by using an encrypted communications channel (HTTPS). Authentication cookies issued as session cookies, do, however,include a time-out valid that limits their lifetime. So a stolen session cookie can only be used in replay attacks as long as the ticket inside the cookie is valid. The default time-out interval is 30 minutes.You can change that by modifying the timeout attribute accompanying the <forms> element in Machine.config or a local Web.config file. Persistent authentication cookies do not time-out and therefore are a more serious security threat if stolen.

What are VSDISCO files?
VSDISCO files are DISCO files that support dynamic discovery of Web services. If you place the following VSDISCO file in a directory on your Web server, for example, it returns references to all ASMX and DISCO files in the host directory and any subdirectories not noted in <exclude> elements:

```
<?xml version="1.0" ?>
<dynamicDiscovery
xmlns="urn:schemas-dynamicdiscovery:disco.2000-03-17">
<exclude path="_vti_cnf" />
<exclude path="_vti_pvt" />
<exclude path="_vti_log" />
<exclude path="_vti_script" />
<exclude path="_vti_txt" />
</dynamicDiscovery>
```

How does dynamic discovery work?
ASP.NET maps the file name extension VSDISCO to an HTTP handler that scans the host directory and subdirectories for ASMX and DISCO files and returns a dynamically generated DISCO document. A client who requests a VSDISCO file gets back what appears to be a static DISCO document.
Note that VSDISCO files are disabled in the release version of ASP.NET. You can reenable them by uncommenting the line in the <httpHandlers> section of Machine.config that maps *.vsdisco to System.Web.Services.Discovery.DiscoveryRequestHandler and granting the ASPNET user account permission to read the IIS metabase. However, Microsoft is actively discouraging the use of VSDISCO files because they could represent a threat to Web server security.

Is it possible to prevent a browser from caching an ASPX page?
Just call SetNoStore on the HttpCachePolicy object exposed through the Response object's Cache property, as demonstrated here:

```
<%@ Page Language="C#" %>
<html>
<body>
<%
Response.Cache.SetNoStore ();
Response.Write (DateTime.Now.ToLongTimeString ());
%>
</body>
</html>
```

SetNoStore works by returning a Cache-Control: private, no-store header in the HTTP response. In this example, it prevents caching of a Web page that shows the current time.

What does AspCompat="true" mean and when should I use it?
AspCompat is an aid in migrating ASP pages to ASPX pages. It defaults to false but should be set to true in any ASPX file that creates apartment-threaded COM objects--that is, COM objects registered

ThreadingModel=Apartment. That includes all COM objects written with Visual Basic 6.0. AspCompat should also be set to true (regardless of threading model) if the page creates COM objects that access intrinsic ASP objects such as Request and Response. The following directive sets AspCompat to true:

<%@ Page AspCompat="true" %>

Setting AspCompat to true does two things. First, it makes intrinsic ASP objects available to the COM components by placing unmanaged wrappers around the equivalent ASP.NET objects. Second, it improves the performance of calls that the page places to apartment- threaded COM objects by ensuring that the page (actually, the thread that processes the request for the page) and the COM objects it creates share an apartment. AspCompat="true" forces ASP.NET request threads into single-threaded apartments (STAs). If those threads create COM objects marked ThreadingModel=Apartment, then the objects are created in the same STAs as the threads that created them. Without AspCompat="true," request threads run in a multithreaded apartment (MTA) and each call to an STA-based COM object incurs a performance hit when it's marshaled across apartment boundaries.

Do not set AspCompat to true if your page uses no COM objects or if it uses COM objects that don't access ASP intrinsic objects and that are registered ThreadingModel=Free or ThreadingModel=Both.

What are ASP.NET Web Forms?
How is this technology different than what is available though ASP?
Web Forms are the heart and soul of ASP.NET. Web Forms are the User Interface (UI) elements that give your Web applications their look and feel. Web Forms are similar to Windows Forms in that they provide properties, methods, and events for the controls that are placed onto them. However, these UI elements render themselves in the appropriate markup language required by the request, e.g. HTML. If you use Microsoft Visual Studio .NET, you will also get the familiar drag-and-drop interface used to create your UI for your Web application.

