CBCS SCHEME

USN 1 C R 2 0 M C 0 9 8

20MCA42

Fourth Semester MCA Degree Examination, July/August 2022
Programming using C#

Time: 3 hrs.

Max. Marks:100

Note: *Answer any FIVE full questions, choosing ONE full question from each module.*

Module-1

1 a. Explain the components of •Net framework with the help of architecture diagram. **(10 Marks)**

b. List and explain the different data types in C#. **(10 Marks)**

OR

2 a. Discuss between value types of reference types and write a program to explain boxing and unboxing. **(10 Marks)**

b. Write short notes on :
i) Windows work flow foundation
ii) Windows cold space and LINQ. **(10 Marks)**

Module-2

3 a. Explain partial classes and partial methods, with the help of a C# program. **(10 Marks)**

b. Write a C# program to explain accessor and mutator properties used in encapsulation. **(10 Marks)**

OR

4 a. Explain the significance of "is – a" and "has – a" relationship in inheritance with an example program. **(10 Marks)**

b. Explain the following with example :
i) Static class and static members
ii) Abstract class and abstract methods. **(10 Marks)**

Module-3

5 a. Explain the four steps involved in creating and using delegate in the program. **(10 Marks)**

b. Explain the architecture of APO•NET with a neat diagram and list the ADO•NET provided. **(10 Marks)**

OR

6 a. Write a C# program using try, catch, finally to explain any predefined exceptions. **(10 Marks)**

b. Explain the properties and methods of Data Reader and Data Adaptor class. **(10 Marks)**

Module-4

7 a. Explain various mouse events in C# windows application. **(10 Marks)**

b. Explain the WPF architecture with a neat diagram. **(10 Marks)**

OR

8 a. Explain the following : (10 Marks)
i) XAML elements
ii) Makeup extension classes in XAML.

b. Write short notes on : (10 Marks)
i) MDI window forms
ii) Event –Driven GUI.

Module-5

9 a. Explain different validation controls with suitable example supported by ASP•NET. (10 Marks)

b. Explain the architecture of three tier web based application. (10 Marks)

OR

10 a. What are Cookies? Explain session management in ASP•NET using cookies. (10 Marks)
b. Explain the controls from AJAX control toolkit. (10 Marks)

*****

1 a.

The .Net framework is a software platform developed by Microsoft organization useful for development. The Net framework helps to build applications that would run on the Windows platform. The 1st version of this framework came into existence in the year 2002. In simple terms, a .Net framework is a virtual machine (VM) for assembling and performing programs written in multiple languages like C#, VB.Net, etc.

This framework is useful to build Form-based applications, Web-based applications, and Web services. Moreover, the .Net framework also provides a lot of functionalities and also supports industry standards. .Net framework supports more than 50 programming languages and a few of them are built by Microsoft.

In short, .Net framework is a;

- The platform builds for app developers.
- Framework; supports several languages and cross-language combinations.
- It also includes an IDE for writing programs.
- It is a set of utilities or in other words, building blocks of the user application system.
- The framework provides a Graphical User Interface in the form of GUI.
- Moreover, .NET is an individual platform but it uses Mono Compilation System (MCS) where it is a mid-level interface.

.Net Framework Architecture

The architecture of the .Net framework is a kind of programming model for the Net platform under which it offers a managed performance environment, development, and deployment in the simplest way. And the integration process with a wide range of coding languages.

Furthermore, the.Net framework architecture is based on some key components that include;

- CLS- Common Language Specification
- CTS- Common Type Specification

· FCL- .Net Framework Class Library

· CLR- Common Language Runtime

· CLI- Common Language Infrastructure

Moreover, there are different range of languages that .Net framework supports.

CLS- Common Language Specification

Common Language Specification or CLS includes a set of procedures that specifies a .NET language. Apps build using several coding languages like C# and VB.NET, are shorten down to CLS or Common Language Specification.

CLI- Common Language Infrastructure

The CLI or Common Language Infrastructure provides a language-independent platform for app development and its performance. It also includes the function for exception handling, waste collection, security, etc.

The CLI has the following key features:

- **Handling Anomaly**- Anomalies are errors/bugs that occur while the application is executed.

Examples of anomalies or exceptions are as follows:

In case an application tries to open a file on the local system, but the file is not present.

In case the application tries to raise some records from a database, but the connection to the database becomes invalid.

- **Waste Collection** – Collecting waste or garbage is the process of deleting unwanted resources when these are no longer needed.

Examples of the waste collection are as follows;

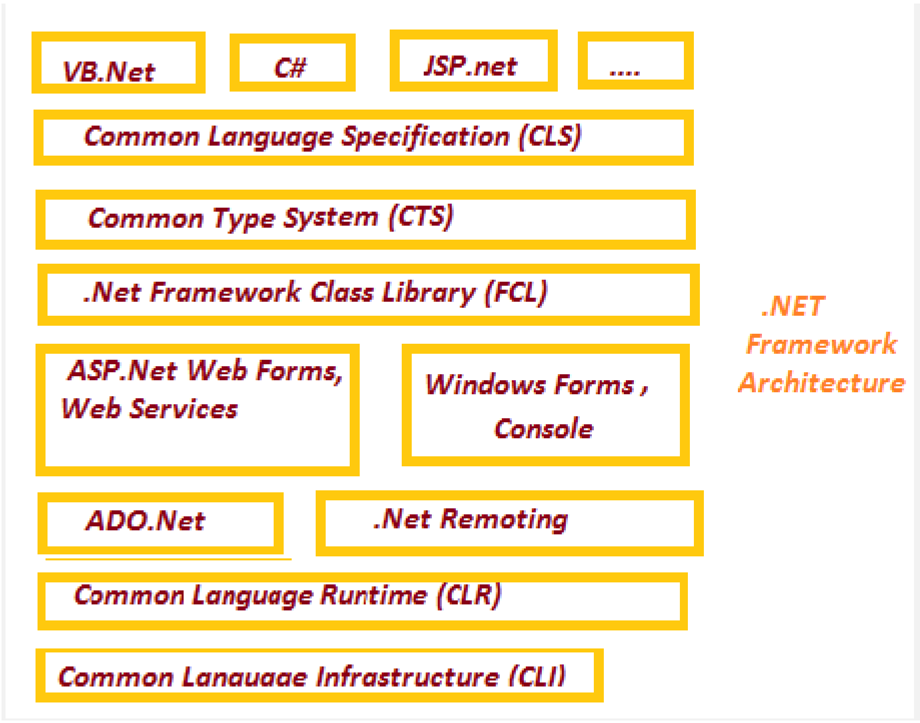
- A file handle that is no longer useful. In case the application has completed all operations on a file, then the file handle may no longer be useful.
- Besides, the database connection that doesn’t have longer use. If the application has completed all its operations on a database, then the database connection may no longer be used.

- **Working with different languages** –

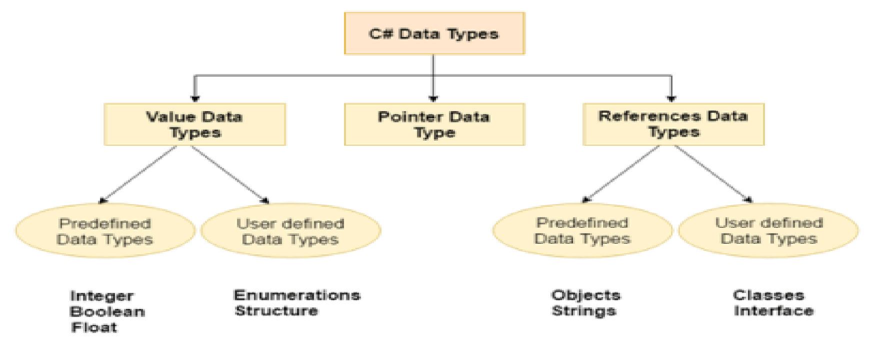
Language – The first level includes the programming language where the most common ones are VB.Net and C# languages.

Compiler – The compiler that exists here may be separate for each coding language. So basically the VB.Net language will have an individual VB.Net compiler. Likely, for C# also user will have another compiler.

CLI– the CLI is the last layer in the net framework. This would be useful to run a .net program built in any coding language. So the afterward compiler will send the coding to the CLI layer to run the .Net application.


VB.Net
C#
JSP.net
....
Common Language Specification (CLS)
Common Type System (CTS)
.Net Framework Class Library (FCL)
ASP.Net Web Forms, Web Services
Windows Forms , Console
ADO.Net
.Net Remoting
Common Language Runtime (CLR)
Common Lanauaae Infrastructure (CLI)
.NET Framework Architecture

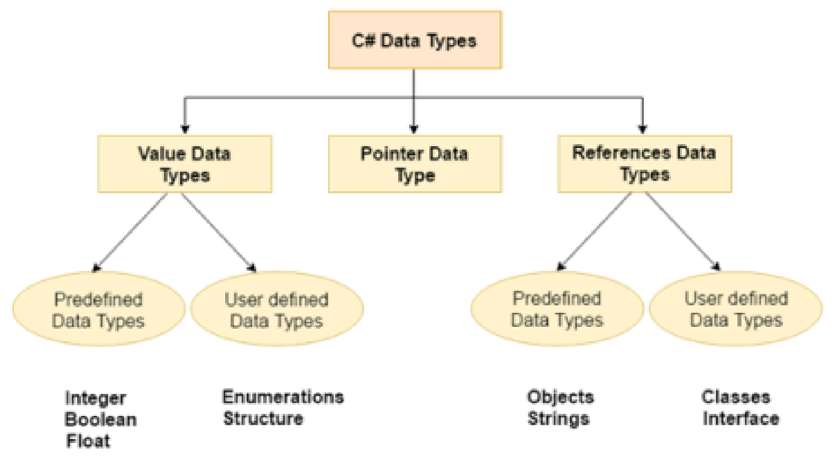
1 b.



Value types: Value types directly contain their data, and instances of value types are either allocated on the stack or allocated inline in a structure. Value types can be built-in (implemented by the runtime), user-defined, or enumerations.

Reference types Reference types store a reference to the value's memory address, and are allocated on the heap. Reference types can be self-describing types, pointer types, or interface types. The type of a reference type can be determined from values of self-describing types. Self-describing types are further split into arrays and class types. The class types are user-defined classes, boxed value types, and

2 a.



Value types: Value types directly contain their data, and instances of value types are either allocated on the stack or allocated inline in a structure. Value types can be built-in (implemented by the runtime), user-defined, or enumerations.

Reference types Reference types store a reference to the value's memory address, and are allocated on the heap. Reference types can be self-describing types, pointer types, or interface types. The type of a reference type can be determined from values of self-describing types. Self-describing types are further split into arrays and class types. The class types are user-defined classes, boxed value types, and

i. Boxing and Unboxing

Boxing is the process of converting a value type to the object type or any interface type implemented by his value type. Boxing is implicit.

Example: Boxing

Copy

```
int i = 10;

bject o = i; //performs boxing
```

In the above example, the integer variable i is assigned to object o. Since object type is a reference type nd base class of all the classes in C#, an int can be assigned to an object type. This process of onverting int to object is called boxing.

Let's look at a more practical example.

Example: Boxing

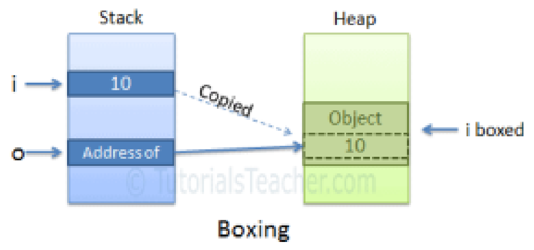
Copy

```
ArrayList list = new ArrayList();

list.Add(10); // boxing

list.Add("Bill");
```

Above, ArrayList is a class in C#, and so it is a reference type. We add an int value 10 in it. So, .NET will perform the boxing process here to assign value type to reference type.

As you know, all the reference types stored on heap where it contains the address of the value and value type is just an actual value stored on the stack. Now, as shown in the first example, int i is assigned to object o. Object o must be an address and not a value itself. So, the CLR boxes the value type by creating a new System.Object on the heap and wraps the value of i in it and then assigns an address of that object to o. So, because the CLR creates a box on the heap that stores the value, the whole process is called 'Boxing'.

The following figure illustrates the boxing process.



Unboxing is the reverse of boxing. It is the process of converting a reference type to value type. Unboxing extract the value from the reference type and assign it to a value type.

Unboxing is explicit. It means we have to cast explicitly.

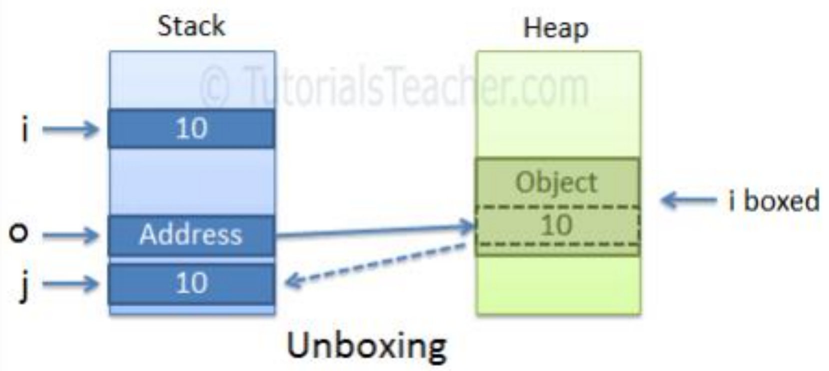
Example: Unboxing

Copy

object o = 10;

int i = (int)o; //performs unboxing

The following figure illustrates the unboxing process.



A boxing conversion makes a copy of the value. So, changing the value of one variable will not impact others.

int i = 10;

object o = i; // boxing

o = 20;

Console.WriteLine(i); // output: 10

The casting of a boxed value is not permitted. The following will throw an exception.

Example: Invalid Conversion

Copy

int i = 10;

object o = i; // boxing

double d = (double)o; // runtime exception

First do unboxing and then do casting, as shown below.

Example: Valid Conversion

Copy

int i = 10;

object o = i; // boxing

double d = (double)(int)o; // valid

2 b.i The Windows Workflow foundation provides a platform built on the .Net Framework for building workflows. It is not an application or a server product, and is included as a component of the .Net 3.0 framework. The platform unleashes the power of a completely different paradigm.

Some of the major criteria for workflows are listed below:

1. Visual Representation of process(es)
2. Can be dynamically modified at run-time
3. Can be long-running

There are two major types of workflows -

- Sequential workflows: used for well-defined, process workflows
- State Machine workflows: Organized as state machine diagrams, typically used for workflows with human interaction.

Workflows can either be built declaratively by using only Markup, or only Code or a combination of both Markup and code.

Once the workflow is developed, it can be hosted on any application that can load the Workflow Runtime such as Windows Forms, Windows Services, ASP.Net Web Sites and Web Services.

Activities: Workflows consist of one or more activities. Activities are the building blocks of workflows. A set of pre-built activities is provided for developers. You can also build your own custom activities.

Services: The workflow runtime engine uses many services when executing a workflow instance. You can either use the services in the Windows Workflow Foundation, or you can customize the available services, or you can build your own services.

Long Running Workflow: Workflows in real life can have a long and unpredictable execution life. The Windows Workflow Foundation handles this capability and can persist workflows, as required.

Compensation: Transactions in the workflow world are different compared to traditional transactions. In cases where we have long running workflows, it may not be possible to exactly "rollback" a set of steps when an exception occurs. Instead, the workflow allows "compensation" which in simple terms is the action taken to cover up for the effect of the part of the transaction that has already been completed.

Tracking: The visual nature of a workflow definition leads us to the next requirement, which is - tracking the progress of a running workflow. Windows Workflow Foundation provides the Tracking services to track the status of workflow instances.

