What is framework 3.0?

.NET 3.0 = .NET 2.0 + Windows Communication Foundation + Windows Presentation Foundation + Windows Workflow Foundation + Windows CardSpace.

to select date part only

```
SELECT CONVERT(char(10),GetDate(),101)
```
--to select time part only
```
SELECT right(GetDate(),7)
```

what is AutoEventWireup?

The default value for AutoEventWireup is *true* for a C# web form, and *false* for a VB.NET web form. The IDE adds the default values to the @ Page directive for a new web form. The difference in defaults is partly because VB.NET has a mechanism for defining an event handler and subscribing to an event in one graceful motion (the *Handles* keyword).

There is no equivalent to the Handles keyword in C# (anonymous event handlers are arguably close, but just not the same). When AutoEventWireup is true, all we need to do is follow the method naming convention of Page_*EventToHandle*. The ASP.NET runtime will automatically find and fire the method for the appropriate event.

```
protected void Page_Load(object sender, EventArgs e)
{

}
```

If we switch to AutoEventWireup=”true” for a VB.NET web form, we can use the magic Page_EventName approach. The only change to the earlier VB.NET code would be to drop the Handles clause, and the events fire correctly.

If we switch to AutoEventWireup=”false” for a C# web form, there is a little extra work to do. Somewhere we need to explicitly wire up events. Here is one approach.

```
public partial class _Default : Page
{
   public _Default() // ctor
   {
      Load += new EventHandler(Page_Load);
      PreInit += new EventHandler(Page_PreInit);
   }

    protected void Page_Load(object sender, EventArgs e)
    {
       // ...
    }

   protected void Page_PreInit(object sender, EventArgs e)
   {
      // ...
    }
}
```

Here are the methods the runtime will look for when AutoEventWireup is true.

- Page_PreInit
- Page_Init
- Page_InitComplete
- Page_PreLoad
- Page_Load
- Page_LoadComplete
- Page_DataBind
- Page_SaveStateComplete
- Page_PreRender
- Page_PreRenderComplete
- Page_Unload
- Page_Error
- Page_AbortTransaction
- Page_CommitTransaction

What are locks?
Microsoft® SQL Server™ 2000 uses locking to ensure transactional integrity and database consistency. Locking prevents users from reading data being changed by other users, and prevents multiple users from changing the same data at the same time. If locking is not used, data within the database may become logically incorrect, and queries executed against that data may produce unexpected results.

Lock mode	**Description**
Shared (S)	Used for operations that do not change or update data (read-only operations), such as a SELECT statement.
Update (U)	Used on resources that can be updated. Prevents a common form of deadlock that occurs when multiple sessions are reading, locking, and potentially updating resources later.
Exclusive (X)	Used for data-modification operations, such as INSERT, UPDATE, or DELETE. Ensures that multiple updates cannot be made to the same resource at the same time.
Intent	Used to establish a lock hierarchy. The types of intent locks are: intent shared (IS), intent exclusive (IX), and shared with intent exclusive (SIX).
Schema	Used when an operation dependent on the schema of a table is executing. The types of schema locks are: schema modification (Sch-M) and schema stability (Sch-S).
Bulk Update (BU)	Used when bulk-copying data into a table and the TABLOCK hint is specified.

What is generic class?

Generic helps us to maximize the reuse of code, Type safety and better performance. Generics are most commonly used to create a collection. There are many new generic collection classes in the System.collection.Generics namespace. These classes are advised to be used for type safety and better performance over araylist.

How can C#app request minimum permission?

Using System.Security.Permissions;
[assembly:FileDialogPermissionAttribute(SecurityAction.RequestMinimum, Unrestricted=true)].

How to prevent caching in browser using asp.net ?
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 is interface class?
Interfaces, like classes, define a set of properties, methods, and events. But unlike classes, interfaces do not provide implementation. They are implemented by classes, and defined as separate entities from classes.
Which one trusted and which one 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.
What is immutable?

The data value may not be changed.? Note: The ***variable*** value may be changed, but the original immutable data value was discarded and a new data value was created in memory.

Difference between string and string builder ?

StringBuilder is more efficient in cases where there is a large amount of string manipulation.? Strings are immutable, so each time a string is changed,? a new instance in memory is created.

Difference between overloading and overriding?
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.
Difference between abstract class and interface?
In an interface class, all methods are abstract - there is no implementation. In an abstract class some methods can be concrete. In an interface class, no accessibility modifiers are allowed. An abstract class may have accessibility modifiers.
What is sealed class ?
The sealed modifier is used to prevent derivation from a class. A compile-time error occurs if a sealed class is specified as the base class of another class. (A sealed class cannot also be an abstract class)
Difference between early binding and 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.
‘ Create a variable to hold a new object.