What is different b/w webconfig.xml & Machineconfig.xml
Web.config & machine.config both are configuration files.Web.config contains settings specific to an application where as machine.config contains settings to a computer. The Configuration system first searches settings in machine.config file & then looks in application configuration files.Web.config, can appear in multiple directories on an ASP.NET Web application server. Each Web.config file applies configuration settings to its own directory and all child directories below it. There is only Machine.config file on a web server.

If I'm developing an application that must accomodate multiple security levels though secure login and my ASP.NET web appplication is spanned across three web-servers (using round-robbin load balancing) what would be the best approach to maintain login-in state for the users?
Use the state server or store the state in the database. This can be easily done through simple setting change in the web.config.

```
<SESSIONSTATE
StateConnectionString="tcpip=127.0.0.1:42424"
sqlConnectionString="data source=127.0.0.1; user id=sa; password="
cookieless="false"
timeout="30"
/>
```

You can specify mode as “stateserver” or “sqlserver”.

Where would you use an iHTTPModule, and what are the limitations of any approach you might take in implementing one
"One of ASP.NET's most useful features is the extensibility of the HTTP pipeline, the path that data takes between client and server. You can use them to extend your ASP.NET applications by adding pre- and post-processing to each HTTP request coming into your application. For example, if you wanted custom

authentication facilities for your application, the best technique would be to intercept the request when it comes in and process the request in a custom HTTP module.

How do you turn off cookies for one page in your site?
Since no Page Level directive is present, I am afraid that cant be done.

Can you give an example of what might be best suited to place in the Application_Start and Session_Start subroutines?
The Application_Start event is guaranteed to occur only once throughout the lifetime of the application. It's a good place to initialize global variables. For example, you might want to retrieve a list of products from a database table and place the list in application state or the Cache object. SessionStateModule exposes both Session_Start and Session_End events.

What are the advantages and disadvantages of viewstate?
Advantages
Simplicity. There is no need to write possibly complex code to store form data between page submissions.
Flexibility. It is possible to enable, configure, and disable ViewState on a control-by-control basis, choosing to persist the values of some fields but not others.
Disadvantages
Does not track across pages. ViewState information does not automatically transfer from page to page. With the session approach, values can be stored in the session and accessed from other pages. This is not possible with ViewState, so storing data into the session must be done explicitly.
ViewState is not suitable for transferring data for back-end systems. That is, data still has to be transferred to the back end using some form of data object.

Describe session handling in a webfarm, how does it work and what are the limits?
ASP.NET Session supports storing of session data in 3 ways, i] in In-Process (in the same memory that ASP.NET uses) , ii] out-of-process using Windows NT Service)in separate memory from ASP.NET) or iii] in SQL Server (persistent storage). Both the Windows Service and SQL Server solution support a webfarm scenario where all the web-servers can be configured to share common session state store.

Windows Service :
We can start this service by Start | Control Panel | Administrative Tools | Services | . In that we service names ASP.NET State Service. We can start or stop service by manually or configure to start automatically. Then we have to configure our web.config file

```
<CONFIGURATION><configuration>
<system.web>
<SessionState
mode = “StateServer”
stateConnectionString = “tcpip=127.0.0.1:42424”
stateNetworkTimeout = “10”
sqlConnectionString=”data source = 127.0.0.1; uid=sa;pwd=”
cookieless =”Flase”
timeout= “20” />
</system.web>
</configuration> </SYSTEM.WEB>
</CONFIGURATION>
```

Here ASP.Net Session is directed to use Windows Service for state management on local server (address : 127.0.0.1 is TCP/IP loop-back address). The default port is 42424. we can configure to any port but for that we have to manually edit the registry.
Follow these simple steps
In a webfarm make sure you have the same config file in all your web servers.
Also make sure your objects are serializable.

For session state to be maintained across different web servers in the webfarm, the application path of the web-site in the IIS Metabase should be identical in all the web-servers in the webfarm.

Which template must you provide, in order to display data in a Repeater control?
You have to use the ItemTemplate to Display data. Syntax is as follows,

```
<ItemTemplate>
<div class =”rItem”>
<img src=”images/<%# Container.DataItem(“ImageURL”)%>”  hspace=”10” />
<% # Container.DataItem(“Title”)%>
</div>
</ItemTemplate>
```

What method do you use to explicitly kill a users session?
You can dump (Kill) the session yourself by calling the method Session.Abandon.

