**1. What is the difference between an abstract class and interfaces?**  
Probably "Difference Between abstract Class and Interface" is the most frequent question being asked in the .Net world.  In this tutorial, I will explain the differences theoretically followed by code snippets.  
Theoretically there are basically 5 differences between Abstract Classes and Interfaces as listed below.

1. A class can implement any number of interfaces but a subclass can at most use only one abstract class.
2. An abstract class can have non-abstract methods (concrete methods) whereas all methods of interfaces must be abstract.
3. An abstract class can declare or use any variables whereas an interface is not allowed to do so.  
     
   The following code compiles properly since no field declaration is in the interface.  
     
   interface TestInterface  
   {  
       int x = 4;  // Filed Declaration in Interface  
       void getMethod();  
       string getName();  
   }  
      
   abstract class TestAbstractClass  
   {  
       int i = 4;  
       int k = 3;  
       public abstract void getClassName();  
   }  
     
   It will generate a compile time error as:  
     
   **Error1**- Interfaces cannot contain fields.  
     
   So we need to omit the Field Declaration to compile the code properly.  
     
   interface TestInterface  
   {  
       void getMethod();  
       string getName();  
   }   
   abstract class TestAbstractClass  
   {  
       int i = 4;  
       int k = 3;  
       public abstract void getClassName();  
   }
4. An abstract class can have a constructor declaration whereas an interface cannot.  
     
   So the following code will not compile:  
     
   interface TestInterface  
   {  
       // Constructor Declaration  
       public TestInterface()  
       {  
       }  
       void getMethod();  
       string getName();  
   }  
      
   abstract class TestAbstractClass  
   {  
       public TestAbstractClass()  
       {  
       }  
       int i = 4;  
       int k = 3;  
       public abstract void getClassName();  
   }  
     
   The code above will generate the compile time error:  
     
   **Error 1**-Interfaces cannot contain constructors    
     
   So we need to omit the constructor declaration from the interface to compile our code.  
     
   The following code compiles perfectly:  
     
   interface TestInterface  
   {  
       void getMethod();  
       string getName();  
   }  
      
   abstract class TestAbstractClass  
   {  
       public TestAbstractClass()  
       {  
       }  
       int i = 4;  
       int k = 3;  
       public abstract void getClassName();  
   }
5. An abstract class is allowed to have all access modifiers for all of its member declarations whereas in an interface we cannot declare any access modifier (including public) since all the members of an interface are implicitly public.    
     
   **Note**: here I am talking about the access specifiers of the member of interfaces and not about the interface.  
     
   The following code will explain it better.  
     
   It is perfectly legal to provide an access specifier as Public (remember only public is allowed).  
     
   public interface TestInterface  
   {  
       void getMethod();  
       string getName();  
   }  
     
   The code above compiles perfectly.  
     
   It is not allowed to provide any access specifier to the members of the interface.  
     
   interface TestInterface  
   {  
       public void getMethod();  
       public string getName();  
   }  
     
   The code above will generate the compile time error:  
     
   **Error 1:** The modifier 'public' is not valid for this item.  
     
   But the best way of declaring an interface will be to avoid access specifiers on interfaces as well as members of interfaces.  
     
   interface Test  
   {  
       void getMethod();  
       string getName();  
   }

**2. What is the difference between overriding and overloading?**  
**Overloading**

* In this approach we can define multiple methods with the same name changing their signature. In other words different parameters
* This can be performed within a class as well as within a child class
* Doesn't require any permission from the parent for overloading its method in the child

**Overriding**

* It is an approach of defining multiple methods with the same name and the same signature
* This can be performed only under child classes
* Requires an explicit permission from the parent to override its methods in the child

**3. String Builder and String class**   
**String**   
A string is a sequential collection of Unicode characters that represent text. String is a class that belongs to the namespace System.String.   
   
String concatenation can be done using the '+' opearator or the String.Concat method.  
   
The String.Concat method concatenates one or more instances of a String.  
   
**Sample code**   
string string1 = "Today is " + DateTime.Now.ToString ("D") + ".";  
Console.WriteLine (string1);  
   
string string2 = "Hi " + "This is Jitendra ";  
string2 += "SampathiRao.";  
Console.WriteLine(string2);  
   
**StringBuilder**  
  
StringBuilder is a class that belongs to the namespace System.Text. This class cannot be inherited.  
   
In StringBuilder we use the Append () method.  
   
**Sample code**  
   
StringBuilder number = new StringBuilder (10000);  
for (int i = 0; i<1000; i++)  
{  
returnNumber.Append (i.ToString ());  
}  
  
So where do we use these classes?  
  
The answer is for simple String manipulations we can use the String class. But when there are more string manipulations it is better to use the StringBuilder class.   
   
Why is the StringBuilder class better for more string manipulations instead of the String class?  
  
The String object is immutable. Every time you use one of the methods in the System.String class, you create a new string object in memory, that requires a new allocation of space for that new object. In situations where you need to perform repeated modifications to a string, the overhead associated with creating a new String object can be costly. The performance of the application might be degraded. So we use StringBuilder in such cases.  
  
A StringBuilder object is mutable. The System.Text.StringBuilder class can be used when you want to modify a string without creating a new object. For example, using the StringBuilder class can boost performance when concatenating many strings together in a loop.  
  
**Differences between String and StringBuilder**

|  |  |
| --- | --- |
| It belongs to String namespace | It belongs to String.Text namespace |
| String object is immutable | StringBuilder object is mutable |
| Assigning: StringBuilder sbuild= new StringBuilder("something important"); | Assigning: String s= "something important"; |
| We can use the '+' operator or Concat method to concatenate the strings. | We use the Append method. |
| When string concatenation is done, additional memory will be allocated. | Here additional memory will only be allocated when the string buffer capacity is exceeded. |

**4. What is the difference  between array and array list?**   
Mirosoft.Net has many namespaces for many purposes. Among them the System.Collections is a very important namespace in the programmer's perceptive. While coding, we look for classes that can reduce manual operation. For instance if you want to sort the values in an array then you need to do the manual operations. Here obviously we look for the classes that can automate the sorting by just calling a method. In other words we can call this namespace as a utility namespace. Let us see the classes in this namespace.  
  
**Collection Classes**  
  
The System.Collections namespace has many classes for the individual purpose. We will discuss the frequently used classes in this article.

* ArrayList
* BitArray
* Stack
* Queue
* Comparer
* HashTable
* SortedList

**ArrayList**  
The ArrayList is one of the important classes in the System.Collection namespace. We can say it is the next generation of the Array in C#.  
   
ArrayList is a dynamic array; it will increase the size of the storage location as required. It stores the value as an object. The allocation of the Array List can be done through the Trim To Size property.  
   