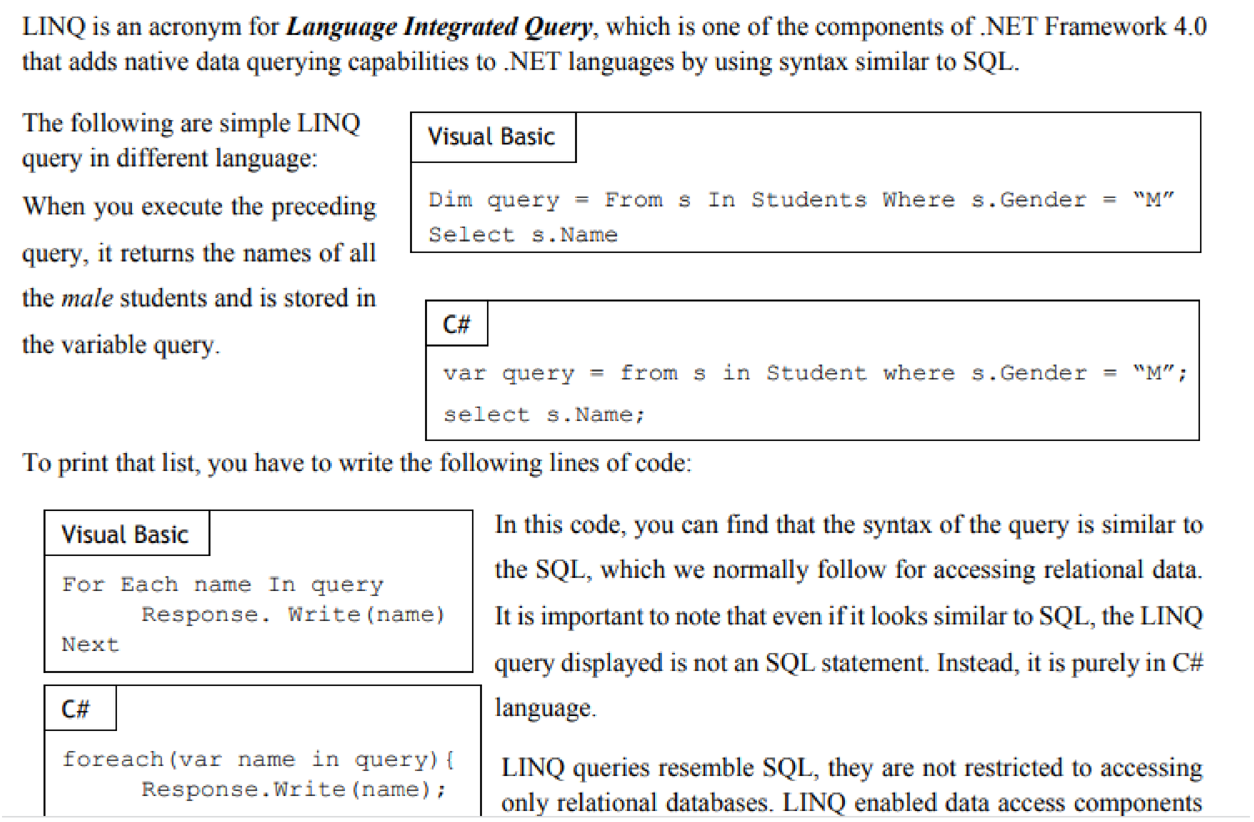
Rules and Conditions: You can apply logical conditions in 2 ways in workflows

1. Specify Code Conditions: programmatically define whether certain paths of the workflow are run. This is similar to specifying a condition in an If Statement, in traditional programming.

2. Rules: programmatically or declaratively define the business rules for workflows.

   Rules allow complex forward chaining rule sets and lend towards easy modification at runtime. Rules also enable the separation of Business logic and Process flow and can be shared across workflows

ii.



LINQ is an acronym for ***Language Integrated Query***, which is one of the components of .NET Framework 4.0 that adds native data querying capabilities to .NET languages by using syntax similar to SQL.

The following are simple LINQ query in different language:

When you execute the preceding query, it returns the names of all the *male* students and is stored in the variable query.

Visual Basic

```
Dim query = From s In Students Where s.Gender = "M"
Select s.Name
```

C#

```
var query = from s in Student where s.Gender = "M";
select s.Name;
```

To print that list, you have to write the following lines of code:

Visual Basic

```
For Each name In query
      Response. Write(name)
Next
```

C#

```
foreach(var name in query){
      Response.Write(name);
```

In this code, you can find that the syntax of the query is similar to the SQL, which we normally follow for accessing relational data. It is important to note that even if it looks similar to SQL, the LINQ query displayed is not an SQL statement. Instead, it is purely in C# language.

LINQ queries resemble SQL, they are not restricted to accessing only relational databases. LINQ enabled data access components

→ **LINQ to ADO.NET:-** Includes two options, LINQ to SQL, which translates a query into an SQL query, and then issues it against the tables specified in the query in an SQL Server database, and LINQ to DataSet, which executes a query on the contents of a DataSet.

→ **LINQ to Object: -** Allows querying objects in a collection. LINQ to Objects is not dynamic. After creating and using the result set, any changes made to the source collection do not automatically update the result set.

→ **LINQ to XML: -** Allows querying of XML data. In addition, it helps in creating and manipulating XML data. This option has a different syntax; however, the basic LINQ query remains the same.

3 a.

The partial class is a class that enables you to specify the definition of a class, structure, or interface in two or more source files. All the source files, each containing a section of class definition, combine when the application is complete. You may need a partial class when developers are working on large projects. A partial class distributes a class over multiple separate files; allowing developers to work on the class simultaneously. You can declare a class as partial by using the "partial" keyword. All the divided sections of the partial class must be available to form the final class when you compile the program. Let's see that in above figure. All the section must have the same accessibility modifiers, such as public or private.

Partial method are only allowed in partial types, such as classes and structs. A partial method consists of 2 parts that listed below: Deals with defining the partial method⌉ Deals with implementing the partial method⌉ declaration Rules: Must have a void return type No access modifier are allowed for declaring a partialmethod except for static Partial methods are private by default

```
using System;
namespace Class_Demos{
    partial class MyTest {
        private int a;
        private int b;
        public void getAnswer(int a1, int b1){
            a = a1;
            b = b1;
        }
        static partial void Message();
    }
    partial class MyTest{
        partial void Message(){
            Console.WriteLine("Successfully accessed. . . . . . ");
        }
        public void DisplayAns(){
            Console.WriteLine("Integer values: {0}, {1}", a, b);
            Console.WriteLine("Addition:{0}", a + b);
            Console.WriteLine("Multiply:{0}", a *  b);
            Message();
        }
    }
    class PartialEx{
        public static void Main(){
            MyTest ts = new MyTest();
            ts.getAnswer(2, 3);
            ts.DisplayAns();
            Console.Read();
}}}
```

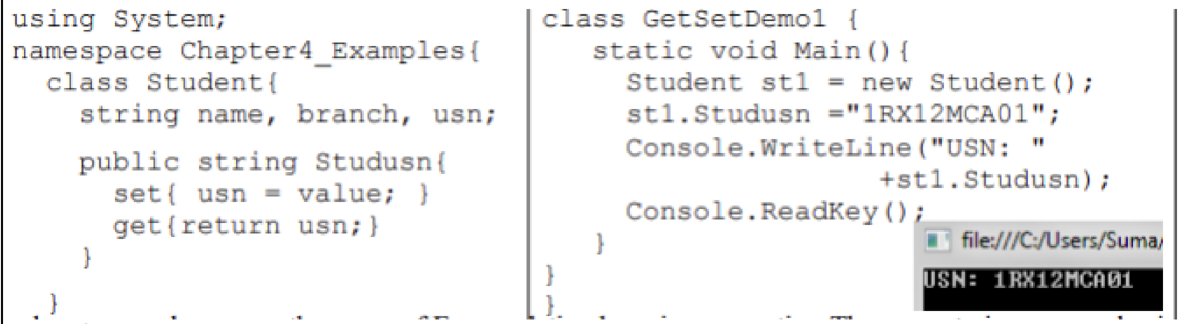
3 b. Encapsulation using accessors and mutators: Rather than defining the data in the form of public, we can declare those fields as private so that we achieved encapsulation. The Private data are manipulated using accessor (get: store the data to private members) and mutator (set: to interact with the variable) methods.

Syntax:

set { }
get { }

. A property defined with both a getter and a setter is called a read-write property. A property defined with only a getter is called a read-only property. A property defined with only a setter is called a write-only property.

Encapsulation using Properties: i. Write-Only Property: Properties can be made write-only. This is accomplished by having only a set mutator in the property implementation.



```
using System;
namespace Chapter4_Examples{
  class Student{
    string name, branch, usn;

    public string Studusn{
      set{ usn = value; }
      get{return usn;}
    }

  }
```

```
class GetSetDemo1 {
   static void Main(){
     Student st1 = new Student();
     st1.Studusn ="1RX12MCA01";
     Console.WriteLine("USN: "
                     +st1.Studusn);
     Console.ReadKey();
   }
}
}
```

file:///C:/Users/Suma/

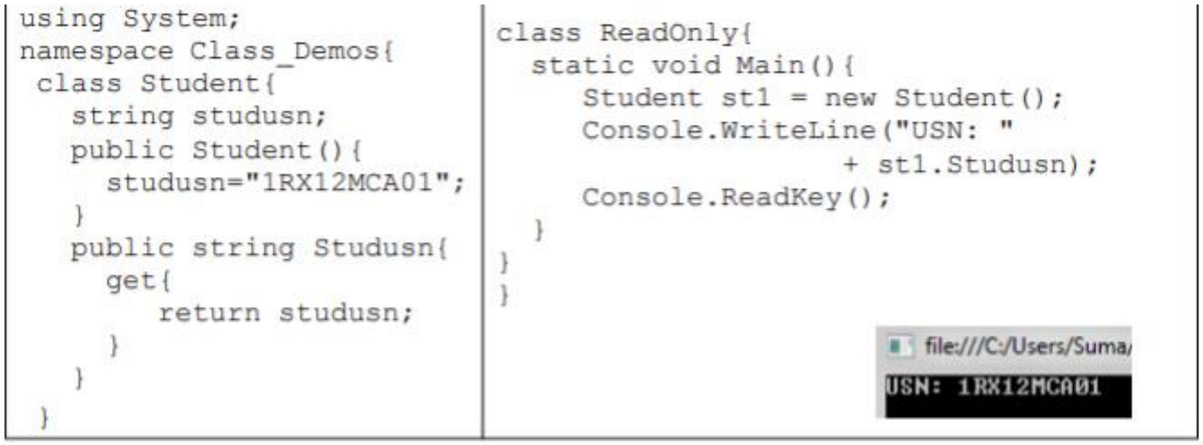
```
USN: 1RX12MCA01
```

Example:
class Student
{
string name, branch, usn;
public string Studusn
{ set{usn = value;} }
}

In this example, the property "Studusn" is having only set (mutator) but not get (accessor). So Studusn can called as a "write-only property"

. ii. Creating Read-Only Fields: Properties can be made read-only. This is accomplished by having only get

accessor in property implementation. Example 2.4: In the below example, the property "Studusn" is having only get (accessor) but not set(mutator). So can call Studusn as a "read-only property". class Student{ string name, branch, usn; public string Studusn{ set{usn = value;} }



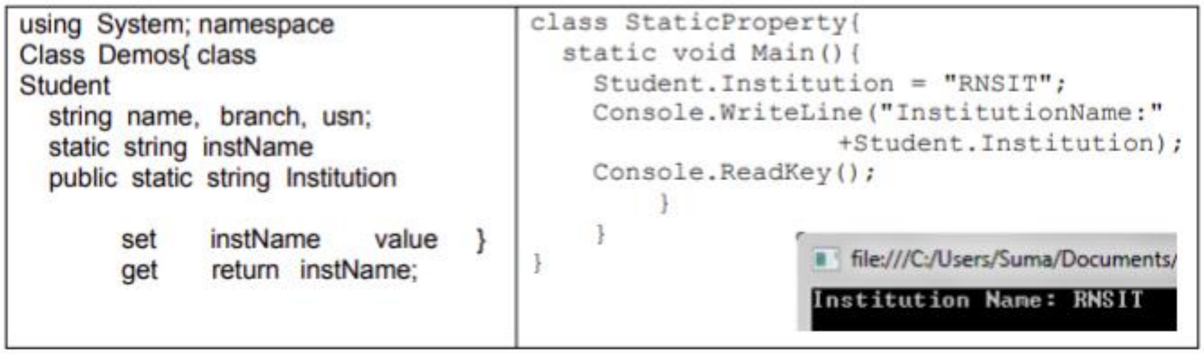
```
using System;
namespace Class_Demos{
 class Student{
   string studusn;
   public Student(){
     studusn="1RX12MCA01";
   }
   public string Studusn{
     get{
        return studusn;
     }
   }
 }
```

```
class ReadOnly{
  static void Main(){
     Student st1 = new Student();
     Console.WriteLine("USN: "
                     + st1.Studusn);
     Console.ReadKey();
  }
}
}
```

```
file:///C:/Users/Suma/
USN: 1RX12MCA01
```

"static" property: C# also supports "*static*" properties. The ***static members*** are accessed at the class level, not from an instance (object) of that class. Static properties are manipulated in the same manner as static methods, as seen here:

Example 2.5: Assume that the Student type defines a point of static data to represent the name of the institution studying these students. You may define a static property as follows:



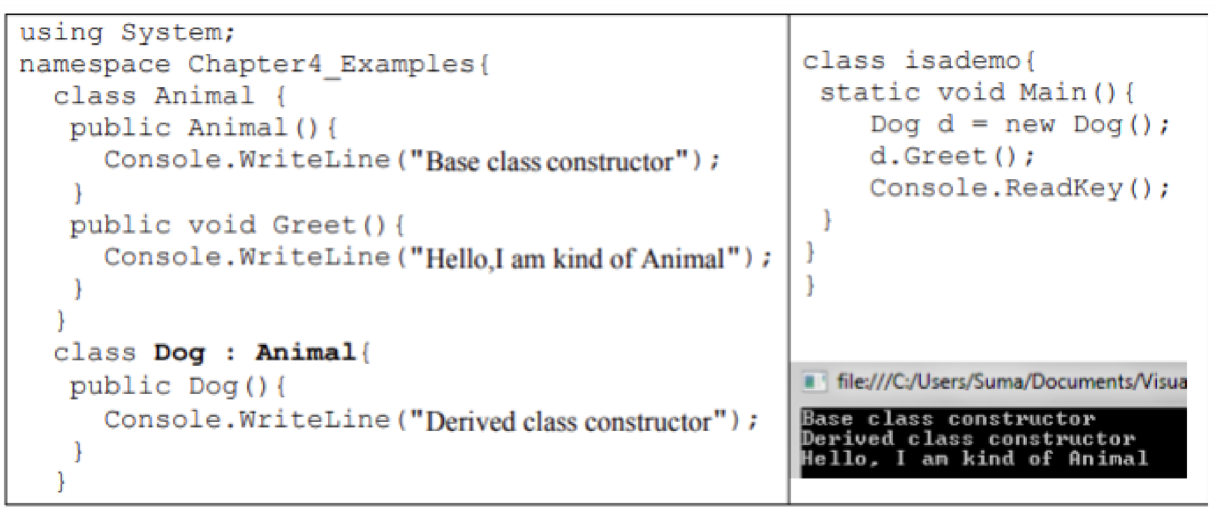
```
using System; namespace
Class Demos{ class
Student
  string name, branch, usn;
  static string instName
  public static string Institution

        set     instName     value    }
        get     return  instName;
```

```
class StaticProperty{
  static void Main(){
     Student.Institution = "RNSIT";
     Console.WriteLine("InstitutionName:"
                     +Student.Institution);
     Console.ReadKey();
        }
     }
}
```

```
file:///C:/Users/Suma/Documents/
Institution Name: RNSIT
```

4 a.