ASP.NET automatically deletes a user's Session object, dumping its contents, after it has been idle for a configurable timeout interval. This interval, in minutes, is set in the <SESSIONSTATE>section of the web.config file. The default is 20 minutes.

How do you turn off cookies for one page in your site?
Use Cookie.Discard property, Gets or sets the discard flag set by the server. When true, this property instructs the client application not to save the Cookie on the user's hard disk when a session ends.

Which two properties are on every validation control?
We have two common properties for every validation controls
Control to Validate,
Error Message.

What tags do you need to add within the asp:datagrid tags to bind columns manually?

```
<asp:DataGrid id="dgCart" AutoGenerateColumns="False" CellPadding="4" Width="448px"
runat="server">
<Columns>
<asp:ButtonColumn HeaderText="SELECT" Text="SELECT"
CommandName="select"></asp:ButtonColumn>
<asp:BoundColumn DataField="ProductId" HeaderText="Product ID"></asp:BoundColumn>
<asp:BoundColumn DataField="ProductName" HeaderText="Product Name"></asp:BoundColumn>
<asp:BoundColumn DataField="UnitPrice" HeaderText="UnitPrice"></asp:BoundColumn>
</Columns>
</asp:DataGrid>
```

How do you register JavaScript for webcontrols ?
You can register javascript for controls using
<CONTROL-name>Attribtues.Add(scriptname,scripttext) method.

Which property on a Combo Box do you set with a column name, prior to setting the DataSource, to display data in the combo box?
DataTextField and DataValueField

Which control would you use if you needed to make sure the values in two different controls matched?
CompareValidator is used to ensure that two fields are identical.

What is validationsummary server control? where it is used?.
The ValidationSummary control allows you to summarize the error messages from all validation controls on a Web page in a single location. The summary can be displayed as a list, a bulleted list, or a single paragraph, based on the value of the DisplayMode property. The error message displayed in the

ValidationSummary control for each validation control on the page is specified by the ErrorMessage property of each validation control. If the ErrorMessage property of the validation control is not set, no error message is displayed in the ValidationSummary control for that validation control. You can also specify a custom title in the heading section of the ValidationSummary control by setting the HeaderText property.
You can control whether the ValidationSummary control is displayed or hidden by setting the ShowSummary property. The summary can also be displayed in a message box by setting the ShowMessageBox property to true.

What are the various ways of securing a web site that could prevent from hacking etc..?
Authentication/Authorization
Encryption/Decryption
Maintaining web servers outside the corporate firewall. etc.,

What is the difference between in-proc and out-of-proc?
An inproc is one which runs in the same process area as that of the client giving tha advantage of speed but the disadvantage of stability because if it crashes it takes the client application also with it.Outproc is one which works outside the clients memory thus giving stability to the client, but we have to compromise a bit on speed.

What does aspnet_regiis -i do ?
Aspnet_regiis.exe is The ASP.NET IIS Registration tool allows an administrator or installation program to easily update the script maps for an ASP.NET application to point to the ASP.NET ISAPI version associated with the tool. The tool can also be used to display the status of all installed versions of ASP. NET, register the ASP.NET version coupled with the tool, create client-script directories, and perform other configuration operations.

When multiple versions of the .NET Framework are executing side-by-side on a single computer, the ASP.NET ISAPI version mapped to an ASP.NET application determines which version of the common language runtime is used for the application.

The tool can be launched with a set of optional parameters. Option "i" Installs the version of ASP.NET associated with Aspnet_regiis.exe and updates the script maps at the IIS metabase root and below. Note that only applications that are currently mapped to an earlier version of ASP.NET are affected

Name and describe some HTTP Status Codes and what they express to the requesting client.
The status code can indicate whether a particular request is successful or unsuccessful and can also reveal the exact reason why a request is unsuccessful.
There are 5 groups ranging from 1xx - 5xx of http status codes exists.
- Switching protocols.
- OK. The client request has succeeded
- Object moved.
- Bad request.
500.13 - Web server is too busy.

Can you create an app domain?
Yes, We can create user app domain by calling on of the following overload static methods of the System.AppDomain class
Public static AppDomain CreateDomain(String friendlyName)
Public static AppDomain CreateDomain(String friendlyName, Evidence securityInfo)
Public static AppDomain CreateDomain(String friendlyName, Evidence securityInfo, AppDomainSetup info)
Public static AppDomain CreateDomain(String friendlyName, Evidence securityInfo, String appBasePath, String appRelativeSearchPath, bool shadowCopyFiles)

What property must you set, and what method must you call in your code, in order to bind the data from some data source to the Repeater control?
You must set the DataSource property and call the DataBind method.