Dim FS As FileStream
‘ Assign a new object to the variable.
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”)

Difference between overloading and overriding?
When overriding, you change the method behavior for a derived class. Overloading simply involves having a method with the same name within the class.
Can you allow class to be inherited, but prevent the method from being overridden?

Yes, just leave the class public and make the method sealed.

If the method is marked as protected internal how can accessed?
Method marked as Protected internally can be accessed by the Classes within the same assembly, and classes derived from the declaring class.
How to declare two dimensional array in c sharp ?
Int[,] arrayname;
What is satellite assembly ?
An assembly containing localized resources for another assembly.
Difference between overriding and shadowing ?
Overriding is used to redefines only the methods, but shadowing redefines the entire element.
Dangling pointer
A dangling pointer arises when you use the address of an object after its lifetime is over. This may occur in situations like returning addresses of the automatic variables from a function or using the address of the memory block after it is freed

```
class Sample
{
public:
int *ptr;
Sample(int i)
{
ptr = new int(i);
}

~Sample()
{
delete ptr;
}
void PrintVal()
{
cout << "The value is " << *ptr;
}
};

void SomeFunc(Sample x)
{
cout << "Say i am in someFunc " << endl;
}

int main()
{
Sample s1 = 10;
SomeFunc(s1);
s1.PrintVal();
}
```

Delegates in C#

Most programmers are used to passing data in methods as input and output parameters. Imagine a scenario where you wish to pass methods around to other methods instead of data.

Where are Delegates used?

The most common example of using delegates is in events.

You define a method that contains code for performing various tasks when an event (such as a mouse click) takes place.

This method needs to be invoked by the runtime when the event occurs. Hence this method, that you defined, is passed as a parameter to a delegate.

What are Generic Classes

> **Generic classes are classes that can hold objects of any class. Containers such as Lists, Arrays, Bags and Sets are examples of generic classes**. Container classes have the property that the type of objects they contain is of little interest to the definer of the container class but of crucial importance to the user of the particular container. Therefore, the type of the contained object is an argument to the container class. The definer specifies the container class in terms of this argument and the user specifies what the type of the contained object is to be for the particular container

Starting Threads/Parallel Processing:

You defined several methods and you wish to execute them simultaneously and in parallel to whatever else the application is doing. This can be achieved by starting new threads. To start a new thread for your method you pass your method details to a delegate.

Generic Classes: Delegates are also used for generic class libraries which have generic functionality defined. However the generic class may need to call certain functions defined by the end user implementing the generic class. This can be done by passing the user defined functions to delegates.

What is a n-tier Application?

Simply stated, an n-tier application helps us distribute the overall functionality into various tiers or layers.

For example in a typical implementation you can have one or more of the following layers
1) Presentation Layer
2) Business Rules Layer
3) Data Access Layer
4) Database/Data store

In certain scenarios some of the layers mentioned above may be split further into one or more sub layers.

Each Layer can be developed independently of the other provided that it adheres to the standards and communicates with the other layers as per the specifications.

This is one of the biggest advantages of the n-tier application.
Each layer can potentially treat the other layers as 'black-box'

In other words, each layer does not care how the other layer processes the data as long as it sends the right data in a correct format.

When n-tier Applications should not be used?

Building and implementing a successful n-tier application requires a lot of Effort, Skill, experience, commitment and Organizational Maturity.

It also implies cost.

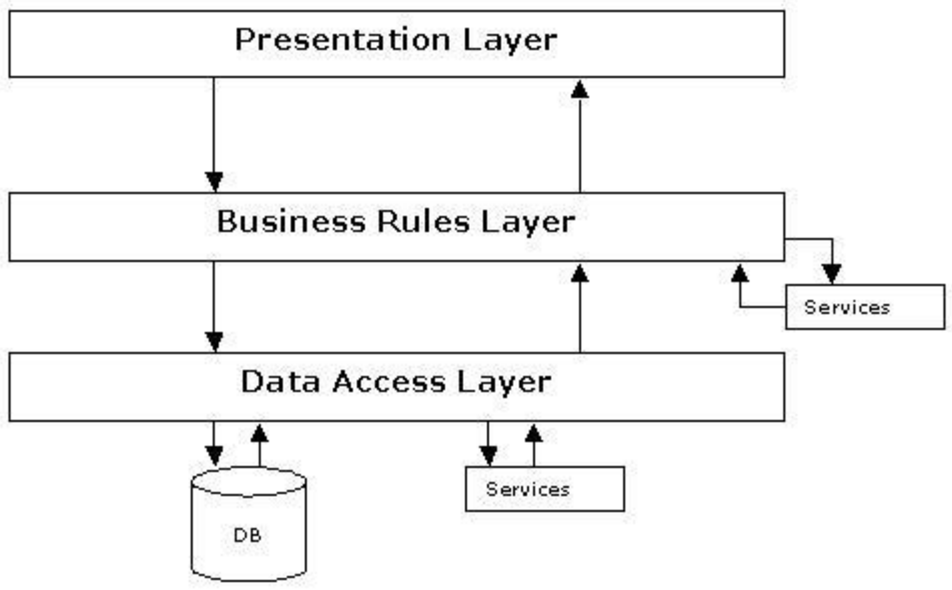
Hence a favorable Cost-Benefit Ratio is necessary before you decide to go ahead with the n-tier Application.