Classical inheritance ("is-a" relationship): When "is-a" relationship have established between classes, we are building a dependency between types. The basic idea behind classical inheritance is that new classes may extend the functionality of other classes. Assume that we wish to define two additional classes to model Animal and Dog. The hierarchy looks like as shown below and we notice that Animal "is-a" Mammal, Dog IS-A Animal; Hence dog IS-A mammal as well. In "is-a" model, base classes are used to define general characteristics that are common to all subclasses and classes are extended by using ":" operator. The derived classes inherit the base class's properties and methods and clients of the derived class have no knowledge of the base class.



```
using System;
namespace Chapter4_Examples{
  class Animal {
   public Animal(){
     Console.WriteLine("Base class constructor");
   }
   public void Greet(){
     Console.WriteLine("Hello,I am kind of Animal");
   }
  }
  class Dog : Animal{
   public Dog(){
     Console.WriteLine("Derived class constructor");
   }
  }
```

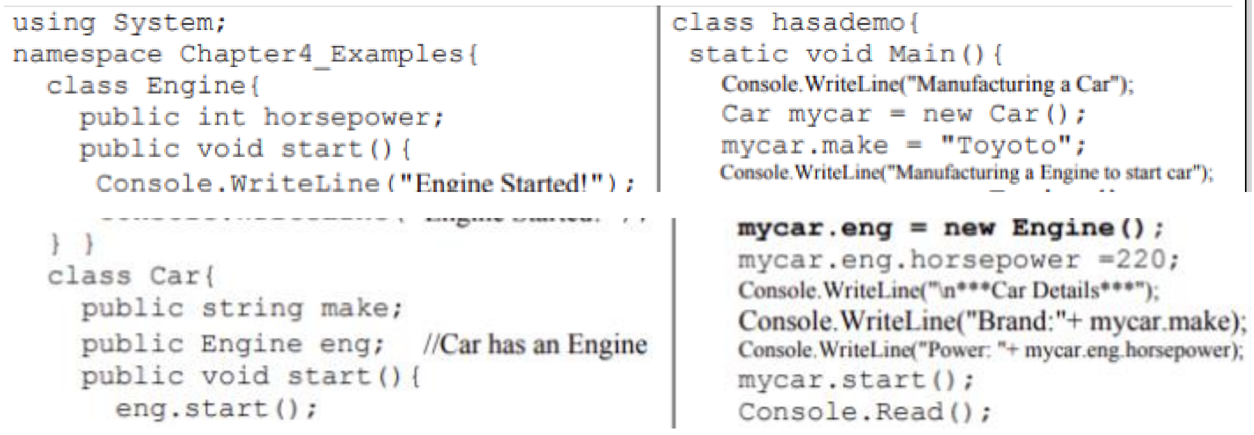
```
class isademo{
 static void Main(){
    Dog d = new Dog();
    d.Greet();
    Console.ReadKey();
  }
}
}
```

file:///C:/Users/Suma/Documents/Visua

```
Base class constructor
Derived class constructor
Hello, I am kind of Animal
```

Containment / Delegation model ("Has-A"): The "HAS-A" relationship specifies how one class is made up of other classes

Consider we have two different classes Engine and a Car when both of these entities share each other's object for some work and at the same time they can exists without each other's dependency(havingtheir own life time) and there should be no single owner both have to be an independent from each other than type of relationship is known as "has-a" relationship i.e. Association.



```
using System;
namespace Chapter4_Examples{
  class Engine{
    public int horsepower;
    public void start(){
     Console.WriteLine("Engine Started!");
  } }
  class Car{
    public string make;
    public Engine eng;   //Car has an Engine
    public void start(){
      eng.start();
```

```
class hasademo{
 static void Main(){
   Console.WriteLine("Manufacturing a Car");
   Car mycar = new Car();
   mycar.make = "Toyoto";
   Console.WriteLine("Manufacturing a Engine to start car");
    mycar.eng = new Engine();
    mycar.eng.horsepower =220;
    Console.WriteLine("\n***Car Details***");
    Console.WriteLine("Brand:"+ mycar.make);
    Console.WriteLine("Power: "+ mycar.eng.horsepower);
    mycar.start();
    Console.Read();
```

4 b.i

In C#, one is allowed to create a static class, by using *static* keyword. A static class can only contain static data members, static methods, and a static constructor.It is not allowed to create objects of the static class. Static classes are **sealed**, means ***you cannot inherit a static class from another class***.

Syntax:

static class Class_Name

{

```
    // static data members

   // static method

}
```

In C#, the static class contains two types of static members as follows:

- **Static Data Members:** As static class always contains static data members, so static data members are declared using static keyword and they are directly accessed by using the class name. The memory of static data members is allocating individually without any relation with the object.
  Syntax:

```
static class Class_name

{

   public static nameofdatamember;

}
```

- **Static Methods:** As static class always contains static methods, so static methods are declared using static keyword. These methods only access static data members, they can not access non-static data members.
  Syntax:

```
static class Class_name {


   public static nameofmethod()

   {

       // code

    }

}
```

Example 1:

- C#

```
// C# program to illustrate the
// concept of static class
using System;

namespace ExampleOfStaticClass {

// Creating static class
// Using static keyword
static class Author {

    // Static data members of Author
    public static string A_name = "Ankita";
    public static string L_name = "CSharp";
    public static int T_no = 84;

    // Static method of Author
    public static void details()
    {
        Console.WriteLine("The details of Author is:");
    }
}

// Driver Class
public class GFG {

    // Main Method
    static public void Main()
    {

        // Calling static method of Author
        Author.details();

        // Accessing the static data members of Author
        Console.WriteLine("Author name : {0} ", Author.A_name);
        Console.WriteLine("Language : {0} ", Author.L_name);
        Console.WriteLine("Total number of articles : {0} ",
                                        Author.T_no);
    }
}
}
```

Output:

```
The details of Author is:

Author name : Ankita

Language : CSharp

Total number of articles : 84
```

4 b.ii

Abstract Classes and Methods

Data **abstraction** is the process of hiding certain details and showing only essential information to the user.
Abstraction can be achieved with either **abstract classes** or **interfaces** (which you will learn more about in the next chapter).

The abstract keyword is used for classes and methods:

- **Abstract class:** is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).

- **Abstract method:** can only be used in an abstract class, and it does not have a body. The body is provided by the derived class (inherited from).

An abstract class can have both abstract and regular methods:

```
abstract class Animal

{

  public abstract void animalSound();

  public void sleep()

  {

    Console.WriteLine("Zzz");

  }

}
```

From the example above, it is not possible to create an object of the Animal class:

```
Animal myObj = new Animal(); // Will generate an error (Cannot create an instance of the abstract class or interface 'Animal')
```

To access the abstract class, it must be inherited from another class. Let's convert the Animal class we used in the Polymorphism chapter to an abstract class.

Remember from the Inheritance chapter that we use the : symbol to inherit from a class, and that we use the override keyword to override the base class method.

Example

```
// Abstract class

abstract class Animal

{

  // Abstract method (does not have a body)

  public abstract void animalSound();

  // Regular method

  public void sleep()

  {

    Console.WriteLine("Zzz");

  }

}



// Derived class (inherit from Animal)

class Pig : Animal

{

  public override void animalSound()

  {

    // The body of animalSound() is provided here

    Console.WriteLine("The pig says: wee wee");

  }

}



class Program
```

```
{
  static void Main(string[] args)
  {
    Pig myPig = new Pig(); // Create a Pig object
    myPig.animalSound();  // Call the abstract method
    myPig.sleep();  // Call the regular method
  }
}
```

5 a .

A delegate is a class that can hold a reference (source of information) to a method. We can say that delegate is like a pointer to a function.

Every delegate object is implicitly derived from class Delegate.

Declaration of Delegate

A delegate can be declared using delegate keyword followed by a function signature as shown below:

Signature of a Single Cast Delegate

<access modifier> delegate <return type> <delegate-name>(< parameter list >)

Example

```
public delegate void myDelegate(string myMessage);
```

Delegate class has four steps process for defining and using delegates – Declaration, Reference, Target and Invoke.

The first step is to declare the delegate with the return type and input parameters.

```
// Delegates declaration
public delegate int myDelegate(int a, int b);
Which is having same signature as method below.
public int Sum(int a, int b)
{
  return a + b;
}
```

The second step is to create a delegate reference.

```
// Create delegate reference
```

```
DelegateClass.myDelegate objSum = null;
```

The third step is to target/point the delegate reference to an object instance.

```
// Target the reference to the Sum method
objSum = objMyclass.Sum;
```

The final step is to invoke the delegate like invoking a regular method.

```
// Invoke the delegate
Console.WriteLine("Sum of two integer is = " + objSum(10, 20));
Console.ReadLine();
```

Finally the Sample Code

```
namespace DelegatesProperties
{
    class DelegateClass
    {
        // Delegates declaration
        public delegate int myDelegate(int a, int b);

        // Methods to Target & called by delegate
        public int Sum(int a, int b)
        {
            return a + b;
        }
    }

    class Program
    {

        static void Main(string[] args)
        {
            DelegateClass objMyclass = new DelegateClass();

            // Create delegate reference
            DelegateClass.myDelegate objSum = null;

            // Target/Point the reference to the Sum method
            objSum = objMyclass.Sum;

            // Invoke the delegate
            if (objSum != null)
            {
                Console.WriteLine("Sum of two integer is = " + objSum(10, 20));
                Console.ReadLine();
            }
        }
    }
}
```

Output

Sum of two integer is 30;

5 b. ADO.NET Architecture

ADO.NET uses a multilayer architecture that mainly has a few concepts, for instance Connection, Reader, Command, Adapter and Dataset objects. ADO.NET introduced data providers that are a set of special classes to access a specific database, execute SQL commands and retrieve data. The Data providers are extensible. Developers can create their own providers for a proprietary data source. There are some examples of data providers such as SQL Server providers, OLE DB and Oracle provider.

ADO.NET provides the following two types of classes objects:

- **Connection-based:** They are the data provider objects such as Connection, Command, DataAdapter, and DataReader. They execute SQL statements and connect to a database.

- **Content-based:** They are found in the System.Data namespace and includes DataSet, DataColumn, DataRow, and DataRelation. They are completely independent of the type of data source.

ADO.NET Namespaces

Namespaces	Description
System.Data	Contains the definition for columns,relations,tables,database,rows,views and constraints.
System.Data.SqlClient	Contains the classes to connect to a Microsoft SQL Server database such as SqlCommand, SqlConnection, and SqlDataAdapter.
System.Data.Odbc	Contains classes required to connect to most ODBC drivers. These classes include OdbcCommand and OdbcConnection.
System.Data.OracleClient	Contains classes such as OracleConnection and OracleCommand required to connect to an Oracle database.

Table 1.1 ADO.NET Namespace

Connection Class

You need to establish a connection class object for inserting, updating, deleting and retrieving data from a database. The Connection class allows you to establish a connection to the data source. The Connection class object needs the necessary information to discover the data source and this information is provided by a connection string.

Connection Strings

You need to supply a connection string in the Connection class object. The connection string is a series of name/value settings separated by semicolons (;). A connection string requires a few pieces of information such as the location of the database, the database name, and the database authentication mechanism.

This connection is used to connect to the Master database on the current computer using integrated security (indicating the currently logged-in Windows user can access the database).

C# Code

```
string conString = "Data Source=localhost;Initial Catalog=Master;Integrated Security=SSPI";
```

In the case of integrated security, it is not supported and then the connection must indicate a valid user name and password combination as in the following.

C# Code

```
string conString = "Data Source=localhost;Database=Master;user id=sa;password=sa";
```

If you use the OLE DB provider than your connection string will need some additional settings that identify OLE DB drivers such as:

C# Code

```
string conString = "Data Source=localhost;Initial Catalog=Master;user id=sa;password=;Provider=MSDAORA";
```

You can specify the details of the connection strings in the global application settings file and then you can retrieve your connection string by name from the ConfigurationManager as in the following.

App.Config

```
<configuration>
    <connectionStrings>
        <add name="Master" connectionString ="Data Source=localhost;Initial Catalog=Master;Integrated Security=SSPI" />
    </connectionStrings>
</configuration>
```

Once you declare all the details in the App.config file that pertains to the Connection string then you can use this definition in the code file also as in the following.

C# Code

```
string conSting = ConfigurationManager.ConnectionStrings["Master"].ConnectionString ;
SqlConnection Conn = new SqlConnection(conSting);
```

Testing a Connection

Once you have configured the right connection string to establish connectivity with the specific data source you simply use the Open() and Close() methods as in the following.

C# Code

```
private void Form1_Load(object sender, EventArgs e)
{
   string conSting =       ConfigurationManager.ConnectionStrings["Master"].Con
nectionString ;
   SqlConnection Conn = new SqlConnection(conSting);

   try
   {
      Conn.Open();
      textBox1.Text = "Server Version=" + Conn.ServerVersion;
      textBox1.Text += "Connection Is=" + Conn.State.ToString();
   }
   catch (Exception err)
   {
      textBox1.Text = err.Message;
   }
   finally
   {
      Conn.Close();
      textBox1.Text += "Connection Is=" + Conn.State.ToString();
   }
}
```

You can also use the SqlConnectionStringBuilder class to configure a connection string rather than specifying it in the App.Config file as in the following.

C# Code

```
SqlConnectionStringBuilder obj = new SqlConnectionStringBuilder();
obj.DataSource = "localhost";
obj.InitialCatalog = "Master";
obj.IntegratedSecurity = true;
SqlConnection Conn = new SqlConnection(obj.ConnectionString);
```

Important: Connections are a limited server resource so it is imperative to release the open connection as soon as possible.

Command and Data Reader Classes

The Command Class allows performing any data definition tasks such as creating and altering tables and databases, retrieving, updating and deleting of records. The Command object used to execute SQL queries can be inline text or a Stored Procedure. It is all dependent on the type of command you are using. Before using the command, you need to configure the Command Type, Text and Connection properties as in the following.

C# Code

```
//Command Class definition
```

```csharp
SqlCommand sc = new SqlCommand();
sc.Connection = Conn;
sc.CommandType = CommandType.Text;
sc.CommandText = query;
```

Alternatively, you can pass the connection argument directly to the Command class as in the following.

C# Code

```csharp
//Command Class definition
SqlCommand sc = new SqlCommand(query,Conn);
```

In the following example, we are creating a window application form with a Text Box control. We are establishing a connection to the Customer table from the AdventureWorks database. Then, using the SqlDataReader class, we will go through all the records of the table and display the FirstName and LastName in the TextBox control by executing the While() loop as in the following.

C# Code

```csharp
private void Form1_Load(object sender, EventArgs e)
{
    //Connection String
    SqlConnectionStringBuilder obj = new SqlConnectionStringBuilder();
    obj.DataSource = "localhost";
    obj.InitialCatalog = "AdventureWorksLT2008";
    obj.IntegratedSecurity = true;

    // Add Connection string to SqlConnection
    SqlConnection Conn = new SqlConnection(obj.ConnectionString);

    // Query to retrieve records from AdventureWorks Database
    string query = "select FirstName,LastName from SalesLT.Customer";

    //Command Class definition
    SqlCommand sc = new SqlCommand();
    sc.Connection = Conn;
    sc.CommandType = CommandType.Text;
    sc.CommandText = query;

    SqlDataReader sdr = null;
    try
    {
       //Open connection
        Conn.Open();
        sdr = sc.ExecuteReader();

        //Get all records
        while(sdr.Read())
        {
            textBox1.AppendText(sdr.GetValue(0) + "\t" + sdr.GetValue(1));
            textBox1.AppendText("\n");
        }

    }
    catch (Exception err)
    {
        textBox1.Text = err.Message;
    }
    finally
    {
        //Release reader and connection object
        sdr.Close();
        Conn.Close();
    }
}
```

It is important to release the objects of the Reader class manually after the job is done or you can set the CommandBehaviour Property to CloseConnection in the ExcuteReader() method to avoid the burden of manually releasing the object as in the following.

C# Code

```
//Automatically releasing the Reader class Object
sdr = sc.ExecuteReader(CommandBehavior.CloseConnection);
```

DataReader Class

The DataReader Class object allows you to read the data returned by a SELECT command by a simple forward-only and read-only cursor. It requires a live connection with the data source and provides a very efficient way of looping and consuming all parts of the result set. The object of the DataReader cannot be directly instantiated. Instead you must call the ExecuteReader method of the Command object and close the connection when you are done using the Data Reader, otherwise the connection remains alive until it is explicitly closed.

DataReader with ExecuteReader() Method

Once you have the DataReader you can cycle through its records by calling the Read() method in a while loop. This moves the row cursor to the next record.

C# Code

```
//Open connection
Conn.Open();
sdr = sc.ExecuteReader(CommandBehavior.CloseConnection);

//Get all records
```

```
while(sdr.Read())
{
  textBox1.AppendText(sdr.GetValue(0) + "\t" + sdr.GetValue(1));
  textBox1.AppendText("\n");
}
```

ExecuteScalar() Method

The ExecuteScalar() method returns the value stored in the first field of the first row of a result set generated by the command's SELECT query. This method is usually used to count the total number of rows in the table as in the following.