Does C# support multiple-inheritance?
No, use interfaces instead.

What does the keyword "virtual" declare for a method or property?
The method or property can be overridden.

What is SOA?
Service Oriented Architecture. In SOA you create an abstract layer that your applications use to access various "services" and can aggregate the services. These services could be databases, web services, message queues or other sources. The Service Layer provides a way to access these services that the applications do not need to know how the access is done. For example, to get a full customer record, I might need to get data from a SGL Server database, a web service and a message queue. The Service layer hides this from the calling application. All the application knows is that it asked for a full customer record. It doesn't know what system or systems it came from or how it was retrieved.

Can you explain some differences between an ADO.NET Dataset and an ADO Recordset? (Or describe some features of a Dataset).
A DataSet can represent an entire relational database in memory, complete with tables, relations, and views. A DataSet is designed to work without any continuing connection to the original data source. Data in a DataSet is bulk-loaded, rather than being loaded on demand. There's no concept of cursor types in a DataSet. DataSets have no current record pointer You can use For Each loops to move through the data. You can store many edits in a DataSet, and write them to the original data source in a single operation. Though the DataSet is universal, other objects in ADO.NET come in different versions for different data sources

Name some of the Microsoft Application Blocks. Have you used any? Which ones?
Examples:
Exception Management
Logging
Data Access
User Interface
Caching Application Block for .NET
Asynchronous Invocation Application Block for .NET
Configuration Management Application Block for .NET
(there are others) We use Exception and Data Access

What is strong name?
A name that consists of an assembly's identity—its simple text name, version number, and culture information (if provided)—strengthened by a public key and a digital signature generated over the assembly.

What are object pooling and connection pooling and difference? Where do we set the Min and Max Pool size for connection pooling?
Object pooling is a COM+ service that enables you to reduce the overhead of creating each object from scratch. When an object is activated, it is pulled from the pool. When the object is deactivated, it is placed back into the pool to await the next request. You can configure object pooling by applying the ObjectPoolingAttribute attribute to a class that derives from the System.EnterpriseServices.ServicedComponent class.
Object pooling lets you control the number of connections you use, as opposed to connection pooling, where you control the maximum number reached.
Following are important differences between object pooling and connection pooling:

Creation. When using connection pooling, creation is on the same thread, so if there is nothing in the pool, a connection is created on your behalf. With object pooling, the pool might decide to create a new object. However, if you have already reached your maximum, it instead gives you the next available object. This is crucial behavior when it takes a long time to create an object, but you do not use it for very long.

Enforcement of minimums and maximums. This is not done in connection pooling. The maximum value in object pooling is very important when trying to scale your application. You might need to multiplex thousands of requests to just a few objects. (TPC/C benchmarks rely on this.)
COM+ object pooling is identical to what is used in .NET Framework managed SQL Client connection pooling. For example, creation is on a different thread and minimums and maximums are enforced.

What is Application Domain?
The primary purpose of the AppDomain is to isolate an application from other applications. Win32 processes provide isolation by having distinct memory address spaces. This is effective, but it is expensive and doesn't scale well. The .NET runtime enforces AppDomain isolation by keeping control over the use of memory - all memory in the AppDomain is managed by the .NET runtime, so the runtime can ensure that AppDomains do not access each other's memory.
Objects in different application domains communicate either by transporting copies of objects across application domain boundaries, or by using a proxy to exchange messages.
MarshalByRefObject is the base class for objects that communicate across application domain boundaries by exchanging messages using a proxy. Objects that do not inherit from **MarshalByRefObject** are implicitly marshal by value. When a remote application references a marshal by value object, a copy of the object is passed across application domain boundaries.

How does an AppDomain get created?
AppDomains are usually created by *hosts*. Examples of hosts are the Windows Shell, ASP.NET and IE. When you run a .NET application from the command-line, the host is the Shell. The Shell creates a new AppDomain for every application.
AppDomains can also be explicitly created by .NET applications. Here is a C# sample which creates an AppDomain, creates an instance of an object inside it, and then executes one of the object's methods.