Building them with C#, .NET:

C#.NET provides us an excellent, robust feature rich platform.
C# being Object Oriented Programming Language it helps in practically laying down the standards. For instance you could create a base class with standard functions and require that all new classes should be derived from this base class. Please see the article on "Inheritance" in this series.

.NET provides type safety, automatic Garbage Collection which is very important in implementing good n-tier apps.

The Logical Building Blocks



The above diagram describes the logical building blocks of the Application.

1) The Presentation Layer: Also called as the client layer comprises of components that are dedicated to presenting the data to the user. For example: Windows/Web Forms and buttons, edit boxes, Text boxes, labels, grids, etc.

2) The Business Rules Layer: This layer encapsulates the Business rules or the business logic of the encapsulations. To have a separate layer for business logic is of a great advantage. This is because any changes in Business Rules can be easily handled in this layer. As long as the interface between the layers remains the same, any changes to the functionality/processing logic in this layer can be made without impacting the others. A lot of client-server apps failed to implement successfully as changing the business logic was a painful process.

3) The Data Access Layer: This layer comprises of components that help in accessing the Database. If used in the right way, this layer provides a level of abstraction for the database structures. Simply put changes made to the database, tables, etc do not effect the rest of the application because of the Data Access layer. The different application layers send the data requests to this layer and receive the response from this layer.

The database is not accessed directly from any other layer/component. Hence the table names, field names are not hard coded anywhere else. This layer may also access any other services that may provide it with data, for instance Active Directory, Services etc. Having this layer also provides an additional layer of security for the database. As the other layers do not need to know the database credentials, connect strings and so on.

4) The Database Layer: This layer comprises of the Database Components such as DB Files, Tables, Views, etc. The Actual database could be created using SQL Server, Oracle, Flat files, etc. In an n-tier application, the entire application can be implemented in such a way that it is independent of the actual Database. For instance, you could change the Database Location with minimal changes to Data Access Layer. The rest of the Application should remain unaffected.

Many packaged n-tier Applications are created so that they can work the same with SQL Server, Oracle, UDB and so on. In the above pages we have seen the background and the logical design of the n-tier application. Large enterprise apps are typically designed as n-tier applications and large portion of them are web based applications. Therefore they can be viewed in a secure manner, from any PC with a browser. This is a good combination of ease of use and security.

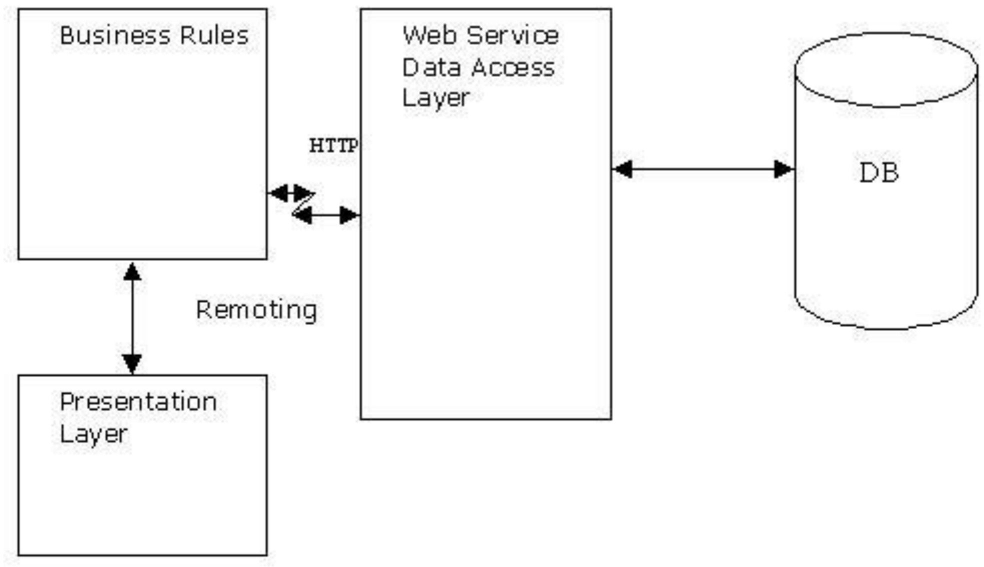
How do the Layers communicate with each other?