C# Code

```
private void Form1_Load(object sender, EventArgs e)
{
    //Connection String
    string conString = @"Data Source=localhost;Database=AdventureWorksLT2008
;Integrated Security=SSPI";

    // Add Connection string to SqlConnection
    SqlConnection Conn = new SqlConnection(conString);

    // Query to retrieve records from AdventureWorks Database
    string query = "select COUNT(*) from SalesLT.Customer";

    //Command Class definition
    SqlCommand sc = new SqlCommand(query, Conn);

    //Open connection
    Conn.Open();
    int CountCustomer = (int)sc.ExecuteScalar();

    //Count all records
    textBox1.AppendText("Total Customer=\t" + CountCustomer.ToString());
}
```

ExecuteNonQuery() Method

The ExecuteNonQuery() method executes commands that don't return a result set, for instance INSERT, UPDATE and DELETE. Here in this example we made a modification to a specific record in the Customer table of the Adventure Works database.

C# Code

```
private void Form1_Load(object sender, EventArgs e)
{
    //Connection String
    string conString = @"Data Source=localhost;Database=AdventureWorksLT2008
;Integrated Security=SSPI";

    // Add Connection string to SqlConnection
    SqlConnection Conn = new SqlConnection(conString);

    // Query to retrieve records from AdventureWorks Database
    string query = @"update AdventureWorksLT2008.SalesLT.Customer
                set FirstName='ajay'
                where CustomerID=2";

    //Command Class definition
    SqlCommand sc = new SqlCommand(query, Conn);

    //Open connection
    Conn.Open();

    //Reflect changes into database
    int CountCustomer = sc.ExecuteNonQuery();

    //Result
    MessageBox.Show("Record Update Successfully");
}
```

DataAdapter and DataTable class

A DataAdapter bridges the gap between the disconnected DataTable objects and the physical data source. The SqlDataAdapter is capable of executing a SELECT, DELETE and UPDATE statement on a data source as well as extracting input from the result set into a DataTable object. The SqlDataAdapter class provides a method called Fill() to copy the result set into the DataTable.

C# Code

```
private void Form1_Load(object sender, EventArgs e)
{
    //Connection String
    string conString = "Data Source=localhost;Database=AdventureWorksLT2008;I
ntegrated Security=SSPI";

    // Add Connection string to SqlConnection
    SqlConnection Conn = new SqlConnection(conString);

    // Query to retrieve records from AdventureWorks Database
    string query = "select FirstName,LastName from SalesLT.Customer";

    //Command Class definition
    SqlCommand sc = new SqlCommand(query, Conn);

    // Data Adapter definition
    SqlDataAdapter sda = new SqlDataAdapter(sc);

    // filling the result set in data table
    DataTable dt = new DataTable();
    sda.Fill(dt);

    //output in data grid
    dataGridView1.DataSource = dt.DefaultView;
}
```

These are the commonly used properties offered by the **SqlDataAdapter** class as in the following:

Property	Description
SelectCommand	This command executed to fill in a Data Table with the result set.
InsertCommand	Executed to insert a new row to the SQL database.
UpdateCommand	Executed to update an existing record on the SQL database.
DeleteCommand	Executed to delete an existing record on the SQL database.

Table 1.2 Data Adapter Properties

SelectCommand Example

C# Code

```
// Query to retrieve records from AdventureWorks Database
string query = "select FirstName,LastName from SalesLT.Customer";

//Command Class definition
SqlCommand sc = new SqlCommand(query, Conn);

// Data Adapter definition
SqlDataAdapter sda = new SqlDataAdapter();
sda.SelectCommand = sc;

// filling the result set in data table
DataTable dt = new DataTable();
sda.Fill(dt);
```

Update Command Example

C# Code

```
string query = @"update AdventureWorksLT2008.SalesLT.Customer
set FirstName='ajay'
where CustomerID=2";

//Command Class definition
```

```
SqlCommand sc = new SqlCommand(query, Conn);

// Data Adapter definition
SqlDataAdapter sda = new SqlDataAdapter();
sda.UpdateCommand = sc;
```

Parameterized Commands (Stored Procedure)

A Stored Procedure is a batch of one or more SQL statements that are stored in the database. They are similar to a function in that they are well-encapsulated blocks of the logic that accept data using an input parameter and return data via result set or output parameter. The following is the SQL code needed to create a procedure for extracting a single something from the customer table on behalf of a specific CustomerID.

Sql.script

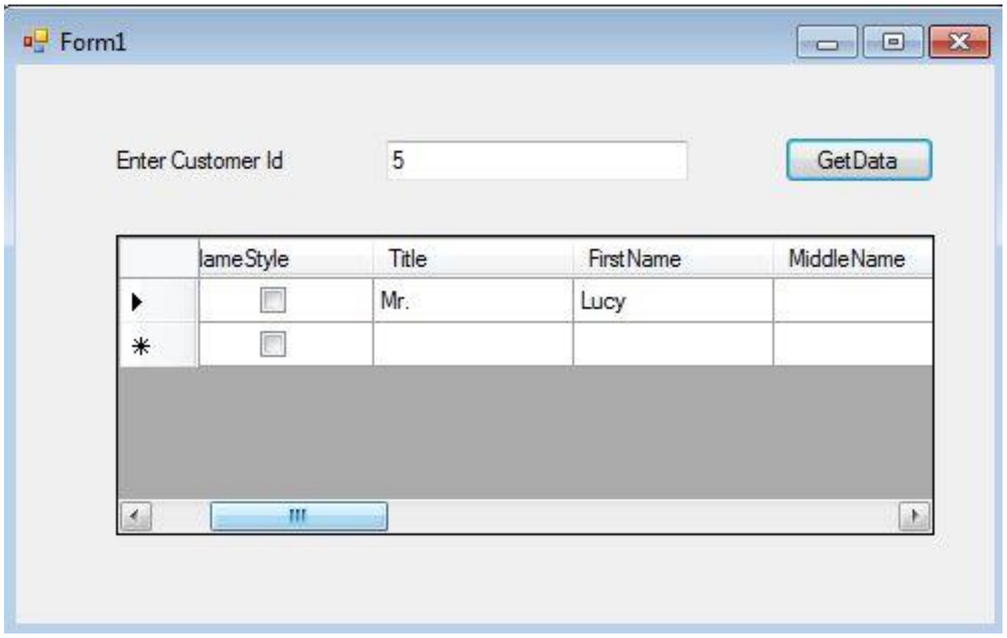
```
Create Proc GetCustomer
@CustID varchar(10)
AS
select * from SalesLT.Customer where CustomerID=@CustID
GO
```

Next, you can create a SqlCommand to wrap the call to the Stored Procedure. This command takes one parameter as input and returns the records. A parameterized command is basically something that uses a placeholder in the SQL text. The placeholder indicates dynamically supplied values that are then sent using a parameters collection of the Command object as in the following.

C# Code

```
private void btnData_Click(object sender, EventArgs e)
{
    //Connection String
    string conString = "Data Source=localhost;Database=AdventureWorksLT2008;Integrated Security=SSPI";

    // Add Connection string to SqlConnection
    SqlConnection Conn = new SqlConnection(conString);

    //Command Class definition
    SqlCommand sc = new SqlCommand("GetCustomer", Conn);
    sc.CommandType = CommandType.StoredProcedure;

    sc.Parameters.Add("@CustID",txtParameter.Text);

    // Data Adapter definition
    SqlDataAdapter sda = new SqlDataAdapter(sc);

    // filling the result set in data table
    DataTable dt = new DataTable();
    sda.Fill(dt);

    //output in data grid
    dataGridView1.DataSource = dt.DefaultView;
}
```

This example uses a parameterized command that is supplied via a text box (Customer ID) and the result is processed using the Stored Procedure in the code file and the result is displayed in the Data Grid View control as in the following:



Enter Customer Id: 5

lameStyle	Title	FirstName	MiddleName
☐	Mr.	Lucy	
☐			

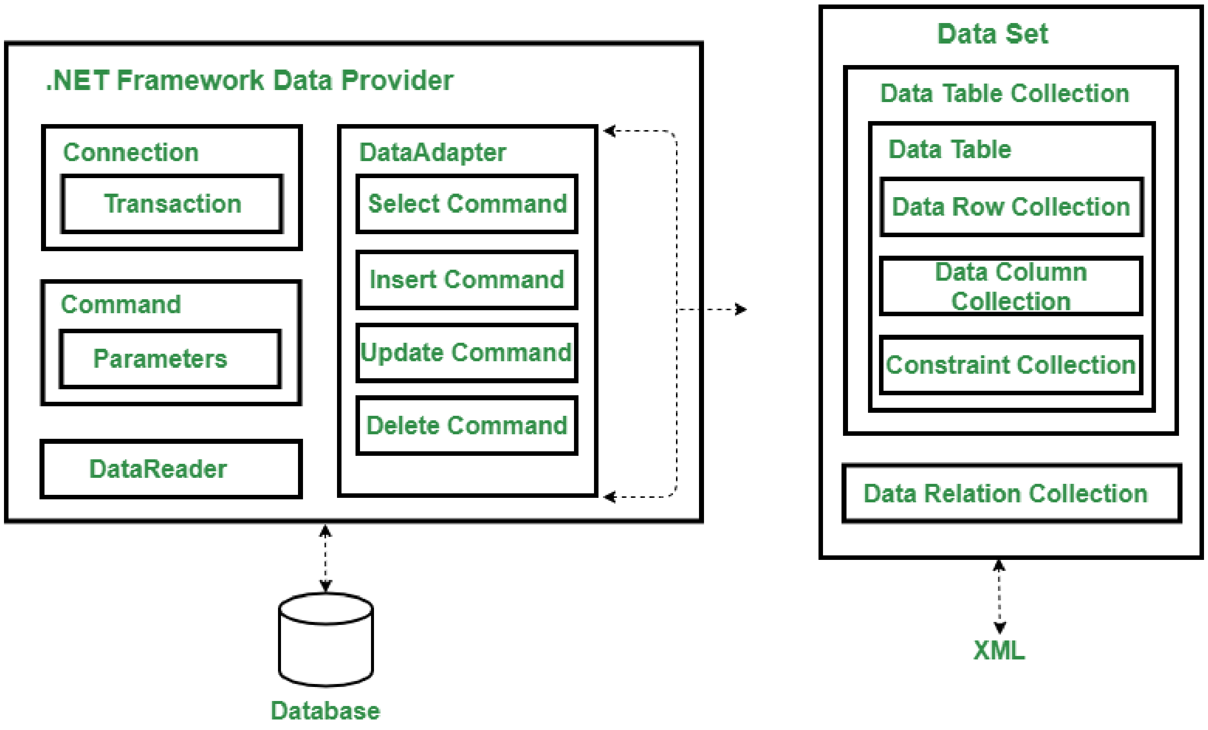
DataSet class

A DataSet is a disconnected architecture technology. It contains zero or more tables and relationships. When you work with a dataset, the data in the data source is not touched at all. Instead, all the changes are made locally to the dataset in memory. In the following example, you will see how to retrieve data from a SQL Server table and use it to fill in a DataTable object in the DataSet.

C# Code

```
private void Form1_Load(object sender, EventArgs e)
{
    //Connection String
    string conString = "Data Source=localhost;Database=AdventureWorksLT2008;Integrated Security=SSPI";

    // Add Connection string to SqlConnection
    SqlConnection Conn = new SqlConnection(conString);

    string query = "select * from SalesLT.Customer";

    //Command Class definition
    SqlCommand sc = new SqlCommand(query, Conn);

    // Data Adapter definition
    SqlDataAdapter sda = new SqlDataAdapter();
    sda.SelectCommand = sc;

    //data Set definition
    DataSet ds = new DataSet();

    // filling the result set in data table
    sda.Fill(ds, "SalesLT.Customer");

    //output in data grid
    dataGridView1.DataSource = ds.Tables["SalesLT.Customer"];
}
```

Here you need to create an empty DataSet and use the SqlDataAdapter Fill() method to execute the query and place the results in a new DataTable in the DataSet.



6 a.

```
6 a using System;
```

```
namespace ErrorHandlingApplication {
  class DivNumbers {
    int result;

    DivNumbers() {
      result = 0;
    }
    public void division(int num1, int num2) {
      try {
        result = num1 / num2;
      } catch (DivideByZeroException e) {
        Console.WriteLine("Exception caught: {0}", e);
      } finally {
        Console.WriteLine("Result: {0}", result);
      }
    }
    static void Main(string[] args) {
      DivNumbers d = new DivNumbers();
      d.division(25, 0);
      Console.ReadKey();
    }
  }
}
```

When the above code is compiled and executed, it produces the following result −

```
Exception caught: System.DivideByZeroException: Attempted to divide by zero.
at ...
Result: 0
```

6 b .

DataAdapter is used to retrieve data from a data source and populate tables within a DataSet. The DataAdapter also resolves changes made to the DataSet back to the data source. The DataAdapter uses the Connection object of the .NET Framework data provider to connect to a data source, and it uses Command objects to retrieve data from and resolve changes to the data source.

Each .NET Framework data provider included with the .NET Framework has a DbDataReader and a DbDataAdapter object: the .NET Framework Data Provider for OLE DB includes an OleDbDataReader and an OleDbDataAdapter object, the .NET Framework Data Provider for SQL Server includes a SqlDataReader and a SqlDataAdapter object, the .NET Framework Data Provider for ODBC includes an OdbcDataReader and an OdbcDataAdapter object, and the .NET Framework Data Provider for Oracle includes an OracleDataReader and an OracleDataAdapter object.

DataReader Class

The DataReader Class object allows you to read the data returned by a SELECT command by a simple forward-only and read-only cursor. It requires a live connection with the data source and provides a very efficient way of looping and consuming all parts of the result set. The object of the DataReader cannot be directly instantiated. Instead you must call the ExecuteReader method of the Command object and close the connection when you are done using the Data Reader, otherwise the connection remains alive until it is explicitly closed.

DataReader with ExecuteReader() Method

Once you have the DataReader you can cycle through its records by calling the Read() method in a while loop. This moves the row cursor to the next record.

C# Code

```
//Open connection
Conn.Open();
sdr = sc.ExecuteReader(CommandBehavior.CloseConnection);

//Get all records
while(sdr.Read())
{
  textBox1.AppendText(sdr.GetValue(0) + "\t" + sdr.GetValue(1));
  textBox1.AppendText("\n");
}
```

ExecuteScalar() Method

The ExecuteScalar() method returns the value stored in the first field of the first row of a result set generated by the command's SELECT query. This method is usually used to count the total number of rows in the table as in the following.

C# Code

```
private void Form1_Load(object sender, EventArgs e)
{
    //Connection String
    string conString = @"Data Source=localhost;Database=AdventureWorksLT2008
;Integrated Security=SSPI";

    // Add Connection string to SqlConnection
    SqlConnection Conn = new SqlConnection(conString);

    // Query to retrieve records from AdventureWorks Database
    string query = "select COUNT(*) from SalesLT.Customer";

    //Command Class definition
    SqlCommand sc = new SqlCommand(query, Conn);

    //Open connection
    Conn.Open();
    int CountCustomer = (int)sc.ExecuteScalar();

    //Count all records
    textBox1.AppendText("Total Customer=\t" + CountCustomer.ToString());
}
```

ExecuteNonQuery() Method

The ExecuteNonQuery() method executes commands that don't return a result set, for instance INSERT, UPDATE and DELETE. Here in this example we made a modification to a specific record in the Customer table of the Adventure Works database.

C# Code

```
private void Form1_Load(object sender, EventArgs e)
{
    //Connection String
    string conString = @"Data Source=localhost;Database=AdventureWorksLT2008
;Integrated Security=SSPI";

    // Add Connection string to SqlConnection
    SqlConnection Conn = new SqlConnection(conString);

    // Query to retrieve records from AdventureWorks Database
    string query = @"update AdventureWorksLT2008.SalesLT.Customer
                    set FirstName='ajay'
                    where CustomerID=2";

    //Command Class definition
    SqlCommand sc = new SqlCommand(query, Conn);

    //Open connection
    Conn.Open();

    //Reflect changes into database
    int CountCustomer = sc.ExecuteNonQuery();

    //Result
    MessageBox.Show("Record Update Successfully");
}
```

DataAdapter and DataTable class

A DataAdapter bridges the gap between the disconnected DataTable objects and the physical data source. The SqlDataAdapter is capable of executing a SELECT, DELETE and UPDATE statement on a data source as well as extracting input from the result set into a DataTable object. The SqlDataAdapter class provides a method called Fill() to copy the result set into the DataTable.