What is serialization in .NET? What are the ways to control serialization?
Serialization is the process of converting an object into a stream of bytes. Deserialization is the opposite process of creating an object from a stream of bytes. Serialization/Deserialization is mostly used to transport objects (e.g. during remoting), or to persist objects (e.g. to a file or database).Serialization can be defined as the process of storing the state of an object to a storage medium. During this process, the public and private fields of the object and the name of the class, including the assembly containing the class, are converted to a stream of bytes, which is then written to a data stream. When the object is subsequently deserialized, an exact clone of the original object is created.
Binary serialization preserves type fidelity, which is useful for preserving the state of an object between different invocations of an application. For example, you can share an object between different applications by serializing it to the clipboard. You can serialize an object to a stream, disk, memory, over the network, and so forth. Remoting uses serialization to pass objects "by value" from one computer or application domain to another.
XML serialization serializes only public properties and fields and does not preserve type fidelity. This is useful when you want to provide or consume data without restricting the application that uses the data. Because XML is an open standard, it is an attractive choice for sharing data across the Web. SOAP is an open standard, which makes it an attractive choice.

Why do I get errors when I try to serialize a Hashtable?
XmlSerializer will refuse to serialize instances of any class that implements IDictionary, e.g. Hashtable. SoapFormatter and BinaryFormatter do not have this restriction.

SQL Server

Cursors allow row-by-row processing of the resultsets. Types of cursors: Static, Dynamic, Forward-only, Keyset-driven.

Concurrency Problems

Lost Updates occur when two or more transactions select the same row and then update the row based on the value originally selected.

Dirty Read occurs when a second transaction selects a row that is being updated by another transaction. The second transaction is reading data that has not been committed yet and may be changed by the transaction updating the row.

Nonrepeatable Read involves multiple reads (two or more) of the same row and each time the information is changed by another transaction; thus, the term nonrepeatable read.

Phantom Read occur when an insert or delete action is performed against a row that belongs to a range of rows being read by a transaction.

Temporary Tables

Are similar to permanent tables, except temporary tables are stored in **tempdb** and are deleted automatically when no longer in use.

Local Temp Table(#) are visible only to the current connection for the user; and they are deleted when the user disconnects from instances of Microsoft® SQL Server™ 2000.

Global Temp Table(##) are visible to any user after they are created; and they are deleted when all users referencing the table disconnect from SQL Server

ISOLATION LEVELS When locking is used as the concurrency control mechanism, it solves concurrency problems. This allows all transactions to run in complete isolation of one another, although there can be more than one transaction running at any time.

	Dirty Read	NonRepeatable Read	Phantom
Read Uncommitted	Yes	Yes	Yes
Read Committed	No	Yes	Yes
Repeatable Read	No	No	Yes
Serializable	No	No	No

Detect Browser

```
String browser = Context.Request.Browser.Type.ToUpper ();
int version = Context.Request.Browser.MajorVersion;
```

ASPCompat

AspCompat is an aid in migrating ASP pages to ASPX pages. It defaults to false but should be set to true in any ASPX file that creates apartment-threaded COM objects--that is, COM objects registered ThreadingModel=Apartment.

Sending Emails

MailMessage and ***SmtpMail*** are classes defined in the .NET Framework Class Library's ***System.Web.Mail*** namespace.

System.Threading.ReaderWriterLock is the perfect tool to lock the **application cache** during an update

Delete VS Truncate

DELETE TABLE is a logged operation, so the deletion of each row gets logged in the transaction log, which makes it slow. TRUNCATE TABLE also deletes all the rows in a table, but it won't log the deletion

of each row, instead it logs the deallocation of the data pages of the table, which makes it faster. Of course, TRUNCATE TABLE can be rolled back.

ReadOnly VS Const
A const field can only be initialized at the declaration of the field. A readonly field can be initialized either at the declaration or in a constructor. Therefore, readonlyfields can have different values depending on the constructor used. Also, while a const field is a compile-time constant, the readonly field can be used for runtime constants, as in the following example: public static readonly uint l1 = (uint) DateTime.Now.Ticks;

What is Assembly manifest? what all details the assembly manifest will contain.
Every assembly, whether static or dynamic, contains a collection of data that describes how the elements in the assembly relate to each other. The assembly manifest contains this assembly metadata. An assembly manifest contains all the metadata needed to specify the assembly's version requirements and security identity, and all metadata needed to define the scope of the assembly and resolve references to resources and classes. The assembly manifest can be stored in either a PE file (an .exe or .dll) withMicrosoft intermediate language (MSIL) code or in a standalone PE file that contains only assembly manifest information. It contains Assembly name, Version number, Culture, Strong name information, List of all files in the assembly, Type reference information, Information on referenced assemblies.