**Methods**

|  |  |  |
| --- | --- | --- |
| 1 | Add | It will add the element as object in the ArrayList |
| 2 | AddRange | It will add the collections of elements in the object as individual objects in the ArrayList |
| 3 | Clear | It will clear the all objects in the ArrayList |
| 4 | BinarySearch | It will return the position of the search object as integer value. |
| 5 | Insert | It will insert the element in the specified location of the index in the ArrayList. |
| 6 | InsertRange | It will insert the elements in the object as individual objects in the specified location. |
| 7 | Remove | It will remove the given object in the first occurrence in the ArrayList. |
| 8 | RemoveAt | It will remove the object as specified in the argument. |
| 9 | RemoveRange | It will remove the set of objects in the ArrayList from the range specified. |
| 10 | Sort | It will sort the elements in ascending order. |
| 11 | Reverse | It will arrange the elements in the reverse order in the ArrayList. |
| 12 | GetEnumerator | It will return the collection of objects in the ArrayList as an enumerator. |
| 13 | Contains | It checks whether the objects exist. |

**Properties in the ArrayList**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Property Name** | **Description** |
| 1 | Capcity | This property sets or gets the size of the ArrayList. As you know it will increase the size of the storage as much as required. The default size will be 16. |
| 2 | Count | It returns the total number of elements in the ArrayList. |
| 3 | IsFixedSize | It returns whether the ArrayList is fixed in size. It returns a Boolean value. |
| 4 | IsReadOnly | It returns whether the ArrayList is Readyonly. It returns a Boolean value. |

**Advantages**

* An ArrayList is not a specific data type storage location, it stores everything as an object.
* No need to do allocation and deallocation explicitly to store the data.
* It has explicit sorting methods.
* It can insert and delete the elements between positions in the ArrayList.
* It can store the object as elements.

**Disadvantages**

* ArrayList is not strongly typed. Type casting is necessary when retrieving the content. When we do the type casting every time, it affects performance.
* It uses the LinkedList storage approach, because if you insert or delete the specific position then it must adjust forward/backward in the storage address.
* Sometimes it leads to a runtime error. Consider an example in which we store the ids of the employee in the arraylist and then we want to retrieve the element for some other operations. Obviously we need to do type casting and when there is a string element then what will happen? It will throw the error.

**Differences between Array and ArrayList**

|  |  |
| --- | --- |
| **Array** | **ArrayList** |
| Array uses the Vector array to store the elements | ArrayList uses the LinkedList to store the elements. |
| Size of the Array must be defined until redim is used (vb) | No need to specify the storage size. |
| Array is a specific data type storage | ArrayList can store everything as an object. |
| No need to do the type casting | Type casting must be done every time. |
| It will not lead to Runtime exception | It leads to a run time error exception. |
| Element cannot be inserted or deleted in between. | Elements can be inserted and deleted. |
| There are no built-in members to do ascending or descending. | An ArrayList has many methods to do operations like Sort, Insert, Remove, BinarySeacrh and so on. |

**Conclusion**  
So far we have seen the ArrayList and its members and properties. I hope that this has provided enough of a practical idea about ArrayList. Next we will discuss the BitArray in the same collection class. If you have any queries or further clarifications about ArrayList then please free to post your feedback and corrections.  
  
**5. What are cursors and constraints ?**  
**Cursors**  
A cursor is a database object that helps in accessing and manipulating data in a given result set.The main advantage of cursors is that you can process data row-by-row.  
  
A result set is defined as a collection of rows obtained from a SELECT statement that meet the criteria specified in the WHERE clause.  
  
Cursors, therefore, serve as a mechanism for applications to operate on a single row or a set of rows. Cursors enable the processing of rows in the given result set in the following ways:

1. Allow specific rows to be retrieved from the result set.
2. Allow the current row in the result set to be modified.
3. Help navigate from the current row in the result set to a different row.
4. Allow data modified by other users to be visible in the result set.

* Declare the cursor
* Initialize the cursor
* Open the cursor
* Fetch each row until the status is 0
* close the cursor
* deallocate the cursor

/\* Create two variables that would store the values returned by the fetch statement \*/  
  
DECLARE @DepartmentName char(25)  
DECLARE @DepartmentHead char(25)  
  
/\* Define the cursor that can be used to access the records of the table,row by row \*/  
  
DECLARE curDepartment cursor for  
SELECT vDepartmentName,vDepartmentHead from Department  
  
-- Open the cursor  
  
OPEN curDepartment  
  
-- Fetch the rows into variables  
  
FETCH curDepartment into @DepartmentName,@DepartmentHead  
  
--Start a loop to display all the rows of the cursor  
  
WHILE(@@fetch\_status=0)  
BEGIN  
Print 'Department Name =' + @DepartmentName  
Print 'Department Head =' + @DepartmentHead  
  
--Fetch the next row from the cursor  
FETCH curDepartment into @DepartmentName,@DepartmentHead  
END  
-- Close the cursor  
CLOSE curDepartment  
-- Deallocate the cursor  
DEALLOCATE curDepartment  
   
The following are various types of cursors available in [SQL Server 2005](http://www.netsqlinterviewquestions.com/SQL-Server_Interview_Questions/264_What-are-the-different-Cursor-Types.aspx?TopicID=4):

1. Base table
2. Static
3. Forward-only
4. Forward-only/Read-only
5. Keyset-driven

**Base table**: Base table cursors are the lowest level of cursor available. Base table cursors can scroll forward or backward with minimal cost, and can be updated  
**Static**: Cursors can move to any record but the changes on the data can't be seen.  
**Dynamic**: Most resource-intensive. Cursors can move anywhere and all the changes on the data can be viewed.  
**Forward-only:** The cursor moves forward but can't move backward.  
**Keyset-driven:** Only updated data can be viewed, deleted and inserted data cannot be viewed.  
  
**Constraint**  
**SQL Constraints**  
SQL constraints are used to specify rules for the data in a table.  
  
If there is any violation between the constraint and the data action, the action is aborted by the constraint.  
  
Constraints can be specified when the table is created (inside the CREATE TABLE statement) or after the table is created (inside the ALTER TABLE statement).  
  
SQL CREATE TABLE + CONSTRAINT Syntax:  
  
CREATE TABLE table\_name  
(  
column\_name1 data\_type(size) constraint\_name,  
column\_name2 data\_type(size) constraint\_name,  
column\_name3 data\_type(size) constraint\_name,  
....  
);  
  
 In SQL, we have the following constraints:

* NOT NULL: Indicates that a column cannot store a NULL value
* UNIQUE: Ensures that each row for a column must have a unique value
* PRIMARY KEY: A combination of a NOT NULL and UNIQUE. Ensures that a column (or combination of two or more columns) have a unique identity that helps to find a specific record in a table more easily and quickly
* FOREIGN KEY: Ensures the referential integrity of the data in one table to match values in another table
* CHECK: Ensures that the value in a column meets a specific condition
* DEFAULT: Specifies a default value when none is specified for this column

**6. What are the differences among foreign, primary, and unique keys**  
While unique and primary keys both enforce uniqueness on the column(s) of one table, foreign keys define a relationship between two tables. A foreign key identifies a column or group of columns in one (referencing) table that refers to a column or group of columns in another (referenced) table; in our example above, the Employee table is the referenced table and the Employee Salary table is the referencing table.  
  
**7. Difference Between SCOPE\_IDENTITY() and @@IDENTITY**   
@@IDENTITY: Returns the last identity values that were generated in any table in the current session. @@IDENTITY is not limited to a specific scope.  
SCOPE\_IDENTITY(): Returns the last identity values that are generated in any table in the current session. SCOPE\_IDENTITY returns values inserted only within the current scope.  
   