C# Code

```
private void Form1_Load(object sender, EventArgs e)
{
    //Connection String
    string conString = "Data Source=localhost;Database=AdventureWorksLT2008;Integrated Security=SSPI";

    // Add Connection string to SqlConnection
    SqlConnection Conn = new SqlConnection(conString);

    // Query to retrieve records from AdventureWorks Database
    string query = "select FirstName,LastName from SalesLT.Customer";

    //Command Class definition
    SqlCommand sc = new SqlCommand(query, Conn);

    // Data Adapter definition
    SqlDataAdapter sda = new SqlDataAdapter(sc);

    // filling the result set in data table
    DataTable dt = new DataTable();
    sda.Fill(dt);

    //output in data grid
    dataGridView1.DataSource = dt.DefaultView;
}
```

These are the commonly used properties offered by the **SqlDataAdapter** class as in the following:

Property	Description
SelectCommand	This command executed to fill in a Data Table with the result set.
InsertCommand	Executed to insert a new row to the SQL database.
UpdateCommand	Executed to update an existing record on the SQL database.
DeleteCommand	Executed to delete an existing record on the SQL database.

Table 1.2 Data Adapter Properties

SelectCommand Example

C# Code

```
// Query to retrieve records from AdventureWorks Database
string query = "select FirstName,LastName from SalesLT.Customer";

```

```
//Command Class definition
SqlCommand sc = new SqlCommand(query, Conn);

// Data Adapter definition
SqlDataAdapter sda = new SqlDataAdapter();
sda.SelectCommand = sc;

// filling the result set in data table
DataTable dt = new DataTable();
sda.Fill(dt);
```

Update Command Example

C# Code

```
string query = @"update AdventureWorksLT2008.SalesLT.Customer
set FirstName='ajay'
where CustomerID=2";

//Command Class definition
SqlCommand sc = new SqlCommand(query, Conn);

// Data Adapter definition
SqlDataAdapter sda = new SqlDataAdapter();
sda.UpdateCommand = sc;
```

7 a. Mouse Events

There are several kinds of mouse events: (list from an MSDN page)

- Click

  This event occurs when the mouse button is released, typically before the MouseUp event. The handler for this event receives an argument of type EventArgs. Handle this event when you only need to determine when a click occurs.

- MouseClick

  This event occurs when the user clicks the control with the mouse. The handler for this event receives an argument of type MouseEventArgs. Handle this event when you need to get information about the mouse when a click occurs.

- DoubleClick

This event occurs when the control is double-clicked. The handler for this event receives an argument of type EventArgs. Handle this event when you only need to determine when a double-click occurs.

- MouseDoubleClick

  This event occurs when the user double-clicks the control with the mouse. The handler for this event receives an argument of type MouseEventArgs. Handle this event when you need to get information about the mouse when a double-click occurs.

- MouseDown

  This event occurs when the mouse pointer is over the control and the user presses a mouse button. The handler for this event receives an argument of type MouseEventArgs.

- MouseEnter

  This event occurs when the mouse pointer enters the border or client area of the control, depending on the type of control. The handler for this event receives an argument of type EventArgs.

- MouseHover

  This event occurs when the mouse pointer stops and rests over the control. The handler for this event receives an argument of type EventArgs.

- MouseLeave

  This event occurs when the mouse pointer leaves the border or client area of the control, depending on the type of the control. The handler for this event receives an argument of type EventArgs.

- MouseMove

  This event occurs when the mouse pointer moves while it is over a control. The handler for this event receives an argument of type MouseEventArgs.

- MouseUp

  This event occurs when the mouse pointer is over the control and the user releases a mouse button. The handler for this event receives an argument of type MouseEventArgs.

- MouseWheel

This event occurs when the user rotates the mouse wheel while the control has focus. The handler for this event receives an argument of type MouseEventArgs. You can use the Delta property of MouseEventArgs to determine how far the mouse has scrolled.

As you may gather, EventArgs provides less information for us than MouseEventArgs.

MouseEventArgs Class

This gives us information about what has just happened with the mouse.

There is a constructor, but it does not seem useful.

Properties: (all "get" only)

- X = x-coordinate of the mouse (integer)
- Y = y-coordinate of the house (integer)
- Location = location of the mouse (Point, hence an x-y pair)
- Button = which mouse button was pressed (a member of the MouseButtons enumeration: Left, Right, Middle, None, etc.)
- Clicks = number of times the button was pressed and released (integer)
- Delta = the number of notches ("detents") by which the mouse has been rotated (integer)

How do we use this?

Suppose we want to handle MouseDown events. We will need a using statement for System.Windows.Forms.

In the InitializeComponent method, we need a line:

```
this.MouseDown += new MouseEventHandler(this.Form_MouseDown);
```

The event handler might look like this:

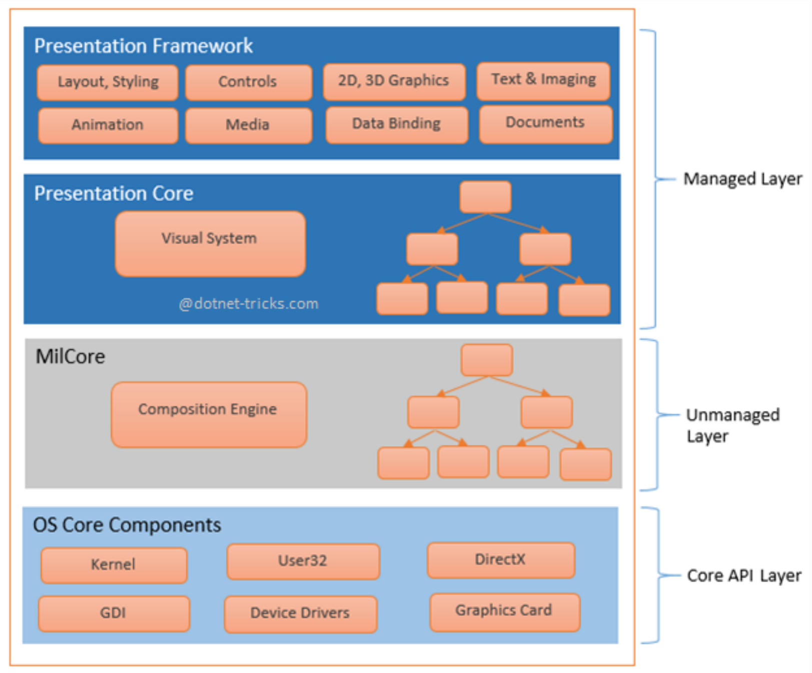
```
private void Form_MouseDown(object sender, MouseEventArgs e)
{
 switch (e.Button)
 {
  case MouseButtons.Left:
     MessageBox.Show("Left Button Click");
     break;
  case MouseButtons.Right:
     MessageBox.Show("Right Button Click");
```

```
        break;
    default:
        break;
  }
}
```

In the Designer file, we need a line to attach the handler to the event:

```
this.MouseDown +=
      new System.Windows.Forms.MouseEventHandler(this.Form_MouseClick);
```

7 b . Windows Presentation Framework is a next generation UI framework to create applications with a rich user experience. It is part of the .NET framework 3.0 and higher. WPF architecture is a layered architecture which have Managed, Unmanaged and Core API layers as shown in below fig.



WPF Architecture

1. Managed Layer

Managed layer has two main components – Presentation Framework and Presentation Core.

1. Presentation Framework provides the required functionalities that we need to build the WPF applications such as controls, data bindings, styling, shapes, media, documents, annotations, animation and more. PresentationFamework.dll is responsible for this purpose.

2. Presentation Core acts as a managed wrapper around MILCore and provides public interface for MIL. Presentation Core is the home for WPF Visual System and provides classes for creating application visual tree. The Visual System creates visual tree which contains applications Visual Elements and rendering instructions. PresentationCore.dll is responsible for this purpose.

2. Unmanaged Layer

This layer is also called milcore or Media Integration Library Core. MilCore is written in unmanaged code in order to enable tight integration with DirectX. DirectX engine is underlying technology used in WPF to display all graphics, allowing for efficient hardware and software rendering. MIL has Composition System that receives rendering instructions from Visual System and translates into data that can be understood by DirectX to render user interface.

3. Core API Layer

This layer has OS core components like Kernel, User32, GDI, Device Drivers, Graphic cards etc. These components are used by the application to access low level APIs. User32 manages memory and process separation.

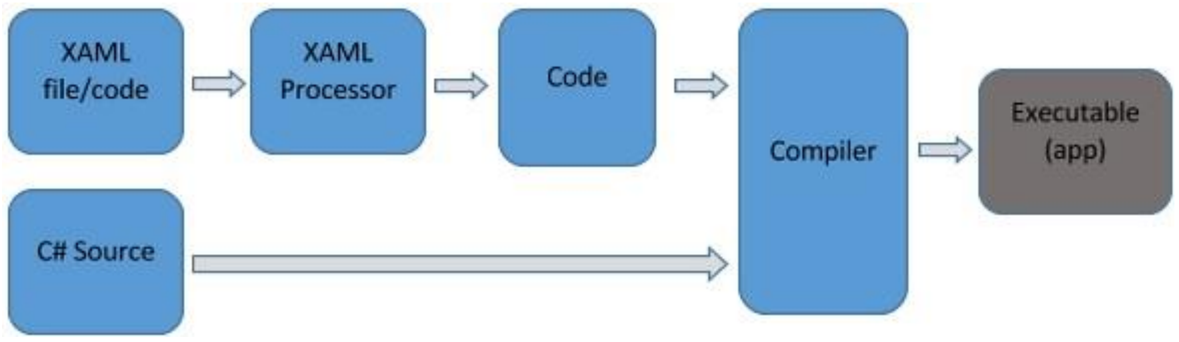
8 a . XAML stands for Extensible Application Markup Language. It's a simple and declarative language based on XML.

- In XAML, it very easy to create, initialize, and set properties of an object with hierarchical relations.
- It is mainly used for designing GUIs.
- It can be used for other purposes as well, e.g., to declare workflow in Workflow Foundation.

XAML can be used in different platforms such as WPF (Windows Presentation Foundation), Silverlight, Mobile Development, and Windows Store App. It can be used across different .Net framework and CLR (common language runtime) versions.

How XAML Works

XAML is a **declarative** language in the sense it defines the **WHAT** and **HOW** you want to do. XAML processor is responsible for the **HOW** part to find out. Let's have a look at the following schema. It sums up the XAML side of things −



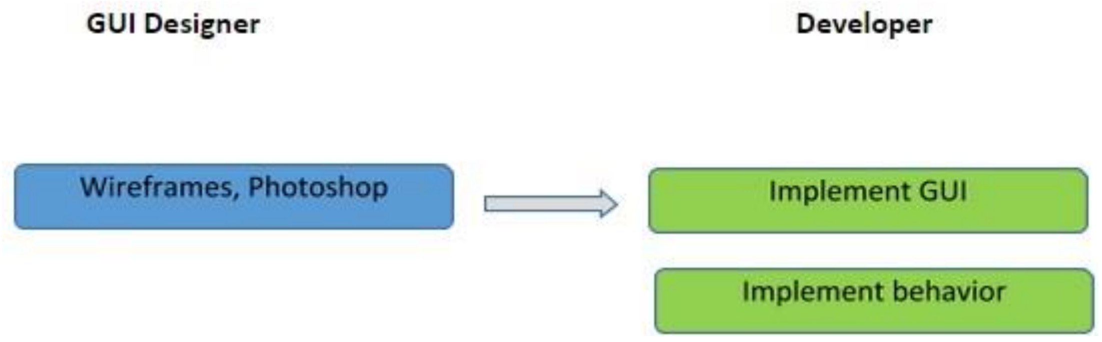
The figure illustrates the following actions −

- The XAML file is interpreted by a platform-specific XAML processor.
- The XAML processor transforms the XAML to internal code that describes the UI element.
- The internal code and the C# code are linked together through partial classes definitions and then the .NET compiler builds the app.

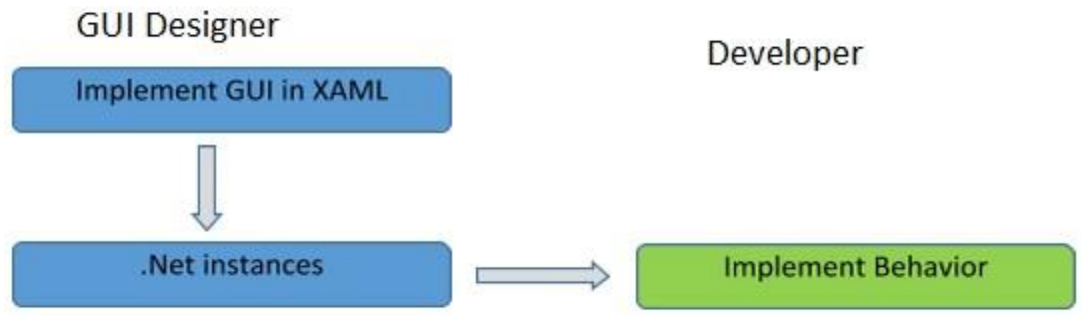
Advantages of XAML

One of the longstanding problems that all of us face with GUI design can be solved by using XAML. It can be used to design UI elements in Windows Forms applications.

In the earlier GUI frameworks, there was no real separation between how an application looks like and how it behaves. Both the GUI and its behavior were created in the same language, e.g. C# or VB.net, which would require more effort from the developer to implement both the UI and the behavior associated with it.



With XAML, it is very easy to separate the behavior from the designer code. Hence, the XAML programmer and the designer can work in parallel. XAML codes are very easy to read and understand.



XAML - Environment Setup

The Root Elements

Each XAML document must have a root element. The root element usually works as a container and defines the namespaces and basic properties of the element. Three most common root elements are <Windows />, <Page />, and <UserControl >. The <ResourceDirectory /> and <Application /> are other two root elements that can be used in a XAML file.

The Window element represents a Window container. The following code snippet shows a Window element with its Height, Width, Title and x:Name attributes. The x:Name attribute of an element represents the ID of an element used to access the element in the code-behind. The code snippet also sets xmlns and xmlns:x attributes that represent the namespaces used in the code. The x:Class attribute represents the code-behind class name.

```
<Window x:Class="HelloXAML.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="MainWindow" Height="350" Width="525">
</Window>
```

The Page element represents a page container. The following code snippet creates a page container. The code also sets the FlowDirection attribute that represents the flow direct of the contents of the page.

```
<Page
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
x:Class="WPFApp.Page1"
x:Name="Page"
WindowTitle="Page"
FlowDirection="LeftToRight"
Width="640" Height="480"
WindowWidth="640" WindowHeight="480">
</Page>
```

The UserControl element represents a user control container. The following code snippet represents a user control container.

```
<UserControl x:Class="HelloXAML.Page"
```

```
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Width="400" Height="300">
</UserControl>
```

XAML Namespaces

The part of the root element of each XAML are two or more attributes pre-fixed with xmlns and xmlns:x.

```
<Window
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
…… >
</Window>
```

The xmlns attribute indicates the default XAML namespace so the object elements in used in XAML can be specified without a prefix. Additionally, the x: prefix is used with more than just the namespace. Here are some common x:prefix syntaxes that are used in XAML.

1. x:Key: Sets a unique key for each resource in a ResourceDictionary.
2. x:Class: Class name provides code-behind for a XAML page.
3. x:Name: Unique run-time object name for the instance that exists in run-time code after an object element is processed.
4. x:Static: Enables a reference that returns a static value that is not otherwise a XAML-compatible property.
5. x:Type: Constructs a Type reference based on a type name.

Elements and Attributes

A type in WPF or Windows RT is represented by an XAML element. The <Page> and <Button> elements represent a page and a button control respectively. The XAML Button element listed in the following code represents a button control.