What are the contents of assembly?
In general, a static assembly can consist of four elements:

- The assembly manifest, which contains assembly metadata.
- Class/Type metadata.
- Microsoft intermediate language (MSIL) code that implements the classes/types.
- A set of resources.

Difference between assembly manifest & metadata assembly manifest
An integral part of every assembly that renders the assembly self-describing. The assembly manifest contains the assembly's metadata. The manifest establishes the assembly identity, specifies the files that make up the assembly implementation, specifies the types and resources that make up the assembly, itemizes the compile-time dependencies on other assemblies, and specifies the set of permissions required for the assembly to run properly. This information is used at run time to resolve references, enforce version binding policy, and validate the integrity of loaded assemblies. The self-describing nature of assemblies also helps makes zero-impact install and XCOPY deployment feasible. metadata - Information that describes every element managed by the common language runtime: an assembly, loadable file, type, method, and so on. This can include information required for debugging and garbage collection, as well as security attributes, marshaling data, extended class and member definitions, version binding, and other information required by the runtime.

What method do you use to explicitly kill a users session?
Abandon()

What is Private Constructor? and it's use? Can you create instance of a class which has Private Constructor?
When a class declares only private instance constructors, it is not possible for classes outside the program to derive from the class or to directly create instances of it. (Except Nested classes)
Make a constructor private if:
You want it to be available only to the class itself. For example, you might have a special constructor used only in the implementation of your class' Clone method.
You do not want instances of your component to be created. For example, you may have a class containing nothing but Shared utility functions, and no instance data. Creating instances of the class would waste memory.

Difference between type constructor and instance constructor? What is static constructor, when it will be fired? And what is its use?
A static constructor is used to initialize a class. It is called automatically to initialize the class before the first instance is created or any static members are referenced.

Class constructors are used for static field initialization. Only one class constructor per type is permitted, and it cannot use the vararg (variable argument) calling convention.

(Class constructor method is also known as type constructor or type initializer)
Instance constructor is executed when a new instance of type is created and the class constructor is executed after the class is loaded and before any one of the class members is accessed. (It will get executed only 1st time, when we call any static methods/fields in the same class.)

Write one code example for compile time binding and one for run time binding? What is early/late binding?
An object is early bound when it is assigned to a variable declared to be of a specific object type. Early bound objects allow the compiler to allocate memory and perform other optimizations before an application executes.
Dim FS As FileStream
FS = New FileStream("C:\tmp.txt", FileMode.Open)
By contrast, an object is late bound when it is assigned to a variable declared to be of type Object. Objects of this type can hold references to any object, but lack many of the advantages of early-bound objects.
Dim xlApp As Object
xlApp = CreateObject("Excel.Application")

In which cases you use override and new base?
Use the new modifier to explicitly hide a member inherited from a base class. To hide an inherited member, declare it in the derived class using the same name, and modify it with the new modifier.

What are ValueTypes
Value types represent primitive types (int, float, . . .). Value types are often allocated on the stack, which means that they can be local variables, parameters, or return values from functions. By default, they are passed by value.

- Primitive ValueTypes
- System.ValueType

What are the reference types:
Reference types combine a location and a sequence of bits. The location provides identity by designating an area in memory where values can be stored and the type of values that can be stored there. Reference types are allocated on the heap, reference types are always accessed through a strongly typed reference rather than directly. Three categories of reference types exist:

- class
- interface
- delegate (pointer types)

What are the built-in reference types:

- object
- string

What is Method overloading?
Method overloading occurs when a class contains two methods with the same name, but different signatures.

What is Method Overriding? How to override a function in C#?
Use the override modifier to modify a method, a property, an indexer, or an event. The overridden base method must be virtual, abstract, or override. You cannot override a non-virtual or static method.