**8. Delegates and Events**  
The delegate topic seems to be confusing and tough for most developers. This article explains the basics of delegates and event handling in C# in a simple manner.  
   
A delegate is one of the base types in .NET. Delegate is a class to create delegates at runtime.  
   
A delegate in C# is similar to a function pointer in C or C++. It's a new type of object in C#. A delegate is a very special type of object since earlier the entire the object was used to defined contained data but a delegate just contains the details of a method.  
   
**The need for delegates**   
There might be a situation in which you want to pass methods around to other methods. For this purpose we create delegates.  
   
A delegate is a class that encapsulates a method signature. Although it can be used in any context, it often serves as the basis for the event-handling model in C# but can be used in a context removed from event handling (for example passing a method to a method through a delegate parameter).  
   
One good way of understanding delegates is by thinking of a delegate as something that gives a name to a method signature.  
   
**Example**  
  
public delegate int DelegateMethod(int x, int y);   
  
Any method that matches the delegate's signature, that consists of the return type and parameters, can be assigned to the delegate. This makes is possible to programmatically change method calls, and also plug new code into existing classes. As long as you know the delegate's signature, you can assign your own-delegated method.  
   
This ability to refer to a method as a parameter makes delegates ideal for defining callback methods.  
   
**Delegate magic**   
In a class we create its object, that is an instance, but in a delegate when we create an instance it is also referred to as a delegate (in other words whatever you do, you will get a delegate).  
   
A delegate does not know or care about the class of the object that it references. Any object will do; all that matters is that the method's argument types and return type match the delegate's. This makes delegates perfectly suited for "anonymous" invocation.  
   
**Benefit of delegates**   
In simple words delegates are object oriented and type-safe and very secure since they ensure that the signature of the method being called is correct. Delegates help in code optimization.  
   
**Types of delegates**

1. Singlecast delegates
2. Multiplecast delegates

A delegate is a class. Any delegate is inherited from the base delegate class of the .NET class library when it is declared. This can be from either of the two classes System.Delegate or System.MulticastDelegate.  
   
**Singlecast delegate**   
a Singlecast delegate points to a single method at a time. In this the delegate is assigned to a single method at a time. They are derived from the System.Delegate class.  
   
**Multicast Delegate**   
When a delegate is wrapped with more than one method then that is known as a multicast delegate.  
  
In C#, delegates are multicast, which means that they can point to more than one function at a time. They are derived from the System.MulticastDelegate class.  
   
There are three steps in defining and using delegates:  
   
**1. Declaration**   
To create a delegate, you use the delegate keyword.  
   
[attributes] [modifiers] delegate ReturnType Name ([formal-parameters]);  
   
The attributes factor can be a normal C# attribute.  
   
The modifier can be one, or an appropriate combination, of the following keywords: new, public, private, protected, or internal.  
   
The ReturnType can be any of the data types we have used so far. It can also be a type void or the name of a class.  
   
"Name" must be a valid C# name.  
   
Because a delegate is some type of a template for a method, you must use parentheses, required for every method. If this method will not take any arguments then leave the parentheses empty.  
   
**Example**  
   
public delegate void DelegateExample();  
   
The code piece defines a delegate DelegateExample() that has a void return type and accepts no parameters.  
   
**2. Instantiation**   
DelegateExample d1 = new DelegateExample(Display);  
   
The preceding code piece shows how the delegate is initiated.  
   
**3. Invocation**   
d1();  
   
The preceding code piece invokes the delegate d1().  
   
The following is a sample to demonstrate a Singlecast delegate:

using System;

namespace ConsoleApplication5

{

    class Program

    {

        public delegate void delmethod();

        public class P

        {

            public static void display()

            {

                Console.WriteLine("Hello!");

            }

            public static void show()

            {

                Console.WriteLine("Hi!");

            }

            public void print()

            {

                Console.WriteLine("Print");

            }

        }

        static void Main(string[] args)

        {

            // here we have assigned static method show() of class P to delegate delmethod()

            delmethod del1 = P.show;

            // here we have assigned static method display() of class P to delegate delmethod() using new operator

            // you can use both ways to assign the delagate

            delmethod del2 = new delmethod(P.display);

            P obj = new P();

            // here first we have create instance of class P and assigned the method print() to the delegate i.e. delegate with class

            delmethod del3 = obj.print;

            del1();

            del2();

            del3();

            Console.ReadLine();

        }

    }

}  
   
The following is a sample to demonstrate a Multicast delegate:

using System;

namespace delegate\_Example4

{

    class Program

    {

        public delegate void delmethod(int x, int y);

        public class TestMultipleDelegate

        {

            public void plus\_Method1(int x, int y)

            {

                Console.Write("You are in plus\_Method");

                Console.WriteLine(x + y);

            }

            public void subtract\_Method2(int x, int y)

            {

                Console.Write("You are in subtract\_Method");

                Console.WriteLine(x - y);

            }

        }

        static void Main(string[] args)

        {

            TestMultipleDelegate obj = new TestMultipleDelegate();

            delmethod del = new delmethod(obj.plus\_Method1);

            // Here we have multicast

            del += new delmethod(obj.subtract\_Method2);

            // plus\_Method1 and subtract\_Method2 are called

            del(50, 10);

            Console.WriteLine();

            //Here again we have multicast

            del -= new delmethod(obj.plus\_Method1);

            //Only subtract\_Method2 is called

            del(20, 10);

            Console.ReadLine();

        }

    }

}

The following are points to remember about delegates:

* Delegates are similar to C++ function pointers, but are type safe.
* Delegate gives a name to a method signature.
* Delegates allow methods to be passed as parameters.
* Delegates can be used to define callback methods.
* Delegates can be chained together; for example, multiple methods can be called on a single event.
* C# version 2.0 introduces the concept of Anonymous Methods, that permit code blocks to be passed as parameters in place of a separately defined method.
* Delegates helps in code optimization.

**Usage areas of delegates**

* The most common example of using delegates is in events.
* They are extensively used in threading.
* Delegates are also used for generic class libraries, that have generic functionality, defined.

**An Anonymous Delegate**   
You can create a delegate, but there is no need to declare the method associated with it. You do not need to explicitly define a method prior to using the delegate. Such a method is referred to as anonymous.  
   
In other words, if a delegate itself contains its method definition then it is known as an anonymous method.  
   
The following is a sample to show an anonymous delegate:

using System;

public delegate void Test();

public class Program

{

    static int Main()

    {

        Test Display = delegate()

        {

            Console.WriteLine("Anonymous Delegate method");

        };

        Display();

        return 0;

    }

}  
  
**Note:** You can also handle events by anonymous methods.  
   
**Events**   
Events and delegates are linked together.  
   
An event is a reference of a delegate, in other words when an event is raised a delegate will be called.  
   
In C# terms, events are a special form of delegates. Events are nothing but a change of state. Events play an important part in GUI programming. Events and delegates work hand-in-hand to provide a program's functionality.  
   
A C# event is a class member that is activated whenever the event it was designed for occurs.  
   