Each Layer comprises of one or more components. Each component being a part of the app may communicate with one or more component. The component may “speak” to the other components using one of the many protocols, HTTP, FTP, TCP/IP and mechanisms such as XML/RPC, SOAP, REMOTING etc. The data as such may be “passed” across in many formats such as binary, string , XML. For the purpose of this article we will use XML format for passing data between the components.

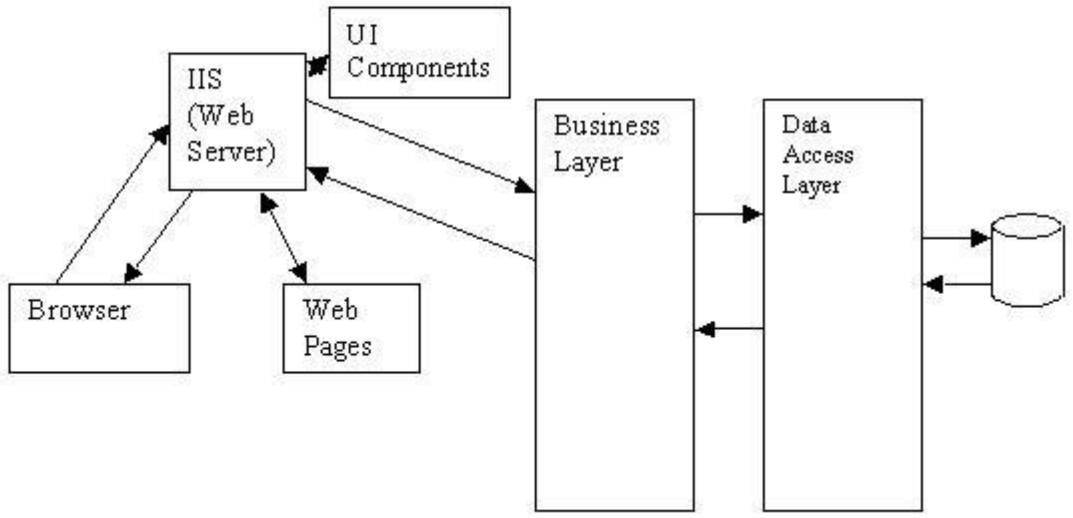
Communication Techniques in n-tier ‘.NET’ Apps :

The different layers of the n-tier applications can be located on physically different machines. Various techniques can be used to communicate between the various layers and components. The most common include

XML Web Services
.NET Remoting



A Web Based N-Tier Application



In the diagram above we observe that the Presentation Layer is made of Web Pages, Web Components and Web Server such as IIS. The end user sees the web pages on a Browser such as IE.

To create a web based n-tier Application using C#, .NET the following steps need to be followed:

1) Define what the Application should do. In other words the functionality of the application.

2) Segregate the application logic. The logic related to the User Interface or presentation layer will be part of the presentation layer, The logic related to the Business Rules goes to the Business Layer and so on.

3) Design the Database Structures, such as Tables, Views and so on. The Database design is a very crucial step and it effects the overall application performance, reliability and usability

4) After this you can design the Data Access Layer. This Layer comprises of code to access the database. The Data Access Layer components may typically be called by the Business Layer Components. The Design also defines how this call should be made. This includes the interface definition, Inputs, Outputs, Data Structure Definition and so on.

5) Similarly Business Application Layer design will define the Components that are part of this layer. Specifications for each component should include the Interface Definition, Inputs, outputs and Data structures (example XML DTD).

6) The Presentation Layer is designed keeping in mind the ease of Use. The User Interface should be intuitive and pleasant. The Presentation Layer may comprise of Web Pages such as .aspx,HTML, and compiled components

Inheritance in C#

This article discusses Inheritance concepts in the context of C#. Before we understand Inheritance in C# it is important to understand the key players involved, viz. Objects, Classes and Structs.

Constructors:

In C#, (like other Objected Oriented languages) constructor is a method having the same name as the class. The constructor is called when the object is being created. It can have one or more parameters.

Interfaces:

In the context of C#, an interface provides a contract. A class that is derived from this interface will implement the functions specified by the interface.

Inheritance:

C# supports two types of Inheritance mechanisms
1) Implementation Inheritance
2) Interface Inheritance

What is Implementation Inheritance?