<Button />

Each of the elements such as <Page> or <Button> have attributes that can be set within the element itself. An attribute of an element represents a property of the type. For example, a Button has Height, Width, Background and Foreground properties that represent the height, width, foreground color and background color of the button respectively. The Content property of the Button represents the text of a button control. The x:Name property represents the unique ID of a control that may be used to access a control in the code behind.

The following code snippet sets the ID, height, width, background color, foreground color, font name and size and content of a button control.

```
<Button Content="Click Me" Width="200" Height="50"
Background="Orange" Foreground="Blue"
FontSize="20" FontFamily="Georgia" FontWeight="Bold"
x:Name="ClickButton">
</Button>
```

Content Property

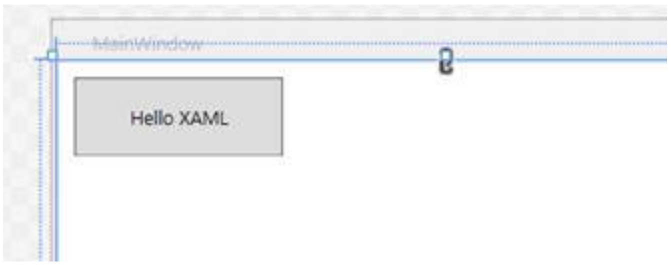
Each XAML object element is capable of displaying different content types. XAML provides a special property called Content that works to display the content of the element depending on the element capabilities. For example, a Content property of a Button can be a set to a string, an object, a UIElement, or even and container. However, the Content property of a ListBox is set using the Items property.

Note: Some XAML object elements may not have the Content property available directly. It must be set through a property.

The code snippet in the following code creates a Button control and sets its Content property to a string "Hello XAML".

```
<Button Height="50" Margin="10,10,350,310" Content="Hello XAML" />
Here is an alternative way to set the Content property of a Button.
<Button Height="50" Margin="10,10,350,310">Hello XAML</Button>
```

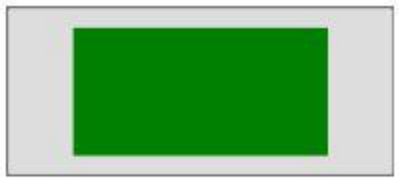
The output of the above code looks like the following.



A Button element can display other child elements as its content. The following code listed sets a Rectangle element as the content of the Button.

```
<Button Height="80" Margin="10,80,300,170">
<Rectangle Height="60" Width="120" Fill="Green"/>
</Button>
```

The output looks like the following.

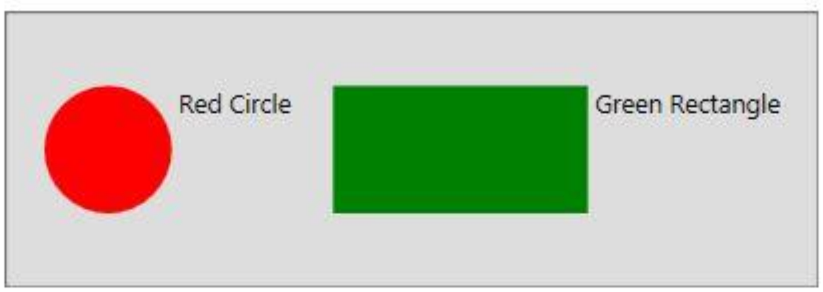


Content property can also be a container or a parent element hosting child elements. The code listed below sets a StackPanel container with 5 child elements as the content of the Button.

```
<Button Margin="10,201,100,40">
<StackPanel Orientation="Horizontal">
<Ellipse Height="60" Width="60" Fill="Red"/>
```

```
<TextBlock TextAlignment="Center"><Run Text=" Red Circle"/></TextBlock>
<TextBlock TextAlignment="Center"><Run Text=" "/></TextBlock>
<Rectangle Height="60" Width="120" Fill="Green"></Rectangle>
<TextBlock TextAlignment="Center"><Run Text=" Green Rectangle"/></TextBlock>
</StackPanel>
</Button>
```

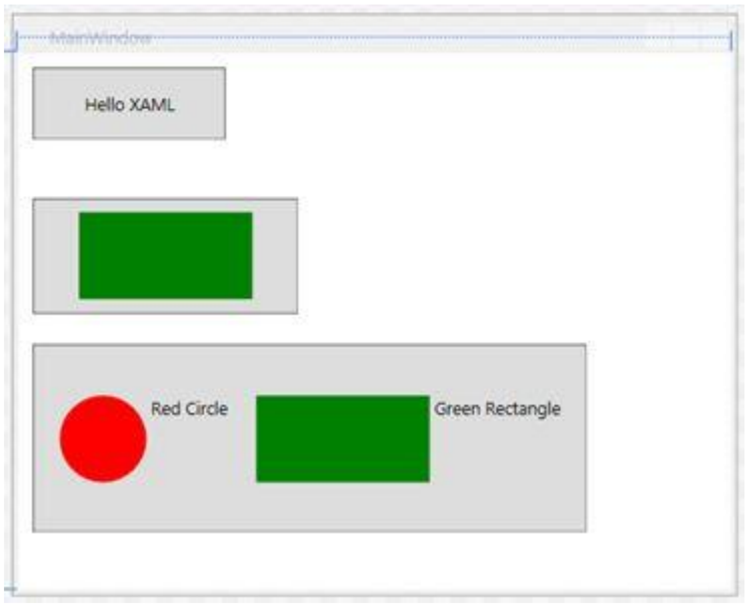
Here is the new output.



Here is the complete XAML code.

```
<Window x:Class="ContentPropertySample.MainWindow"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="MainWindow" Height="400" Width="500" >
<Grid x:Name="ParentGrid">
<Button Height="50" Margin="10,10,350,310" Content="Hello XAML" />
<Button Height="80" Margin="10,80,300,170">
<Rectangle Height="60" Width="120" Fill="Green"></Rectangle>
</Button>
<Button Margin="10,201,100,40">
<StackPanel Orientation="Horizontal">
<Ellipse Height="60" Width="60" Fill="Red"/>
<TextBlock TextAlignment="Center"><Run Text=" Red Circle"/></TextBlock>
<TextBlock TextAlignment="Center"><Run Text=" "/></TextBlock>
<Rectangle Height="60" Width="120" Fill="Green"></Rectangle>
<TextBlock TextAlignment="Center"><Run Text=" Green Rectangle"/></TextBlock>
</StackPanel>
</Button>
</Grid>
</Window>
```

There is the final output.



As you can imagine from the preceding examples, you can pretty much host any user interface as the content of a XAML element.

The following code creates the preceding Button controls dynamically in the code and sets their Content properties to a string, a Rectangle and a StackPanel respectively.

```
// Button with string content
Button helloButton = new Button();
helloButton.Margin = new Thickness(10,10,350,310);
helloButton.Content = "Hello XAML";
// Button with a UIElement
Button buttonWithRectangle = new Button();
buttonWithRectangle.Height = 80;
buttonWithRectangle.Margin = new Thickness(10, 80, 300, 170);
// Create a Rectangle
Rectangle greenRectangle = new Rectangle();
greenRectangle.Height = 60;
greenRectangle.Width = 120;
greenRectangle.Fill = Brushes.Green;
// Set Rectangle as Button.Content
buttonWithRectangle.Content = greenRectangle;
// Button with a Container, StackPanel
Button buttonWithStackPanel = new Button();
buttonWithStackPanel.Margin = new Thickness(10, 10, 350, 310);
// Create a StackPanel and set its orinetation to horizontal
StackPanel stackPanel = new StackPanel();
stackPanel.Orientation = Orientation.Horizontal;
// Create an Ellipse
Ellipse redEllipse = new Ellipse();
redEllipse.Width = 60;
redEllipse.Height = 60;
redEllipse.Fill = Brushes.Red;
// Add to StackPanel
stackPanel.Children.Add(redEllipse);
// Create a TextBlock
TextBlock textBlock1 = new TextBlock();
textBlock1.TextAlignment = TextAlignment.Left;
textBlock1.Text = "Red Circle";
// Add to StackPanel
stackPanel.Children.Add(textBlock1);
// Create a TextBlock
TextBlock space = new TextBlock();
space.TextAlignment = TextAlignment.Center;
space.Text = " ";
// Add to StackPanel
stackPanel.Children.Add(space);
// Create a Rectangle
Rectangle greenRectangle2 = new Rectangle();
greenRectangle2.Height = 60;
greenRectangle2.Width = 120;
greenRectangle2.Fill = Brushes.Green;
// Add to StackPanel
stackPanel.Children.Add(greenRectangle2);
// Create a TextBlock
TextBlock textBlock2 = new TextBlock();
textBlock2.TextAlignment = TextAlignment.Left;
textBlock2.Text = "Green Rectangle";
// Add to StackPanel
stackPanel.Children.Add(textBlock2);
// Set StackPaenl as Button.Content
buttonWithStackPanel.Content = stackPanel;
// Add dynamic button controls to the Window
ParentGrid.Children.Add(helloButton);
ParentGrid.Children.Add(buttonWithRectangle);
ParentGrid.Children.Add(buttonWithStackPanel);
```

8 a. ii.

In XAML applications, markup extensions are a method/technique to gain a value that is neither a specific XAML object nor a primitive type. Markup extensions can be defined by opening and closing curly braces and inside that curly braces, the scope of the markup extension is defined.

Data binding and static resources are markup extensions. There are some predefined XAML markup extensions in **System.xaml** which can be used.

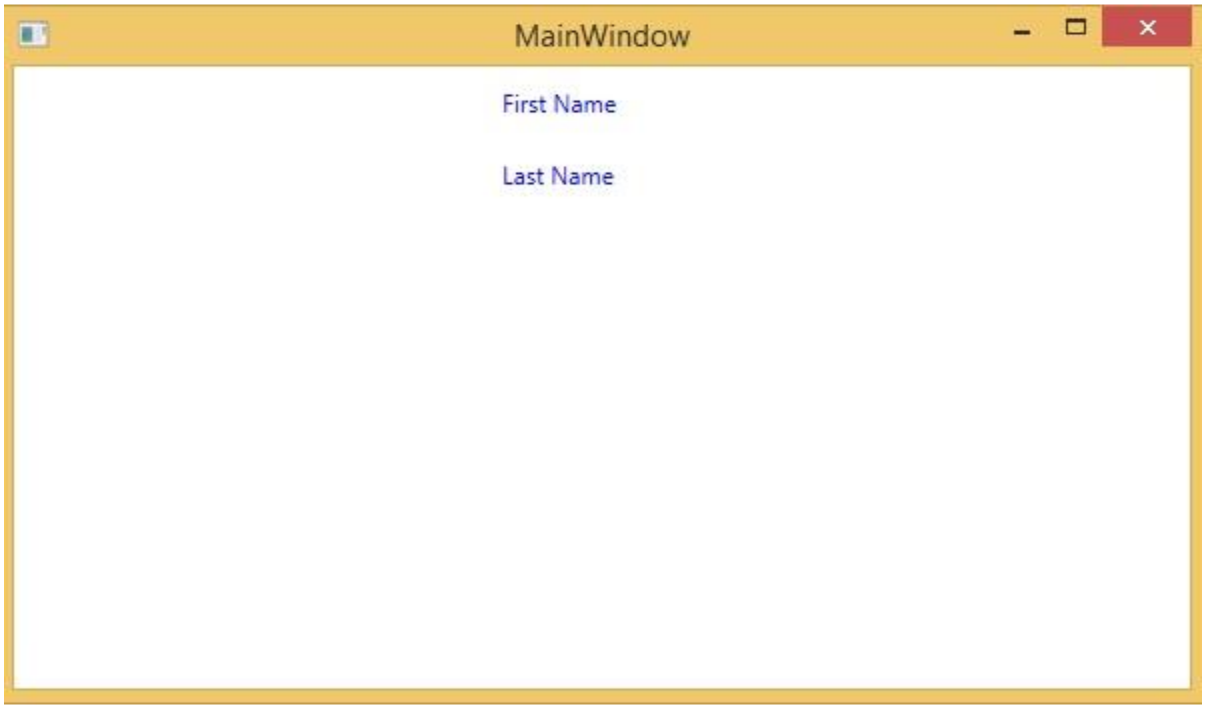
Let’s have a look at a simple example where **StaticResources** markup extension is used which is a predefined XAML markup extension.

The following XAML code creates two text blocks with some properties and their foreground is defined in **Window.Resources**.

```
<Window x:Class = "XAMLStaticResourcesMarkupExtension.MainWindow"
  xmlns = "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x = "http://schemas.microsoft.com/winfx/2006/xaml"
  Title = "MainWindow" Height = "350" Width = "525">

  <Window.Resources>
    <SolidColorBrush Color = "Blue" x:Key = "myBrush"></SolidColorBrush>
  </Window.Resources>

  <Grid>
    <StackPanel Orientation = "Vertical">
      <TextBlock Foreground = "{StaticResource myBrush}" Text = "First Name"
        Width = "100" Margin = "10" />
      <TextBlock Foreground = "{StaticResource myBrush}" Text = "Last Name"
        Width = "100" Margin = "10" />
    </StackPanel>
  </Grid>

</Window>
```

In **Window.Resources**, you can see **x:Key** is used which uniquely identifies the elements that are created and referenced in an XAML defined dictionary to identify a resource in a resource dictionary.

When you compile and execute the above code, it will produce the following MainWindow. You can see the two text blocks with blue foreground color.



In XAML, custom markup extensions can also be defined by inheriting MarkupExtension class and overriding the ProvideValue method which is an abstract method in the MarkupExtension class.

Let’s have a look at a simple example of custom markup extension.

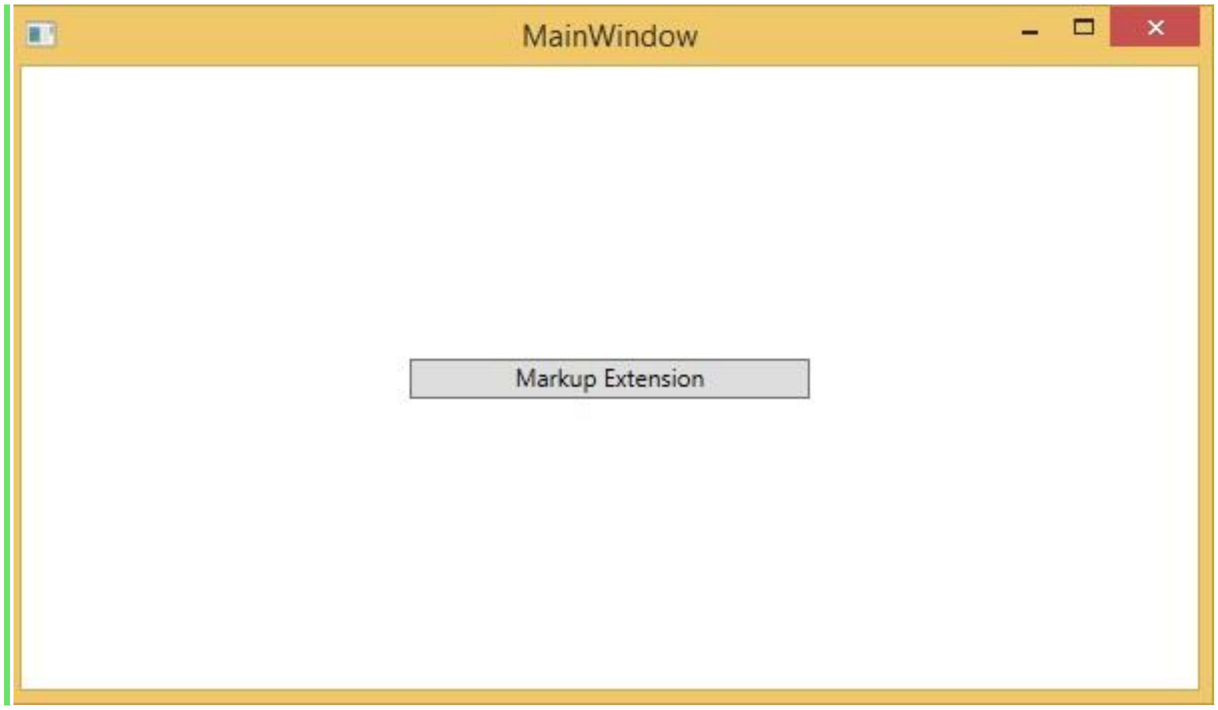
```
<Window x:Class = "XAMLMarkupExtension.MainWindow"
  xmlns = "http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x = "http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:my = "clr-namespace:XAMLMarkupExtension"
  Title = "MainWindow" Height = "350" Width = "525">

  <Grid>
    <Button Content = "{my:MyMarkupExtension FirstStr = Markup, SecondStr = Extension}"
      Width = "200" Height = "20" />
  </Grid>

</Window>
```

In the above XAML code, a button is created with some properties and for the content value, a custom markup extension **(my:MyMarkupExtension)** has been used with two values “Markup” and “Extension” which are assigned to FirstStr and SecondStr respectively.