It starts with a class that declares an event. Any class, including the same class that the event is declared in, may register one of its methods for the event. This occurs through a delegate, that specifies the signature of the method that is registered for the event. The event keyword is a delegate modifier. It must always be used in connection with a delegate.  
   
The delegate may be one of the pre-defined .NET delegates or one you declare yourself. Whichever is appropriate, you assign the delegate to the event, that effectively registers the method that will be called when the event fires.  
   
**How to use events?**   
Once an event is declared, it must be associated with one or more event handlers before it can be raised. An event handler is nothing but a method that is called using a delegate. Use the += operator to associate an event with an instance of a delegate that already exists.  
   
The following is an example:  
   
obj.MyEvent += new MyDelegate(obj.Display);  
   
An event has the value null if it has no registered listeners.  
   
Although events are most commonly used in Windows Controls programming, they can also be implemented in console, web and other applications.  
   
The following is a sample of creating a custom Singlecast delegate and event:

using System;

namespace delegate\_custom

{

    class Program

    {

        public delegate void MyDelegate(int a);

        public class XX

        {

            public event MyDelegate MyEvent;

            public void RaiseEvent()

            {

                MyEvent(20);

                Console.WriteLine("Event Raised");

            }

            public void Display(int x)

            {

                Console.WriteLine("Display Method {0}", x);

            }

        }

        static void Main(string[] args)

        {

            XX obj = new XX();

            obj.MyEvent += new MyDelegate(obj.Display);

            obj.RaiseEvent();

            Console.ReadLine();

        }

    }

}  
  
**9. ASP.NET page cycle?**Each request for an .aspx page that hits IIS is handed over to the HTTP Pipeline. The HTTP Pipeline is a chain of managed objects that sequentially process the request and convert it to plain HTML text content. The starting point of a HTTP Pipeline is the HttpRuntime class. The ASP.NET infrastructure creates each instance of this class per AppDomain hosted within the worker process. The HttpRuntime class picks up an HttpApplication object from an internal pool and sets it to work on the request. It determines what class handles the request. The association between the resources and handlers are stored in the configurable file of the application. In web.config and also inmachine.config you will find these lines in section.  
  
If you run through the following program then it will be much easier to follow:  
  
<add verb="\*" path="\*.aspx" type="System.Web.UI.PageHandlerFactory"/>  
  
This extension can be associated with HandlerClass or HandlerFactory class. HttpApplicationobject gets the page object that implements the IHttpHandler Interface. The process of generating the output to the browser is started when the object calls the ProcessRequest method.  
  
**Page Life Cycle**  
Once the HTTP page handler class is fully identified, the ASP.NET runtime calls the handler's ProcessRequest to start the process. This implementation begins by calling the methodFrameworkInitialize(), that builds the control trees for the page. This is a protected and virtual member of the TemplateControl class, the class from which the page itself derives.  
  
Next processRequest() makes the page transition through various phases: initialization, the loading of viewstate and postback data, the loading of the page's user code and the execution of postback server-side events. Then the page enters into render mode, the viewstate is updated and the HTML is generated and sent to the output console. Finally the page is unloaded and the request is considered completely served.  
  
Stages and corresponding events in the life cycle of the ASP.NET page cycle:

|  |  |
| --- | --- |
| **Stage** | **Events/Method** |
| Page Initialization | Page\_Init |
| View State Loading | LoadViewState |
| Postback data processing | LoadPostData |
| Page Loading | Page\_Load |
| PostBack Change Notification | RaisePostDataChangedEvent |
| PostBack Event Handling | RaisePostBackEvent |
| Page Pre Rendering Phase | Page\_PreRender |
| View State Saving | SaveViewState |
| Page\_Render | Page Rendering |
| Page\_UnLoad | Page Unloading |

Some of the events listed above are not visible at the page level. It will be visible if you happen to write server controls and write a class that is derived from page.  
  
**1. PreInit**   
The entry point of the page life cycle is the pre-initialization phase called "PreInit". This is the only event where programmatic access to master pages and themes is allowed.  You can dynamically set the values of master pages and themes in this event.  You can also dynamically create controls in this event.  
   
**Example**: Override the event as given below in your code-behind cs file of your aspx page.

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.UI;

using System.Web.UI.WebControls;

public partial class \_Default : System.Web.UI.Page

{

    protected void Page\_PreInit(object sender, EventArgs e)

    {

        //  Use this event for the following:

        //  Check the IsPostBack property to determine whether this is the first time the page is being processed.

        //  Create or re-create dynamic controls.

        //  Set a master page dynamically.

        //  Set the Theme property dynamically.

    }

**2. Init**   
This event fires after each control has been initialized, each control's UniqueID is set and any skin settings have been applied. You can use this event to change initialization values for controls. The "Init" event is fired first for the most bottom control in the hierarchy, and then fired up the hierarchy until it is fired for the page itself.   
   
**Example :** Override the event as given below in your code-behind cs file of your aspx page.  
  
protected void Page\_Init(object sender, EventArgs e)  
{  
    // Raised after all controls have been initialized and any skin settings have been applied.   
    // Use this event to read or initialize control properties.  
}  
   
**3. InitComplete**   
Raised once all initializations of the page and its controls have been completed. Until now the viewstate values are not yet loaded, hence you can use this event to make changes to view state that you want to ensure are persisted after the next postback.  
   
**Example :**Override the event as given below in your code-behind cs file of your aspx page.  
  
protected void Page\_InitComplete(object sender, EventArgs e)  
{  
       // Raised by the  Page object. Use this event for processing tasks that require all initialization be complete.   
}  
   
**4. PreLoad**   
Raised after the page loads the view state for itself and all controls and then processes postback data included with the Request instance.  
  
Loads ViewState: ViewState data are loaded to controls   
  
Note: The page viewstate is managed by ASP.NET and persists information over a page roundtrip to the server. Viewstate information is saved as a string of name/value pairs and contains information such as control text or value. The viewstate is held in the value property of a hidden control that is passed from page request to page request.   
    
Loads Postback data: postback data are now handed to the page controls.  
  
Note: During this phase of the page creation, form data that was posted to the server (termed postback data in ASP.NET) is processed against each control that requires it. Hence, the page fires the LoadPostData event and parses through the page to find each control and updates the control state with the correct postback data. ASP.NET updates the correct control by matching the control's unique ID with the name/value pair in the NameValueCollection. This is one reason that ASP.NET requires unique IDs for each control on any given page.  
   
**Example:** Override the event as given below in your code-behind cs file of your aspx page.  
  
protected override void OnPreLoad(EventArgs e)  
{  
        // Use this event if you need to perform processing on your page or control before the  Load event.  
        // Before the Page instance raises this event, it loads view state for itself and all controls, and   
        // then processes any postback data included with the Request instance.  
}  
   
**5. Load**   
The important thing to note about this event is the fact that by now, the page has been restored to its previous state in case of postbacks. Code inside the page load event typically checks for PostBack and then sets control properties appropriately. This method is typically used for most code, since this is the first place in the page lifecycle that all values are restored. Most code checks the value of IsPostBack to avoid unnecessarily resetting state. You may also wish to call Validate and check the value of IsValid in this method. You can also create dynamic controls in this method.  
   