- When a class (type) is derived from another class(type) such that it inherits all the members of the base type it is Implementation Inheritance

What is Interface Inheritance?

- When a type (class or a struct) inherits only the signatures of the functions from another type it is Interface Inheritance.

In general Classes can be derived from another class, hence support Implementation inheritance. At the same time Classes can also be derived from one or more interfaces. Hence they support Interface inheritance. Structs can derive from one more interface, hence support Interface Inheritance. Structs cannot be derived from another class they are always derived from System.ValueType

Multiple Inheritance:

C# does not support multiple implementation inheritance. A class cannot be derived from more than one class. However, a class can be derived from multiple interfaces.

Inheritance Usage Example:
Here is a syntax example for using Implementation Inheritance.

```
Class derivedClass:baseClass

{
}
```

derivedClass is derived from baseClass.

Interface Inheritance example:

```
private Class derivedClass:baseClass , InterfaceX , InterfaceY
{
}
```

derivedClass is now derived from interfaces – InterfaceX, InterfaceY
Similarly a struct can be derived from any number of interfaces

```
private struct childStruct:InterfaceX, InterfaceY
{
}
```

Virtual Methods:

If a function or a property in the base class is declared as virtual it can be overridden in any derived classes

Usage Example:

```
class baseClass
{
         public virtual int fnCount()
         {
```

```
                  return 10;
         }
}
class derivedClass :baseClass
{
        public override int fnCount()
         {
                return 100;
         }
}
```

This is useful because the compiler verifies that the 'override' function has the same signature as the virtual function

Hiding Methods:
Similar to the above scenario if the methods are declared in a child and base class with the same signature but without the key words virtual and override, the child class function is said to hide the base class function

```
class someBaseClass
{
}
class abcClass:someBaseClass
{
        public int fnAge()
         {
                  return 99;
         }
}
```

```
class grandchildClass: abcClass
{
        public int fnAge()
        {
                return 10;
        }
}
```

What are Abstract Classes?

1) An abstract class cannot be instantiated.
2) An abstract class can have one or more abstract functions.
3) Abstract functions are virtual.
4) They do not have any implementation.
5) They need to be overridden and implemented in a derived non-abstract class

Abstract Classes and Functions example:

```
abstract class Car
{
        public int headlights = 2;
        public abstract price();
}
```

Sealed Classes:

If a class is declared as Sealed you cannot inherit from that class.
Declaring a method as sealed prevents it from being overridden

Example

Sealed class test

{

}

```
Class bankmgr

{

Public sealed override bool autherzid()

{

}

}
```
The compiler gives an error if you override the above method

Constructors and Inheritance:

C# allocates a default zero parameter constructor to every class that does not have any explicit constructors defined. If you instantiate a child class, all the constructors in the hierarchy are called. The base call constructor is called first and then the next child class constructor. This sequence continues until all the constructors are called If an explicit constructor is defined for a class anywhere in the hierarchy there is a possibility that the above chain is broken. If this is the case the compiler raises an Error and the code will not compile.

For instance, if you declare an explicit constructor with one or more parameters the above described sequence of calls to constructors in the class hierarchy which was being handled automatically is now broken. This is because when you supply a constructor C# does not provide a default constructor. In this case, we have to explicitly maintain the 'chain'. Another scenario for this error is when you define an explicit constructor with zero parameters and mark it as private. The compiler will raise an error in this case.

Visibility Modifiers in C#:

public: Any types or members can be prefixed with this modifer. If a member or type is prefixed with public it is visible to all the code.

protected: It can be used for any member or a nested type. This causes the member/nested type to be visible to any derived type.

private: This can be used for any type or member and the member/type will be visible only inside the type where it was defined

internal: This causes the member/nested type to be visible within the assembly where it is defined

protected internal: This causes the member/nested type to be visible within the assembly where it is defined and any derived type.

Interfaces:

When a class derives from an Interface it implements the functions specified by the interface.

Defining an Interface

We can define an interface as follows:

```
Public interface ibkmgr

{

            Bool authorize();

}

Public class check : ibkmgr

{

            Bool authorize()

{



}

}
```

What Partial class?

One of the language enhancements in .NET 2.0—available in both VB.NET 2005 and C# 2.0—is support for partial classes. In a nutshell, partial classes mean that your class definition can be split into multiple physical files. Logically, partial classes do not make any difference to the compiler. During compile time, it simply groups all the various partial classes and treats them as a single entity.

One of the greatest benefits of partial classes is that it allows a clean separation of business logic and the user interface (in particular the code that is generated by the visual designer). Using partial classes, the UI code can be hidden from the developer, who usually has no need to access it anyway. Partial classes will also make debugging easier, as the code is partitioned into separate files.