Actually, MyMarkupExtension is a class which is derived from MarkupExtension as shown below in the C# implementation. This class contains two string variables, FirstStr and SecondStr, which are concatenated and return that string from the ProvideValue method to the Content of a button.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Markup;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace XAMLMarkupExtension {
  /// <summary>
    /// Interaction logic for MainWindow.xaml
  /// </summary>

  public partial class MainWindow : Window {
    public MainWindow() {
      InitializeComponent();
    }
  }
  public class MyMarkupExtension : MarkupExtension {
    public MyMarkupExtension() { }
    public String FirstStr { get; set; }
    public String SecondStr { get; set; }

    public override object ProvideValue(IServiceProvider serviceProvider) {
     return FirstStr + " " + SecondStr;
    }
  }
}
```

Let's run this application and you can see immediately in our MainWindow that "markup extension" has been successfully used as the content of the button.



8 b.i. Implementing an MDI Form

The Multiple-Document Interface (MDI) is a specification that defines a user interface for applications that enable the user to work with more than one document at the same time under one parent form (window).

Visualize the working style of an application in which you are allowed to open multiple forms in one parent container window, and all the open forms will get listed under the Windows menu. Whereas having an individual window for each instance of the same application is termed as single document interface (SDI); applications such as Notepad, Microsoft Paint, Calculator, and so on, are SDI applications. SDI applications get opened only in their own windows and can become difficult to manage, unlike when you have multiple documents or forms open inside one MDI interface.

Hence, MDI applications follow a parent form and child form relationship model. MDI applications allow you to open, organize, and work with multiple documents at the same time by opening them under the context of the MDI parent form; therefore, once opened, they can't be dragged out of it like an individual form.

The parent (MDI) form organizes and arranges all the child forms or documents that are currently open. You might have seen such options in many Windows applications under a Windows menu, such as Cascade, Tile Vertical, and so on.

Try It: Creating an MDI Parent Form with a Menu Bar

In this exercise, you will create an MDI form in the WinApp project. You will also see how to create a menu bar for the parent form, that will allow you to navigate to all the child forms. To do so, follow these steps:

1. Navigate to Solution Explorer, select the WinApp project, right-click, and select "Add" -> "Windows form". Change the Name value from "Form1.cs" to "ParentForm.cs", and click "Add".
2. Select the newly added ParentForm in the Design View. Select the ParentForm form by clicking the form's title bar, navigate to the Properties window, and set the following properties:
    - Set the "IsMdiContainer" property to True (the default value is False). Notice that the background color of the form has changed to dark gray.
    - Set the Size property's Width to 546 and Height to 411.
3. Drag a MenuStrip control to the ParentForm. In the top-left corner, you should now see a drop-down showing the text "Type Here". Enter the text "Open Forms" in the drop-down. This will be your main, top-level menu.
4. Now under the Open Forms menu, add a submenu by entering the text "Win App".
5. Under the Win App submenu, enter "User Info".
6. Now click the top menu, "Open Forms", and on the right side of it, type "Help". Under the Help menu, enter "Exit".
7. Now, click the top menu, on the right side of Help, type "Windows".
8. Under the Windows menu, add the following options as separate submenus: Cascade, Tile Horizontal, Tile Vertical, and Arrange Icons. These will help in arranging the child forms.
9. Now it's time to attach code to the submenus you have added under the main menu Open Forms. First, you'll add code for the submenu Win App, that basically will open the WinApp form. In the Design View, double-click the "Win App" submenu, that will take you to the Code View. Under the click event, add the following code:

    ```
    WinApp objWA = new WinApp();
    objWA.Show();
    ```

10. Now to associate functionality with the User Info submenu: double-click this submenu, and under the click event add the following code:

    ```
    UserInfo objUI = new UserInfo();
    objUI.Show();
    ```

11. To associate functionality with the Exit submenu located under the Help main menu, double-click "Exit", and under the click event add the following code:

    ```
    Application.Exit();
    ```

12. Now you have the form-opening code functionality in place, and you are nearly set to run the application. But first, you need to set the ParentForm as the start-up object. To do so, open Program.cs, and modify the "Application.Run(new UserInfo());" statement to the following:

    Application.Run(new ParentForm());

13. Now build the solution, and run the application by pressing F5; the MDI application will open and should look as in Figure 1-1.

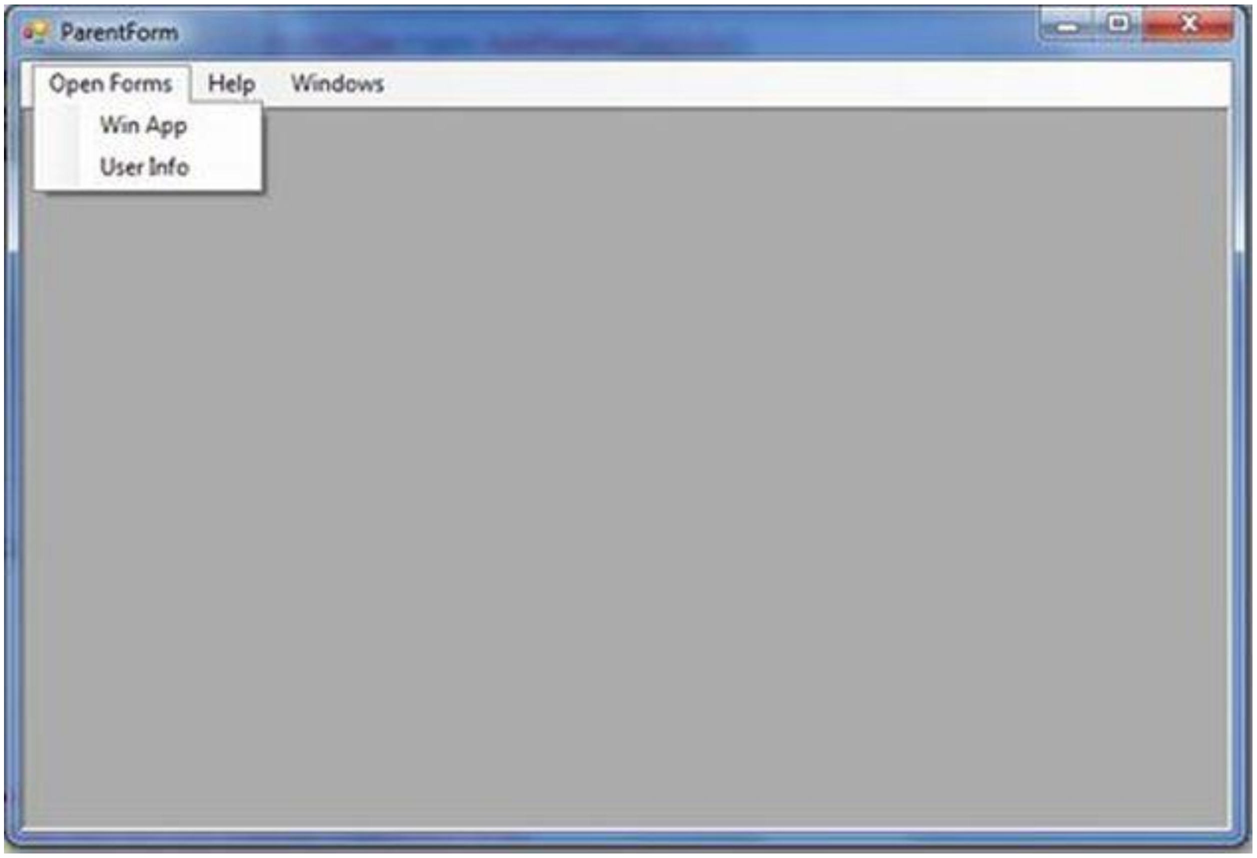


Figure 1-1. Running an MDI form application

14. Now if you click "Win App" and then "User Info" then both the forms will open one by one. These forms can be opened and dragged outside of the MDI form. This is not an expected behavior from a MDI application, as shown in Figure 1-2.

    This issue will be addressed later in this chapter.

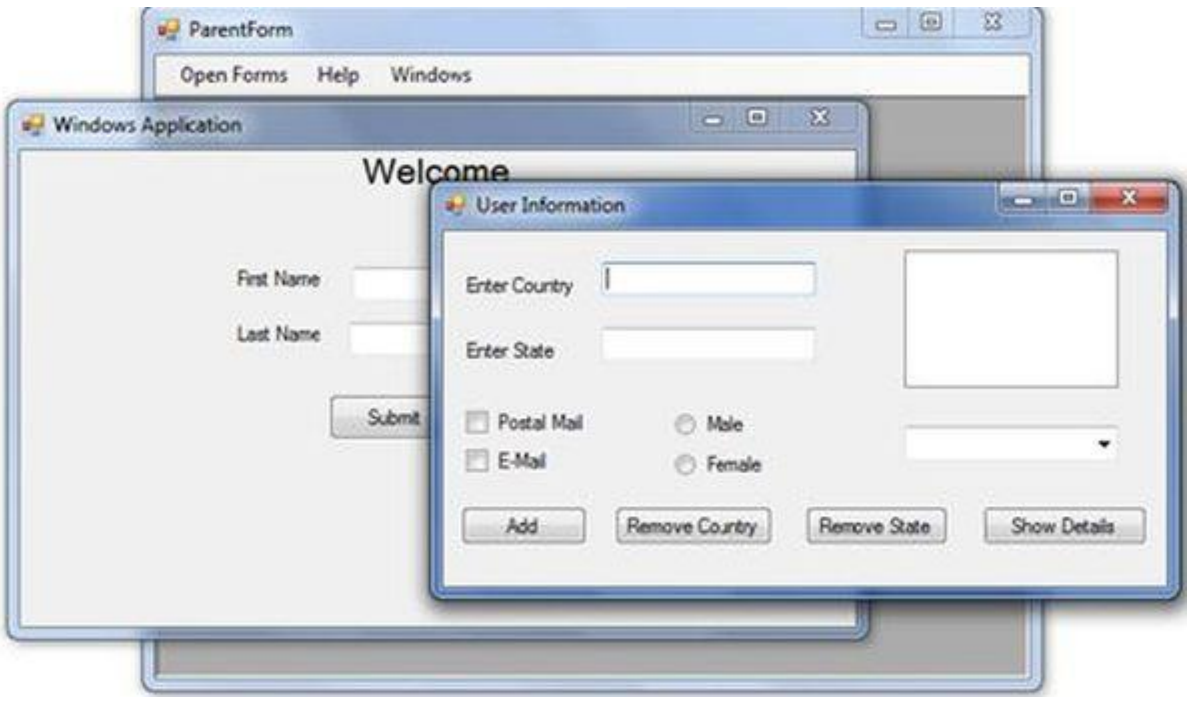


Figure 1-2. Running an MDI form application

How It Works

Each Windows Forms form is a class and exposes a Show() function by an instance created for it. You use the following code, that is creating an object and then invoking the Show() method. This opens the other form from the MDI parent form.

This creates an instance of the WinApp form and opens it for you:

```
WinApp objWA = new WinApp();
objWA.Show();
```

The following code creates an instance of the UserInfo form and opens it for you:

```
UserInfo objUI = new UserInfo();
objUI.Show();
```

You close the application with the following code:

```
Application.Exit();
```

Try It: Opening an MDI Child Form Within an MDI Application

As mentioned and shown in Figure 1-2 above, the problem is that even though the MDI form shows a parent menu, the forms are still able to open outside, and context is moved from a form to another. You can try clicking the title bar of each the open form, and you will see how you can move back and forth with these opened forms.

In this exercise, you will overcome this problem and associate all the forms you created earlier as MDI child forms to the main MDI parent form you created in the previous task.

1. In the project you modified in the previous exercise, you'll first make the WinApp form an MDI child form. To do so, you need to set the MdiParent property of the child form's object to the MDI parent form itself, but in the Code View. You have already added functionality in the previous task (opening the WinApp form); just before the line where you are calling the Show() method, add the following code (this code can be found under Win App menu click event):

   ```
   objWA.MdiParent=this;
   ```

   After adding this line, the code will appear as follows:

   ```
   WinApp objWA = new WinApp();
   objWA.MdiParent = this;
   objWA.Show();
   ```

   Note: this is a C# language keyword that represents the current instance of the class. In this case, it refers to ParentForm. Because you are writing this code inside ParentForm, you can use the this keyword for the same.

   Now you will make the UserInfo form an MDI child form. To do so, you need to set the MdiParent property to the name of the MDI parent form but in the Code View. Add the following code as you did in the previous step (this code can be found under the User Info menu click event):

   ```
   objUI.MdiParent=this;
   ```

   After adding this line, the code will appear as follows:

   ```
   UserInfo objUI = new UserInfo();
   objUI.MdiParent=this;
   objUI.Show();
   ```

   Now build the solution, and run the application by pressing F5; the MDI application will open and should appear as shown in Figure 1-3.

   Click "Open Form" -> "Win App"; the WinApp form should open. Again, open the main menu and click "User Info". Both the forms should now be open inside your main MDI parent form application, and unlike before, you will not be able to drag these out of your MDI parent form (as shown in Figure 1-2). Figure 1-3 shows the expected behavior of an MDI application with opened form(s).

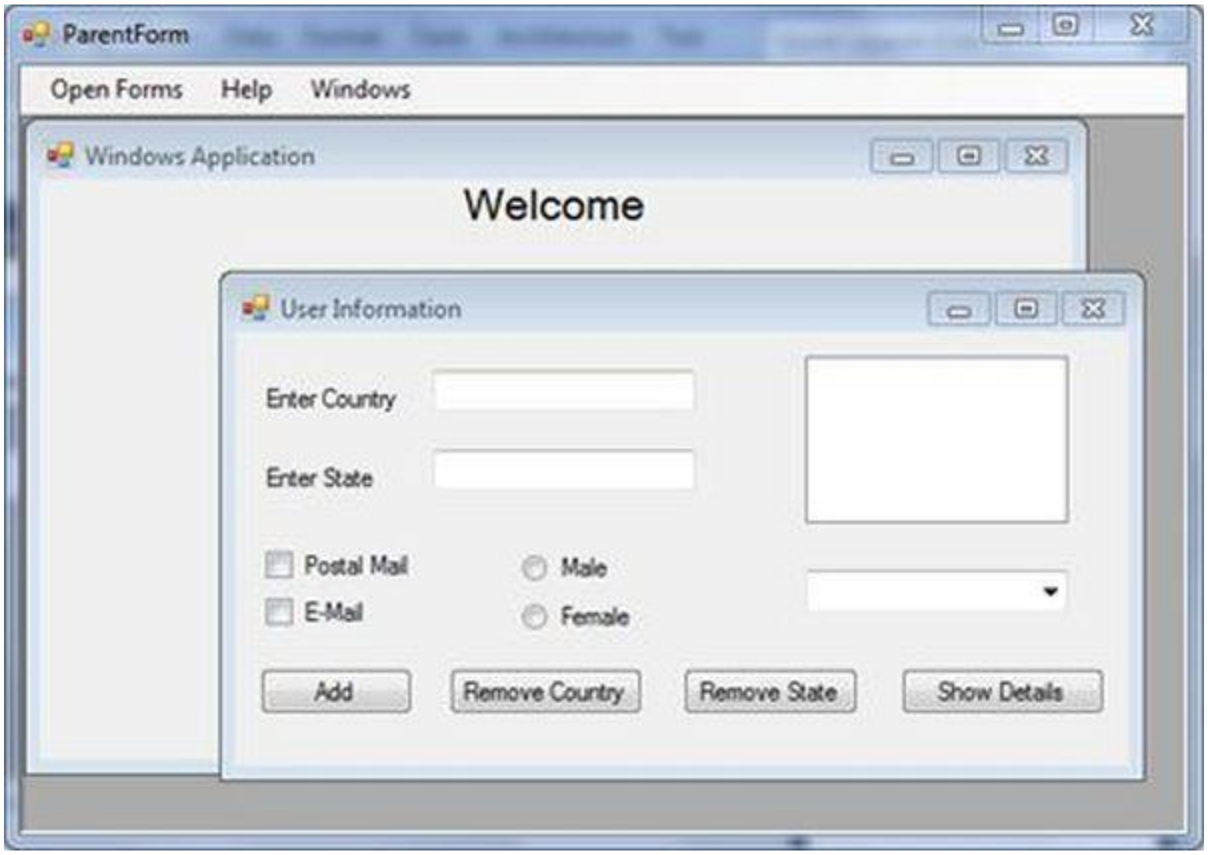


Figure 1-3. Opening child forms inside an MDI form application:

Because both the forms are open inside one MDI parent, it becomes easier to work with them, and they are not draggable outside of the MDI parent. Switch among these forms by clicking their title bars. Once you are done with the forms, close the application by selecting "Help" -> "Exit".

How It Works

As you noticed in an earlier exercise, the only issue discussed was that child forms opened and were able to be dragged outside. In other words, they didn't really belong to a parent form. An MDI application is about claiming a form with a menu bar as an MDI parent so all the child forms can open inside it.

To do so, first an object of the child form needs to be created:

```
WinApp objWA = new WinApp();
```

But before we really invoke the Show() method on the objWA, you need to tell the object who is its parent so it operates from within the MDI parent form. To accomplish this, you will use the "this" keyword, that represents the current form class.