**Example:** Override the event as given below in your code-behind cs file of your aspx page.  
  
protected void Page\_Load(object sender, EventArgs e)  
{  
        // The  Page calls the  OnLoad event method on the  Page, then recursively does the same for each child  
        // control, which does the same for each of its child controls until the page and all controls are loaded.  
        // Use the OnLoad event method to set properties in controls and establish database connections.  
}  
   
**6. Control (PostBack) event(s)**   
ASP.NET now calls any events on the page or its controls that caused the PostBack to occur. This might be a button's click event or a dropdown's selectedindexchange event, for example.  
   
These are the events, the code for which is written in your code-behind class (.cs file).  
   
**Example:** Override the event as given below in your code-behind cs file of your aspx page.  
  
protected void Button1\_Click(object sender, EventArgs e)  
{  
        // This is just an example of control event.. Here it is button click event that caused the postback  
}  
   
**7. LoadComplete**   
This event signals the end of Load.  
   
**Example:** Override the event as given below in your code-behind cs file of your aspx page.  
  
protected void Page\_LoadComplete(object sender, EventArgs e)  
{  
        // Use this event for tasks that require that all other controls on the page be loaded.  
}  
   
**8. PreRender**  
   
Allows final changes to the page or its control. This event takes place after all regular PostBack events have taken place. This event occurs before saving ViewState, so any changes made here are saved.  
   
For example: After this event, you cannot change any property of a button or change any viewstate value. Because, after this event, the SaveStateComplete and Render events are called.  
   
**Example:** Override the event as given below in your code-behind cs file of your aspx page.  
  
protected override void OnPreRender(EventArgs e)  
{  
        // Each data bound control whose DataSourceID property is set calls its DataBind method.  
        // The PreRender event occurs for each control on the page. Use the event to make final   
        // changes to the contents of the page or its controls.  
}  
   
**9. SaveStateComplete**   
Prior to this event the view state for the page and its controls is set.  Any changes to the page's controls at this point or beyond are ignored.   
   
**Example:** Override the event as given below in your code-behind cs file of your aspx page.  
  
protected override void OnSaveStateComplete(EventArgs e)  
{  
        // Before this event occurs,  ViewState has been saved for the page and for all controls.   
        // Any changes to the page or controls at this point will be ignored.  
        // Use this event perform tasks that require view state to be saved, but that do not make any changes to controls.  
}  
   
**10. Render**  
   
This is a method of the page object and its controls (and not an event). At this point, ASP.NET calls this method on each of the page's controls to get its output. The Render method generates the client-side HTML, Dynamic Hypertext Markup Language (DHTML), and script that are necessary to properly display a control at the browser.  
   
**Note**: Right-click on the web page displayed in the client's browser and view the page's source. You will not find any aspx server control in the code. Because all aspx controls are converted to their respective HTML representation. The browser is capable of displaying HTML and client-side scripts.  
   
**Example:** Override the event as given below in your code-behind cs file of your aspx page.  
  
// Render stage goes here. This is not an event  
   
**11. UnLoad**   
This event is used for cleanup code. After the page's HTML is rendered, the objects are disposed of. During this event, you should destroy any objects or references you have created in building the page. At this point, all processing has occurred and it is safe to dispose of any remaining objects, including the Page object.   
   
Cleanup can be performed on:  
   
(a)Instances of classes in other words objects  
(b)Closing opened files  
(c)Closing database connections.  
   
**Example:** Override the event as given below in your code-behind cs file of your aspx page.  
  
protected void Page\_UnLoad(object sender, EventArgs e)  
{  
        // This event occurs for each control and then for the page. In controls, use this event to do final cleanup   
        // for specific controls, such as closing control-specific database connections.  
        // During the unload stage, the page and its controls have been rendered, so you cannot make further   
        // changes to the response stream.  
        //If you attempt to call a method such as the Response.Write method, the page will throw an exception.  
    }  
}  
   
**10. Request and response in IIS**   
When the client requests some information from a web server, the request first reaches to HTTP.SYS of IIS. HTTP.SYS then sends the request to the respective Application Pool. The Application Pool then forwards the request to the worker process to load the ISAPI Extension that will create an HTTPRuntime Object to process the request via HTTPModule and HTTPHanlder. After that the ASP.NET Page LifeCycle events start.  
   
**11. What are ADO.net objects ?**   
ADO.NET is designed to help developers work efficiently with multi tier databases, across intranet or internet scenarios.   
   
The ADO.NET object model consists of two key components as follows:

* Connected model (.NET Data Provider: a set of components including the Connection, Command, DataReader, and DataAdapter objects)
* Disconnected model (DataSet).

**Connection**   
The Connection object is the first component of ADO.NET. The connection object opens a connection to your data source.  
   
All of the configurable aspects of a database connection are represented in the Connection object, that includes ConnectionString and ConnectionTimeout.  
   
Connection object helps in accessing and manipulating a database. Database transactions are also dependent upon the Connection object.  
   
In ADO.NET the type of the Connection is dependent on what database system you are working with. The following are the most commonly used for connections in ADO.NET:

* SqlConnection
* OleDbConnection
* OdbcConnection

**Command**  
   
The Command object is used to perform an action on the data source. The Command object can execute Stored Procedures and T-SQL commands.  
   
You can execute SQL queries to return data in a DataSet or a DataReader object. The Command object performs the standard Select, Insert, Delete and Update T-SQL operations.  
   
**DataReader**   
The DataReader is built as a way to retrieve and examine the rows returned in response to your query as quickly as possible.  
   
No DataSet is created; in fact, no more than one row of information from the data source is in-memory at a time. This makes the DataReader very efficient at returning large amounts of data.  
   
The data returned by a DataReader is always read only.  This class was built to be a lightweight forward only, read only, way to run through data quickly (this was called a Firehose cursor in ADO).  
   
However, if you need to manipulate a schema or use some advanced display features such as automatic paging then you must use a DataAdapter and DataSet.  
   
A DataReader object works in the connected model.  
   
**DataAdapter**   
The DataAdapter takes the results of a database query from a Command object and pushes them into a DataSet using the DataAdapter.Fill() method. Additionally the DataAdapter.Update() method will negotiate any changes to a DataSet back to the original data source.  
   
A DataAdapter object works in a connected model. A DataAdapter performs the following five steps:

1. Create/open the connection
2. Fetch the data as per command specified
3. Generate XML file of data
4. Fill data into DataSet.
5. Close connection.

**Command Builder**   
It is used to save changes made in an in-memory cache of data on the backend. The work of Command Builder is to generate a Command as per changes in DataRows.  
   
Command Builder generates commands on the basis of row state. There are five row states:

1. Unchanged
2. Added
3. Deleted
4. Modified
5. Detached

Command Builder works on add, delete and modified row state only.  
   
Detached is used when an object is not created from row state.  
   
**Transaction**   
The Transaction object is used to execute a backend transaction. Transactions are used to ensure that multiple changes to database rows occur as a single unit of work.   
   
The Connection class has a BeginTransaction method that can be used to create a Transaction.  
   
A definite best practice is to ensure that transactions are placed in Using statements for rapid cleanup if they are not committed.  Otherwise the objects (and any internal locks that may be needed) will remain active until the GC gets around to cleaning it up.  
   