Partial classes allow you to split the code that makes up a class over multiple files, here's how, this isn't the full code so you have to add it to the generated files in visual studio.

Book1.cs

--

```
public partial class Book
{
        public string Render()
        {
                return "ASP.NET 2.0 Step by Step";
        }
}
```

Book2.cs

--

```
public partial class Book
{
        public string GetReviews()
        {
                return "Here are the reviews...";
        }
}
```

Default.aspx

--

```
        protected void Page_Load(object sender, EventArgs e)

        {

                Book book = new Book();

                Label1.Text = book.Render() + " --> " +
book.GetReviews();

        }
```

ASP.NET 2.0 has five DataSource controls:

- **AccessDataSource:** Connects data binding controls to an Access database
- **ObjectDataSource:** Connects data binding controls to data objects/components
- **SiteMapDataSource:** Connects site navigation controls to site map data
- **SqlDataSource:** Connects data binding controls to a SQL Server database
- **XmlDataSource:** Connects data binding controls to XML data

ObjectDataSource :-

ObjectDataSource belongs to the family of data source controls in ASP.NET, which enables a declarative databinding model against a variety of underlying data stores, such as SQL databases or XML. Most data source controls encourage a two-tiered application architecture, where the presentation layer (the page) interacts directly with the backend data provider. However, it is also common for page developers to encapsulate data retrieval (and optionally business logic) into a component object, introducing an additional layer between the presentation page and data provider. The ObjectDataSource control allows developers to structure their applications using this traditional three-tiered architecture and still take advantage of the ease-of-use benefits of the declarative databinding model in ASP.NET.

The ObjectDataSource control object model is similar to the SqlDataSource control. Instead of a ConnectionString property, ObjectDataSource exposes a **TypeName** property that specifies an object type (class name) to instantiate for performing data operations. Similar to the command properties of SqlDataSource, the ObjectDataSource control supports properties such as **SelectMethod**, **UpdateMethod**, **InsertMethod**, and **DeleteMethod** for specifying methods of the associated type to call to perform these data operations. This section describes techniques for building data access layer and business logic layer components and exposing them through an ObjectDataSource control.

Example :-

create one class file

```
public class CustomerData

{

public CustomerData()
      {
            //
            // TODO: Add constructor logic here
            //

      }

public System.Data.DataSet GetMasterData()
    {
        //string connectionString =
ConfigurationManager.ConnectionStrings["ConnString"].ConnectionString;
        //System.Data.IDbConnection dbConnection = new
System.Data.SqlClient.SqlConnection(connectionString);

        SqlConnection dbConnection = new
SqlConnection(ConfigurationManager.AppSettings["ConnString"]);
        string strqry;
        strqry = "select masterid,mastername,masterdetail,masterurl
from dbo.mastertable";
        SqlDataAdapter adp = new SqlDataAdapter(strqry, dbConnection);
        DataSet myds = new DataSet();
        adp.Fill(myds);
        return myds;

    }

}
```

Now in default.aspx page

```
<form id="form1" runat="server">
    <div>
        <asp:GridView ID="GridView1" runat="server" AllowPaging="True"
AllowSorting="True"
            DataSourceID="ObjectDataSource1">
        </asp:GridView>

    </div>
        <asp:ObjectDataSource ID="ObjectDataSource1" runat="server"
SelectMethod="GetMasterData"
            TypeName="CustomerData"></asp:ObjectDataSource>

    </form>
```

<u>Difference between dataset and data reader</u>

Dataset is work in disconnected mode and data reader work always work connection mode.

But now onwards data reader work on disconnect mode also

Datatablereader use for that. That is good example for that see below.

```
void BindData()
    {
        string sql = "select masterid,mastername,masterdetail,masterurl
from dbo.mastertable";
  SqlDataAdapter da = new SqlDataAdapter(sql,conreader);
  DataTable dt = new DataTable();
  da.Fill(dt);
  //DataTableReader dtr = dt.CreateDataReader();
  System.Data.DataTableReader dtr = dt.CreateDataReader();

  if (dtr.HasRows)
  {
    while (dtr.Read())
    {
      Response.Write(dtr["mastername"].ToString() + "<br/>");
    }
  }
  else
    Response.Write("No Data");

    }
```

Example 2: Clean up a string having invalid characters:

```
string CleanInput(string strInput)
{
     // Replace invalid characters with empty strings.
     return Regex.Replace(strInput, @"[^\w\.@-]", "");
}
```