What is the difference between a Struct and a Class?
A struct is a value type, while a class is a reference type.
The struct type is suitable for representing lightweight objects such as Point, Rectangle, and Color. When you create a struct object using the new operator, it gets created and the appropriate constructor is called. Unlike classes, structs can be instantiated without using the new operator. There is no inheritance for structs as there is for classes. A struct cannot inherit from another struct or class, and it cannot be the base of a class. Structs, however, inherit from the base class Object. A struct can implement interfaces.

What is the difference between Array and Arraylist?
As elements are added to an ArrayList, the capacity is automatically increased as required through reallocation. The capacity can be decreased by calling TrimToSize or by setting the Capacity property explicitly.

What is Jagged Arrays?
A jagged array is an array whose elements are arrays. The elements of a jagged array can be of different dimensions and sizes. A jagged array is sometimes called an "array-of-arrays."

What are indexers?
Indexers are similar to properties, except that the get and set accessors of indexers take parameters, while property accessors do not.

What is the difference between ref & out parameters?
An argument passed to a ref parameter must first be initialized. Compare this to an out parameter, whose argument does not have to be explicitly initialized before being passed to an out parameter.
Const VS ReadOnly
Both are meant for constant values. A **const** field can only be initialized at the declaration of the field. A **readonly** field can be initialized either at the declaration or in a constructor. Therefore, **readonly** fields can have different values depending on the constructor used.
while a **const** field is a compile-time constant, the **readonly** field can be used for runtime constants

What are Namespaces?
The **namespace** keyword is used to declare a scope. This namespace scope lets you organize code and gives you a way to create globally-unique types. Even if you do not explicitly declare one, a default namespace is created. This unnamed namespace, sometimes called the global namespace, is present in every file. Any identifier in the global namespace is available for use in a named namespace.
Namespaces implicitly have public access and this is not modifiable.

using directive vs **using** statement

- You create an instance in a **using** statement to ensure that **Dispose** is called on the object when the **using** statement is exited. A **using** statement can be exited either when the end of the **using** statement is reached or if, for example, an exception is thrown and control leaves the statement block before the end of the statement.
- The **using** directive has two uses:
  - Create an alias for a namespace (a **using** alias).
  - Permit the use of types in a namespace, such that, you do not have to qualify the use of a type in that namespace (a **using** directive).

How do I send e-mail from an ASP.NET application?

```
MailMessage message = new MailMessage ();
message.From = <email>;
message.To = <email>;
message.Subject = "Scheduled Power Outage";
message.Body = "Our servers will be down tonight.";
SmtpMail.SmtpServer = "localhost";
SmtpMail.Send (message);
```

MailMessage and SmtpMail are classes defined in the .NET Framework Class Library's System.Web.Mail namespace. Due to a security change made to ASP.NET just before it shipped, you need to set SmtpMail's SmtpServer property to "localhost" even though "localhost" is the default. In addition, you must use the IIS configuration applet to enable localhost (127.0.0.1) to relay messages through the local SMTP service.

When do you set "<IDENTITY impersonate="true" />" ?

Identity is a webconfig declaration under System.web, which helps to control the application Identity of the web applicaton. Which can be at any level(Machine,Site,application,subdirectory,or page), attribute impersonate with "true" as value specifies that client impersonation is used.

JavaScript

Values

JavaScript recognizes the following types of values:

Numbers, such as 42 or 3.14159
Boolean, either true or false
Strings, such as "Howdy!"
null, a special keyword denoting a null value;
null is also a primitive value.
Because JavaScript is case-sensitive, null is not the same as Null, NULL, or any other variant
undefined, a top-level property whose value is undefined;
undefined is also a primitive value.

defining a variable

```
var a = 'hello';
var b = 2;
x = "The answer is " + 42  // returns "The answer is 42"
y = 42 + " is the answer"  // returns "42 is the answer"
"37" - 7   // returns 30
"37" + 7   // returns "377"
```

Declaring Variables

```
You can declare a variable in two ways:
var x = 42 //to declare both local and global variables.
x = 42.    //always declares a global variable and generates a strict JavaScript warning.
           //You shouldn't use this variant.
```

Evaluating Variables

A variable declared using var statement with no initial value specified has the value undefined.
An attempt to access an undeclared variable will result in ReferenceError exception being thrown:

```
var a;                    //a has the value undefined
const c = 3.14;
```