**Parameters**   
A Parameter object is used to solve SQL Injection attack problems while dealing with the user input parameters.  
   
A Parameter object allows passing parameters into a Command object. The Parameter class allows you to quickly put parameters into a query without string concatenation.  
   
Note: See my other article on ADO.NET Objects Part II.  
   
**Conclusion**   
Hope the article has helped you to understand ADO.NET objects.  
   
Your feedback and constructive contributions are welcome.  Please feel free to contact me for feedback or comments you may have about this article. ADO.NET is designed to help developers work efficiently with multi-tier databases, across intranet or Internet scenarios.   
   
The ADO.NET object model consists of two key components as follows:

* Connected model (.NET Data Provider: a set of components including the Connection, Command, DataReader, and DataAdapter objects)
* Disconnected model (DataSet).

**DataSet**   
The DataSet Object is the parent object to most of the other objects in the System.Data namespace. The DataSet is a disconnected, in-memory representation of data.  
   
Its primary role is to store a collection of DataTables and the relations and constraints between those DataTables.  
   
DataSet also contains several methods for reading and writing XML, as well as merging other DataSets, DataTables and DataRows.  
   
Some of the advantages of the new DataSet object are:

* Read / Write
* Connectionless
* Contains one or more DataTable objects with relationships defined in a collection of DataRelation objects
* Supports filtering and sorting
* The contained DataView object can be bound to data-aware forms controls
* Supports automatic XML serialization (creation of XML document)

**DataTable**   
DataTable stores a table of information, typically retrieved from a data source. DataTable allows you to examine the actual rows of a DataSet through rows and columns collections.  
   
Once the DataTable is filled the database connection is released and operates disconnected only.   
   
You can complex-bind a control to the information contained in a data table. Controls like DataGrid are used for this purpose.  
   
DataTable also stores metatable information such as the primary key and constraints.  
   
**DataRows**   
The DataRow class permits you to reference a specific row of data in a DataTable. This is the class that permits you to edit, accept, or reject changes to the individual DataColumns of the row.   
  
You would use a DataRow object and its properties and methods to retrieve and evaluate the values in a DataTable object.  
   
The DataRowCollection represents the actual DataRow objects that are in the DataTable object, and the DataColumnCollection contains the DataColumn objects that describe the schema of the DataTable object.  
   
**DataColumns**   
DataColumns is the building block of the DataTable. A number of such objects make up a table. Each DataColumn object has a DataType property that determines the kind of data that the column is holding. data table.  
   
DataColumn instances are also the class of object that you use to read and write individual columns of a database. The Items property of a DataRow returns a DataColumn.  
   
**DataView**   
A DataView is an object that provides a custom view of data in a DataTable. DataViews provide sorting, filtering, and other types of customizations.  
   
Each DataTable object has a DefaultView property, that holds the default DataView object for that DataTable. Modifying the DefaultView object sets the default display characteristics for the DataTable.  
   
You can create an instance of a DataView and associate it with a specific DataTable in a DataSet. This permits you to have two or more different views of the same DataTable simultaneously available.  
   
**DataRelation**   
A DataRelation identifies that two or more of the DataTables in a DataSet contain data related in a one-to-one or one-to-many (parent-child) association. You define a relationship between two tables by adding a new DataRelation object to the DataSets.  
   
**Constraints**   
Each DataColumn may have multiple Constraints.  Conditions such as unique are applied through this class.  Constraint objects are maintained through the DataTables Constraints collection.  
   
**DataRowView**   
A DataRowView is a special object that represents a row in a DataView. Each DataView can have a different RowStateFilter, the DataRowView obtained from a DataView will contain data consistent with the DataView's RowStateFilter, providing a custom view of the data.  
  
**12. What is the difference between Stored Procedures and functions?**  
1. Procedures can have input/output parameters for it whereas functions can have only input parameters. 2. Procedure allows select as well as DML statement in it whereas a function only allows select statements in it. 3. We can go for transaction management in procedures whereas we can't for functions. 4. Procedures cannot be utilized in a select statement whereas functions can be embedded in a select statement.

**13. What is the difference between Stored Procedures and triggers?**

* The Stored Procedures are used for performing tasks

Stored Procedures normally used to performing user specified tasks. It can have the parameters. It can return multiple results set.

* The Triggers for auditing work

Triggers normally are used for auditing work. It can be used to trace the activities of the table events.

* The Stored Procedures can have the parameters

Procedures can have the input and output parameters with all the data types available in the SQL Server as well as user defined data types.

* The Triggers cannot have any parameters

Triggers cannot have any parameters.

* Stored Procedure can be run independently

The Stored Procedures can be run independently. They are stored as a database object. It can be called from the application.

* The triggers executes based on table events

The DML triggers are get executed based on the table events defined on the specific table. There are various types of triggers, like DML triggers, DDL triggers (from 2005 onwards) and logon triggers (from 2008 onwards).  
  
The DDL triggers can control the Stored Procedures creation, drop and so on.

* The Stored Procedure cannot call triggers

The Stored Procedures cannot call the triggers directly. But when we do the DML operations on the table, if the corresponding table has the trigger then it will then be triggered.

* The triggers can call Stored Procedures

The triggers can call Stored Procedures.

**14. Difference between DataSet and DataReader?  
  
DataReader**  
DataReader reads data from a database. It only reads using a forward-only connection oriented architecture during the fetching of the data from the database. A DataReader will fetch the data very fast compared with a DataSet. Generally we will use an ExecuteReader object to bind data to a DataReader.  
  
To bind DataReader data to a GridView we need to write the code such as shown below:  
   
Protected void BindGridview()  
{  
using (SqlConnection conn = new SqlConnection("Data Source=abc;Integrated Security=true;Initial Catalog=Test"))  
{  
con.Open();  
SqlCommand cmd = new SqlCommand("Select UserName, First Name,LastName,Location FROM Users", conn);  
SqlDataReader sdr = cmd.ExecuteReader();  
gvUserInfo.DataSource = sdr;  
gvUserInfo.DataBind();  
conn.Close();  
}  
}

* Holds the connection open until you are finished (don't forget to close it!).
* Can typically only be iterated over once
* Is not as useful for updating back to the database

**DataSet**  
DataSet is a disconnected oriented architecture. That means that there is no need for active connections to work with DataSets and it is a collection of DataTables and relations between tables. It holds multiple tables with data. You can select data form tables, create views based on tables and ask child rows over relations. Also DataSet provides you with rich features like saving data as XML and loading XML data.  
  
protected void BindGridview()  
{  
    SqlConnection conn = new SqlConnection("Data Source=abc;Integrated Security=true;Initial Catalog=Test");  
    conn.Open();  
    SqlCommand cmd = new SqlCommand("Select UserName, First Name,LastName,Location FROM Users", conn);  
    SqlDataAdapter sda = new SqlDataAdapter(cmd);  
    DataSet ds = new DataSet();  
    da.Fill(ds);  
    gvUserInfo.DataSource = ds;  
    gvUserInfo.DataBind();  
}  
  
**DataAdapter**  
A DataAdapter will act as a bridge between a DataSet and the database. This DataAdapter object reads the data from the database and binds that data to a DataSet. DataAdapter is a disconnected oriented architecture. Check the following sample code to see how to use a DataAdapter in code:  
  
protected void BindGridview()  
{  
    SqlConnection con = new SqlConnection("Data Source=abc;Integrated Security=true;Initial Catalog=Test");  
    conn.Open();  
    SqlCommand cmd = new SqlCommand("Select UserName, First Name,LastName,Location FROM Users", conn);  
    SqlDataAdapter sda = new SqlDataAdapter(cmd);  
    DataSet ds = new DataSet();  
    da.Fill(ds);  
    gvUserInfo.DataSource = ds;  
    gvUserInfo.DataBind();  
}

* Lets you close the connection as soon it's done loading data, and may even close it for you automatically
* All of the results are available in memory
* You can iterate over it as many times as you need, or even look up a specific record by index
* Has some built-in faculties for updating back to the database.