Example 3: Find a sub-string corresponding to a particular pattern

```
using System;

using System.Collections;

using System.Text.RegularExpressions;

int FindString(string strinput)

{

     string matchMyStyle = “<.”;


     Regex RE1 = new Regex(matchMyStyle , RegexOptions.Multiline);

     MatchCollection ListMatched = RE1.Matches(strinput);

     int i = ListMatched.Count;

     return i;

}
```

Example 4: Find a specific occurrence of a Pattern Match

```
string strinput ="< Employee > < / Employee > ";

string strGetValue ="";

int FindOccurrence = 2; //to find the second occurrence


string matchMyStyle = @"< .....";  //match pattern



Regex RE1 = new Regex(matchMyStyle , RegexOptions.Multiline);

MatchCollection ListMatched = RE1.Matches(strinput);


int i = ListMatched.Count;
```

```
     // if total no of occurrences is less than expected do nothing
if ( FindOccurrence< =i )
{
     //get the value of specified of occurrence.
     strGetValue = ListMatched[FindOccurrence-1].Value.ToString();

}
```

Example 5: Adding Comments to a Regular Expression

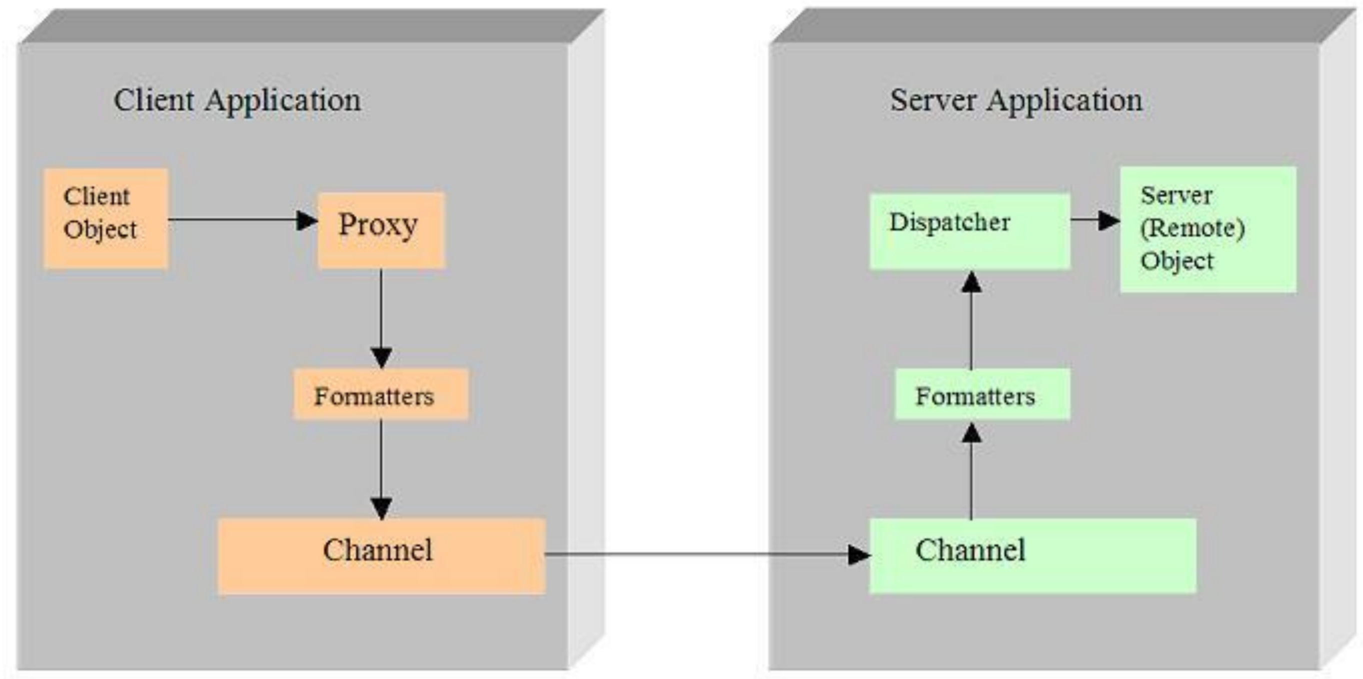
To add a comment within a Regular Expression use the # sign and include the option RegexOptions.IgnorePatternWhitespace as shown in the example below

```
int FindString(string strinput)
{
     string matchMyStyle = @"<.# Need to Find this";

     Regex RE1 = new Regex(matchMyStyle ,
     RegexOptions.Multiline|RegexOptions.IgnorePatternWhitespace);
     MatchCollection ListMatched = RE1.Matches(strinput);
     int i = ListMatched.Count;
     return i;
}
```

NET Remoting

Remoting enables software components to interact across application domains. The components interacting with each other can be in different processes and systems. This enables us to create n-tier Distributed applications. For more details on N-tier Applications see the article on n-tier Applications in this series. In this article we try to explain .NET Remoting in a clear and concise manner



The Key Players are:

- Client Object
- Server (Remote) Object
- Proxy Object
- Formatter
- Channel

Client Object is the object or component that needs to communicate with(call) a remote object.