**DataTable**  
DataTable represents a single table in the database. It has rows and columns. There is not much difference between DataSet and datatable, a DataSet is simply a collection of datatables.  
  
protected void BindGridview()  
{  
     SqlConnection con = new SqlConnection("Data Source=abc;Integrated Security=true;Initial Catalog=Test");  
     conn.Open();  
     SqlCommand cmd = new SqlCommand("Select UserName, First Name,LastName,Location FROM Users", conn);  
     SqlDataAdapter sda = new SqlDataAdapter(cmd);  
     DataTable dt = new DataTable();  
     da.Fill(dt);  
     gridview1.DataSource = dt;  
     gvidview1.DataBind();  
}  
   
**15. Differnce between JavaScript  jQuery and AJAX?**   
**JavaScript**

1. JavaScript is a client-side (in the browser) scripting language.
2. JavaScript lets you supercharge your HTML with animation, interactivity, and dynamic visual effects
3. JavaScript can make web pages more useful by supplying immediate feedback.
4. JavaScript is the common term for a combination of the ECMAScript programming language plus some means for accessing a web browser's windows and the document object model (DOM).
5. JavaScript was designed to add interactivity to HTML pages
6. Everyone can use JavaScript without purchasing a license

**jQuery**

1. jQuery is a library/framework built with JavaScript.
2. It abstracts away cross-browser compatibility issues and it emphasises unobtrusive and callback-driven JavaScript programming
3. jQuery (website) is a JavaScript framework that makes working with the DOM easier by building many high level functionality that can be used to search and interact with the DOM
4. jQuery implements a high-level interface to do AJAX requests
5. jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and AJAX interactions for rapid web development

**Asynchronous JavaScript XML (AJAX)**

1. Asynchronous JavaScript XML (AJAX) is a method to dynamically update parts of the UI without having to reload the page, to make the experience more similar to a Desktop application.
2. AJAX is a technique to do an XMLHttpRequest (out of band Http request) from a web page to the server and send/retrieve data to be used on the web page
3. It uses JavaScript to construct an XMLHttpRequest, typically using various techniques on various browsers.
4. AJAX is a set of functions of the language JavaScript
5. Examples of applications using AJAX: Google Maps, Gmail, Youtube, and Facebook tabs.

**Why AJAX?**   
This is probably one of the most asked questions about AJAX.  
   
The main advantage of using AJAX enabled ASP.NET Web applications is improved efficiency and a reduction of the page refresh time. AJAX enables us to refresh only parts of a Web page that have been updated, rather refreshing the entire page.  
   
For example, if you have four controls on a web page, say a DropDownList, a TextBox, a ListBox, and a GridView. The GridView control shows some data based on the selection in a DropDownList and other controls. Now let's say a GridView also has paging and sorting options. So whenever you move to the next page or apply a sort, the entire page and all four controls on the page will be refreshed and you will notice a page flicker because ASP.NET must render the entire page on the client-side and it happens once.  
   
In an AJAX-enabled web page, you will see only the GridView data is being refreshed and rest of the page and controls do not. Doing so, we not only get better performance and faster refresh, we also get a better (or should I say "smoother") user experience.  
   
You may want to see a live example of an AJAX enabled GridView on our [www.mindcracker.com](http://www.mindcracker.com/) web site in the Jobs section here: <http://www.mindcracker.com/Jobs/> . On this page, if you click on the "Next" page link then you will see only the GridView data is being refreshed. We are also implementing AJAX on C# Corner and other sites as well.  
   
**What Browsers Does AJAX Support?**   
AJAX is JavaScript based and supports most of the browsers including Internet Explorer, Mozilla Firefox, and Apple Safari.  
   
**What are ASP.NET AJAX Extensions?**   
ASP.NET AJAX is a combination of client-script libraries (JavaScript) and ASP.NET server components that are integrated to provide a robust development framework.   
   
**What are ASP.NET AJAX Server Controls?**   
The ASP.NET AJAX server controls consist of server and client code that integrate to produce AJAX-like behavior. The following controls are available in the AJAX 1.0 library:

1. ScriptManager: Manages script resources for client components, partial-page rendering, localization, globalization, and custom user scripts. The ScriptManager control is required for the use of the UpdatePanel, UpdateProgress, and Timer controls.
2. UpdatePanel: Enables you to refresh selected parts of the page instead of refreshing the entire page by using a synchronous postback.
3. UpdateProgress: Provides status information about partial-page updates in UpdatePanel controls.
4. Timer: Performs postbacks at defined intervals. You can use the Timer control to post the entire page, or use it together with the UpdatePanel control to perform partial-page updates at a defined interval.

**What is the ASP.NET AJAX Control Toolkit?**   
The ASP.NET AJAX Control Toolkit is a collection of samples and components that show you some of the experiences you can create with rich client ASP.NET AJAX controls and extenders. The Control Toolkit provides both samples and a powerful SDK to simplify creation and reuse of your custom controls and extenders. You can download the ASP.NET AJAX Control Toolkit from the ASP.NET AJAX Web site.  
  
**16. What is the differnce between WCF and web services?**  
**Web Service in ASP.NET**  
A Web Service is programmable application logic accessible via standard Web protocols. One of these Web protocols is the Simple Object Access Protocol (SOAP). SOAP is a W3C submitted note (as of May 2000) that uses standards based technologies (XML for data description and HTTP for transport) to encode and transmit application data.  
  
Consumers of a Web Service do not need to know anything about the platform, object model, or programming language used to implement the service; they only need to understand how to send and receive SOAP messages (HTTP and XML).  
  
**WCF Service**  
Windows Communication Foundation (WCF) is a framework for building service-oriented applications. Using WCF, you can send data as asynchronous messages from one service endpoint to another. A service endpoint can be a part of a continuously available service hosted by IIS, or it can be a service hosted in an application. An endpoint can be a client of a service that requests data from a service endpoint. The messages can be as simple as a single character or word sent as XML, or as complex as a stream of binary data.  
  
The following are the scenarios that WCF must be used in:

* A secure service to process business transactions.
* A service that supplies current data to others, such as a traffic reports or other monitoring service.
* A chat service that allows two people to communicate or exchange data in real time.
* A dashboard application that polls one or more services for data and presents it in a logical presentation.
* Exposing a workflow implemented using Windows Workflow Foundation as a WCF service.
* A Silverlight application to poll a service for the latest data feeds.

**Features of WCF**

* Service Orientation
* Interoperability
* Multiple Message Patterns
* Service Metadata
* Data Contracts
* Security
* Multiple Transports and Encoding
* Reliable and Queued Messages
* Durable Messages
* Transactions
* AJAX and REST Support
* Extensibility

**Difference between a Web Service in ASP.NET & WCF Service**WCF is a replacement for all earlier web service technologies from Microsoft. It also does much more than what is traditionally considered as "web services".  
  
WCF "web services" are part of a much broader spectrum of remote communication enabled through WCF. You will get a much higher degree of flexibility and portability doing things in WCF than through traditional ASMX because WCF is designed, from the ground up, to summarize all of the various distributed programming infrastructures offered by Microsoft. An endpoint in WCF can be communicated with just as easily over SOAP/XML as it can over TCP/binary and to change this medium is simply a configuration file modification. In theory, this reduces the amount of new code needed when porting or changing business needs, targets, and so on.  
  
ASMX is older than WCF, and anything ASMX can do so can WCF (and more). Basically you can see WCF as trying to logically group together all the various ways of getting two apps to communicate in the world of Microsoft; ASMX was just one of these many ways and so is now grouped under the WCF umbrella of capabilities.  
  
Web Services can be accessed only over HTTP & it works in a stateless environment, whereas WCF is flexible because its services can be hosted in various types of applications. Common scenarios for hosting WCF services are IIS,WAS, Self-hosting and Managed Windows Service.  
  
The major difference is that Web Services use XmlSerializer. But WCF uses DataContractSerializer that is better in performance than XmlSerializer.  
  
Key issues with XmlSerializer to serialize .NET types to XML are:

* Only Public fields or Properties of .NET types can be translated into XML
* Only the classes that implement IEnumerable interface
* Classes that implement the IDictionary interface, such as Hash table cannot be serialized

Some important differences between DataContractSerializer and XMLSerializer are:

* A practical benefit of the design of the DataContractSerializer is better performance overXmlserializer.
* XML Serialization does not indicate which fields or properties of the type are serialized into XML whereasDataCotractSerializer
* Explicitly shows the which fields or properties are serialized into XML
* The DataContractSerializer can translate the HashTable into XML

**Using the Code**The development of a web service with ASP.NET relies on defining data and relies on the XmlSerializer to transform data to or from a service.  
  
Key issues with XmlSerializer to serialize .NET types to XML:

* Only Public fields or Properties of .NET types can be translated into XML
* Only the classes that implement IEnumerable interface
* Classes that implement the IDictionary interface, such as Hash table cannot be serialized

The WCF uses the DataContractAttribute and DataMemeberAttribute to translate .NET FW types into XML.  
  
 Collapse | Copy Code  
[DataContract]   
public class Item   
{   
    [DataMember]   
    public string ItemID;   
    [DataMember]   
    public decimal ItemQuantity;   
    [DataMember]   
    public decimal ItemPrice;  
}  
  
The DataContractAttribute can be applied to the class or a strcture. DataMemberAttribute can be applied to a field or a property and theses fields or properties can be either public or private.  
  
Some important differences between DataContractSerializer and XMLSerializer are:

* A practical benefit of the design of the DataContractSerializer is better performance over XML serialization.
* XML Serialization does not indicate which fields or properties of the type are serialized into XML whereas DataContractSerializer explicitly shows which fields or properties are serialized into XML.
* The DataContractSerializer can translate the HashTable into XML.

**Developing Service**To develop a service using ASP.NET, we must add the WebService attribute to the class and WebMethodAttribute to any of the class methods.  
  
**Example**

[WebService]

public class Service : System.Web.Services.WebService

{

    [WebMethod]

    public string Test(string strMsg)

    {

        return strMsg;

    }

}  
  
To develop a service in WCF, we will write the following code:

   [ServiceContract]

    public interface ITest

    {

        [OperationContract]

        string ShowMessage(string strMsg);

    }

    public class Service : ITest

    {

        public string ShowMessage(string strMsg)

        {

            return strMsg;

        }

    }

The ServiceContractAttribute specifies that an interface defines a WCF service contract,   
  
OperationContract attribute indicates which of the methods of the interface defines the operations of the service contract.  
  
A class that implements the service contract is referred to as a service type in WCF.  
  
**Hosting the Service**ASP.NET web services are compiled into a class library assembly and a service file with an extension .asmx will have the code for the service. The service file is copied into the root of the ASP.NET application and Assembly will be copied to the bin directory. The application is accessible using URL of the service file.  
  
WCF Service can be hosted within IIS or WindowsActivationService as in the following:

* Compile the service type into a class library
* Copy the service file with an extension .SVC into a virtual directory and assembly into bin sub directory of the virtual directory.
* Copy the web.config file into the virtual directory.

**Client Development**Clients for the ASP.NET Web services are generated using the command-line tool WSDL.EXE.  
  
WCF uses the ServiceMetadata tool (svcutil.exe) to generate the client for the service.  
  
**Message Representation**  
The Header of the SOAP Message can be customized in ASP.NET Web service.  
  
WCF provides attributes MessageContractAttribute, MessageHeaderAttribute andMessageBodyMemberAttribute to describe the structure of the SOAP Message.  
  
**Service Description**  
Issuing a HTTP GET Request with query WSDL causes ASP.NET to generate WSDL to describe the service. It returns the WSDL as a response to the request.  
  
The generated WSDL can be customized by deriving the class of ServiceDescriptionFormatExtension.  
  
Issuing a Request with the query WSDL for the .svc file generates the WSDL. The WSDL that generated by WCF can be customized by using ServiceMetadataBehavior class.  
  
**Exception Handling**In ASP.NET Web services, unhandled exceptions are returned to the client as SOAP faults.  
  
In WCF Services, unhandled exceptions are not returned to clients as SOAP faults. A configuration setting is provided to have the unhandled exceptions returned to clients for the purpose of debugging.  
  
**16. SOAP, WSDL, UDDI overview.**  
**XML**: Describes only data. So, any application that understands XML, regardless of the application's programming language or platform, has the ability to format XML in a variety of ways (well-formed or valid).  
  
**SOAP**: Provides a communication mechanism between services and applications.

**WSDL:** Offers a uniform method of describing web services to other programs.  
  
**UDDI**: Enables the creation of searchable Web services registries  
   
**Note: Some Contents are copied from various Interview Websites.**

**Differences Between Hashtable and Dictionary**

1. Hashtable is threadsafe and while Dictionary is not.  
2. Dictionary is types means that the values need not to boxing while Hashtable   
    values  need to be boxed or unboxed because it stored the  values and keys as   
    objects.           
3. When you try to get the value of key which does not exists in the collection, the   
    dictionary throws an exception of 'KeyNotFoundException' while hashtable returns  
    null value.  
4. When using large collection of key value pairs hashtable would be considered more   
    efficient than dictionary.  
5. When we retrieve the record in collection the hashtable does not maintain the order   
    of entries while dictionary maintains the order of entries by which entries were added.  
6. Dictionary relies on chaining whereas Hashtable relies on rehashing