The Server (Remote) Object receives the request from the client object and responds.

Proxy Object:

When the client object needs to call a method from the Remote Object it uses a proxy object to do this. Every public method that defined in the remote object class can be made available in the proxy and thus can be called from clients. The proxy object acts as a representative of the remote object. It ensures that all calls made on the proxy are forwarded to the correct remote object instance. There are two types of proxies transparent and real proxy

The **TransparentProxy** contains a list of all classes, as well as interface methods of the remote object. It examines if the call made by the client object is a valid method of the remote object and if an instance of the remote object resides in the same application domain as the proxy. If this is true, a simple method call is routed to the remote object.

If the object is in a different application domain, the call (call parameters on the stack are packaged into an IMessage object) is forwarded to a **RealProxy** class by calling its Invoke method.
This class is then responsible for forwarding messages to the remote object.

Formatter

The formatting can be done be done in three ways –

a) Binary
b) SOAP or
c) Custom.

The remoting framework comes with two formatters: the binary and SOAP formatters. The binary formatter is extremely fast, and encodes method calls in a proprietary, binary format. The SOAP formatter is slower. Developers can also write their own Custom Formatter and have the ability to use that.

Channels

Channels are used to transport messages to and from remote objects. You can choose a **TcpChannel** or a **HttpChannel** or extend one of these to suit your requirements.

HTTP channel : The HTTP channel transports messages to and from remote objects using the SOAP protocol. However, All messages are passed through the SOAP formatter, where the message is changed into XML and serialized, and the required SOAP headers are added to the stream. The resulting data stream is then transported to the target URI using the HTTP protocol.

TCP Channel : The TCP channel uses a binary formatter to serialize all messages to a binary stream and transport the stream to the target URI using the TCP protocol. It is also possible to configure the TCP channel to the SOAP formatter.

Understanding .NET Remoting

.NET Remoting enables software components to interact across application domains. Previously Inter-process Communication between various apps on the Microsoft Platform was widely handled using DCOM. DCOM has its limitations as it relies on proprietary format. Also the difficulty of communication between COM objects spread across firewalls. .NET Remoting eliminates these difficulties as it supports various transport and communication protocols and is adaptable to diverse network environments. It supports state management options. It can correlate multiple calls from the same client and support callbacks. However it relies on the existence of the common language runtime assemblies. Needless to say, it requires the clients be built using .NET. With this background let us explore what it takes to implement .NET Remoting

Implementing .NET Remoting:

Step 1: Create a Remote Object (Server)
There are three types of objects that can be configured to serve as .NET remote objects.

· Single Call
Single Call objects service one and only one request coming in. Single Call objects are useful in

scenarios where the objects are required to do a finite amount of work. Single Call objects are usually not required to store state information, and they cannot hold state information between method calls. However, Single Call objects can be configured in a load-balanced fashion.

· Singleton Objects
Singleton objects are those objects that service multiple clients and hence share data by storing state information between client invocations. They are useful in cases in which data needs to be shared explicitly between clients and also in which the overhead of creating and maintaining objects is substantial.

· Client-Activated Objects (CAO)
Client-activated objects (CAO) are server-side objects that are activated upon request from the client. This way of activating server objects is very similar to the classic COM coclass activation. When the client submits a request for a server object using "new" operator, an activation request message is sent to the remote application. The server then creates an instance of the requested class and returns an ObjRef back to the client application that invoked it. A proxy is then created on the client side using the ObjRef. The client's method calls will be executed on the proxy. Client-activated objects can store state information between method calls for its specific client and not across different client objects. Each invocation of "new" returns a proxy to an independent instance of the server type.

l) Conversions A conversion enables an expression of one type to be treated as another type. Conversions can be implicit or explicit. A conversion enables an expression of one type to be treated as another type. Conversions can be implicit or explicit.

A conversion enables an expression of one type to be treated as another type. Conversions can be implicit or explicit.

m) Arrays An array is a data structure. It contains one or more variables that are accessed through computed indices. The elements of the array, are all of the same type.

n) Memory Management: One of the most important features of C# is automatic memory management implemented using a ‘garbage collector’. The process scans thru the objects created in the program and if the object can no longer be accessed the memory is cleared up

o) Indexers : An indexer enables an object to be indexed in the same way as an array. Indexer declarations are similar to property declarations. The indexing parameters are provided between square brackets. Example