```
objWA.MdiParent = this;
```

You have an object created and its context set to a MDI parent form, so now it's a perfect time to call the Show() method, that will launch the form so you can work with it.

```
objWA.Show();
```

The "wa.MdiParent=this;" line tells the child form which form is its parent. Because you want all the child forms to appear inside ParentForm and you write the code inside the MDI parent form, you can use the "this" keyword to represent the current object.

Set the previous suggested changes for UserInfo as well.

```
UserInfo objUI = new UserInfo();
objUI.MdiParent=this;
objUI.Show();
```

Try It: Arranging MDI Child Forms Within an MDI Application

Multiple forms will open within one MDI window, so once you have a few open, your MDI application will be cluttered. It's hard to move forms around to shift your focus from one to another. Hence, it is a prime concern to have a mechanism that allows you to arrange the forms in an organized manner.

For example, in most applications you can arrange the forms and then cascade them so you can see the stack of open forms. Or you can tile them vertically or horizontally so you can see multiple forms side by side. You can even minimize all the open forms and arrange them as an icon.

To accomplish this, in this exercise you will add the Windows menu as shown in Figure 1-4.

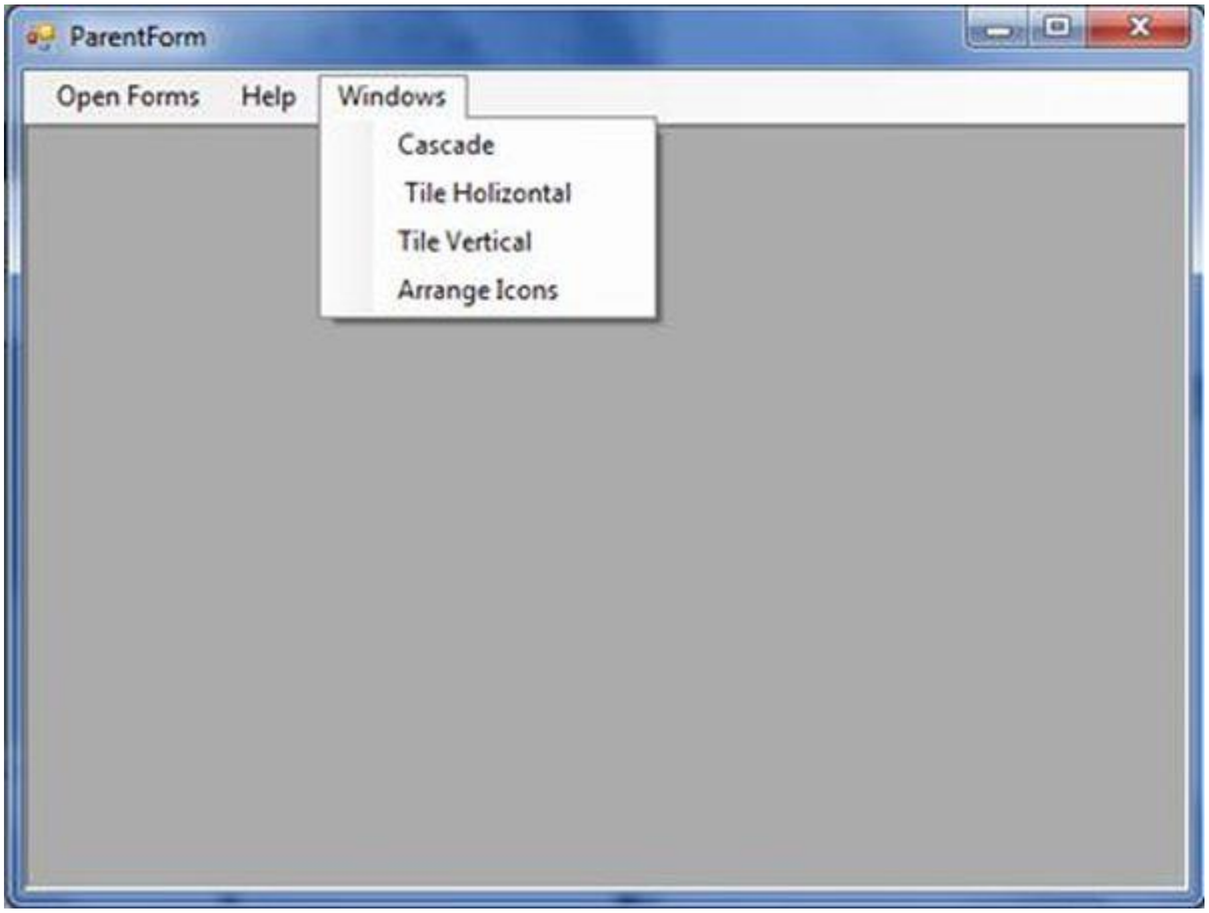


Figure 1-4. Windows menu of MDI form application to arrange child forms

.NET's Windows Forms offers the LayoutMdi method, that takes the MdiLayout enumeration to rearrange the child forms in an MDI parent form. There are four modes you can arrange your forms into: Cascade, Tile Horizontal, Tile Vertical, and ArrangeIcons.

1. Open the ParentForm in the Design View, and click the Windows menu, as shown in Figure 1-4.
2. Double-click the first option, Cascade, under Windows, and it will take you to its click event. Add the following code:

   ```
   LayoutMdi(MdiLayout.Cascade);
   ```

3. Double-click Tile Horizontal, and in the Code View under the click event, add the following code:

   ```
   LayoutMdi(MdiLayout.TileHorizontal);
   ```

4. Double-click the Tile Vertical, and in the Code View under the click event, add the following code:

```
LayoutMdi(MdiLayout.TileVertical);
```

5. Double-click Tile Vertical, and in the Code View under the click event, add the following code:

```
LayoutMdi(MdiLayout.ArrangeIcons);
```

6. Now build the solution, and run the application by pressing F5; the MDI application will open. After it opens, go to the Open Forms menu and click Win App and User Info one by one. It is important to have at least two forms open in the MDI parent form. Now go to the Windows menu and try the available options by clicking Cascade, then Arrange Vertical, then Arrange Horizontal, and finally Arrange Icons. When you will try these options, Tile Vertical will show the child forms arranged as in Figure 1-5.

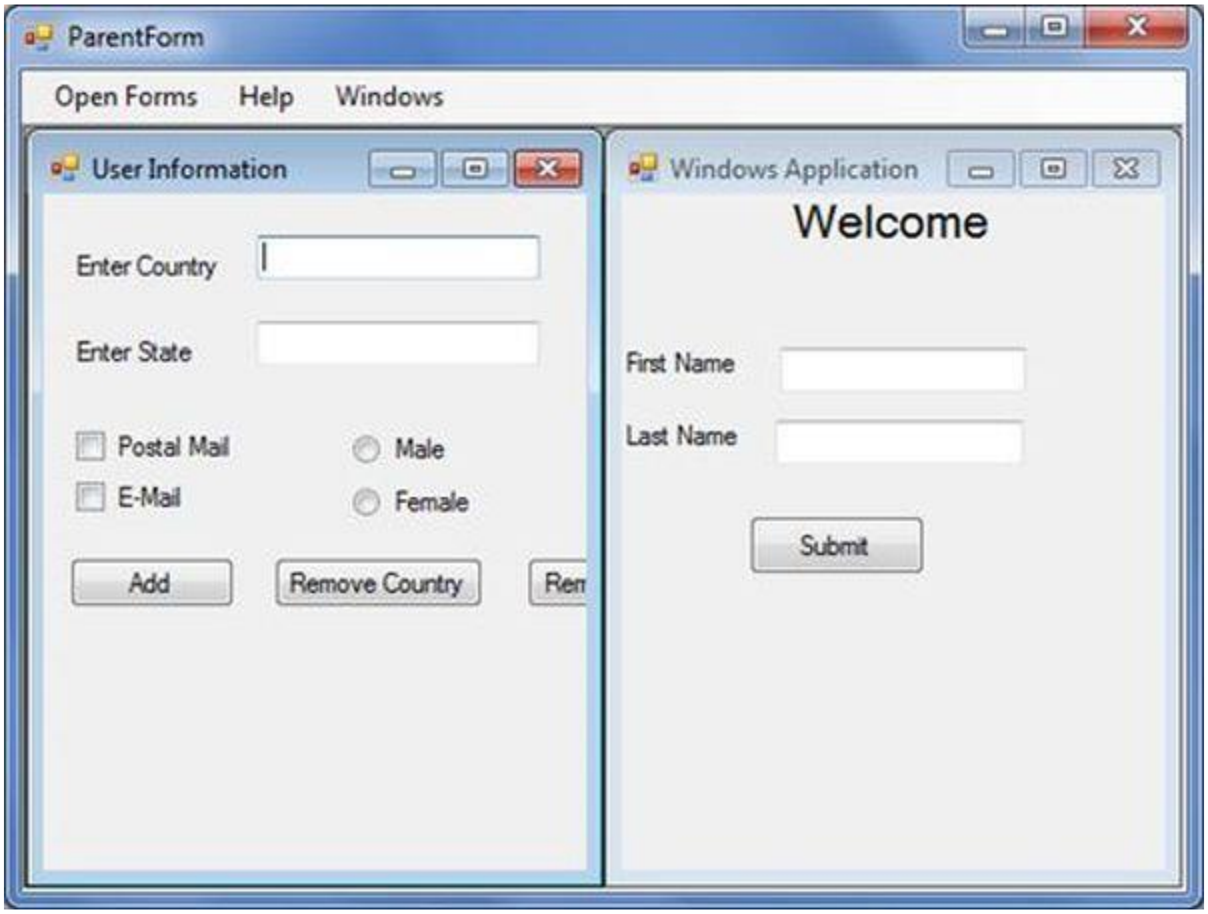


Figure 1-5. Arranging (Tile Vertical) child forms in the MDI form application

8 b.ii Event means something we do like posting this article, reading this article etc. Everything we do, is an event. In order to illustrate event driven programming, I would like to tell a story and show how I can explain this real life scenario using events. So start thinking OOPs.

First define one class Parent with a name and a method.

```
class Parent
    {
        public String Name { get; set; }
        public void NotifyMe(int pGPAScore)
        {
            Console.WriteLine(String.Format("{0} notified about the GPA {1}", Name,
pGPAScore.ToString()));
        }
    }
```

Declare one delegate

```
public delegate void NotifyGpaDelegate(int GPAScore);
```

Why do we use this delegate? Because we need to call Parent's NotifyMe method. So the delegate signature would be identical with the NotifyMe method's signature.

Declare Student class with name and GPA score

```
class Student
    {
        public event NotifyGpaDelegate NotifyToParent;
        public String Name { get; set; }
        public int GPAScore { get; set; }
        public void RecordGPAScore()
        {
            if (NotifyToParent != null)
                NotifyToParent(GPAScore);
        }
    }
```

Here we need to understand that Student calls the method NotifyMe of Parent class. So we need to declare a delegate which will point to the NotifyMe method.

Suppose in the future, Student class needs to notify another class like Teacher class, then you don't need to instantiate Teacher class within Student. You just need to write the eventhandler for it.

Think about the Button class and TextBox class. Button class has OnClick event. Now if we write any specific code within Button class, would it be reusable? So better define delegate, then an event of that delegate type and tell to developers that the method you want to call, should match this delegate's signature. The same thing we are doing here. We defined one delegate NotifyGPADelegate, then an event NotifyToParent of that delegate type and ask developers to subscribe to this event.

Here how we wire up these two objects

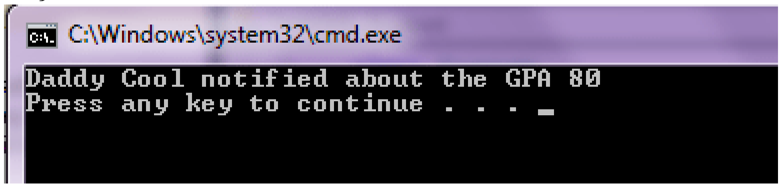
```
Student oStudent = new Student();
        oStudent.Name = "James";
        oStudent.GPAScore = 80;
        Parent oParent = new Parent();
        oParent.Name = "Daddy Cool";
        oStudent.NotifyToParent += new NotifyGpaDelegate(oParent.NotifyMe);
        oStudent.RecordGPAScore();
```

We are instantiating one Student class and Parent class. Then we subscribe the NotifyToParent event with the method NotifyMe of Parent class. At last we call the method to RecordGPAScore, which in turn calls the NotifyMe of Parent class.

Sample code looks like

```
class Program
   {
      static void Main(string[] args)
       {
          Student oStudent = new Student();
          oStudent.Name = "James";
          oStudent.GPAScore = 80;
          Parent oParent = new Parent();
          oParent.Name = "Daddy Cool";
          oStudent.NotifyToParent += new NotifyGpaDelegate(oParent.NotifyMe);
          oStudent.RecordGPAScore();
       }
   }
   public delegate void NotifyGpaDelegate(int GPAScore);
   class Student
   {
       public event NotifyGpaDelegate NotifyToParent;
       public String Name { get; set; }
       public int GPAScore { get; set; }
       public void RecordGPAScore()
       {
          if (NotifyToParent != null)
              NotifyToParent(GPAScore);
       }
   }
```

```
    class Parent
    {
        public String Name { get; set; }
        public void NotifyMe(int pGPAScore)
        {
            Console.WriteLine(String.Format("{0} notified about the GPA {1}", Name,
pGPAScore.ToString()));
        }
    }
```



```
C:\Windows\system32\cmd.exe
Daddy Cool notified about the GPA 80
Press any key to continue . . . _
```

9 a. Validation controls are used to,

- Implement presentation logic.
- To validate user input data.
- Data format, data type and data range is used for validation.

Validation is of two types

1. Client Side
2. Serve Side

Client side validation is good but we have to be dependent on browser and scripting language support.

Client side validation is considered convenient for users as they get instant feedback. The main advantage is that it prevents a page from being postback to the server until the client validation is executed successfully.

For developer point of view serve side is preferable because it will not fail, it is not dependent on browser and scripting language.

You can use ASP.NET validation, which will ensure client, and server validation. It work on both end; first it will work on client validation and than on server validation. At any cost server validation will work always whether client validation is executed or not. So you have a safety of validation check.

For client script .NET used JavaScript. WebUIValidation.js file is used for client validation by .NET

Validation Controls in ASP.NET

An important aspect of creating ASP.NET Web pages for user input is to be able to check that the information users enter is valid. ASP.NET provides a set of validation controls that provide an easy-to-use but powerful way to check for errors and, if necessary, display messages to the user.

There are six types of validation controls in ASP.NET

1. RequiredFieldValidation Control
2. CompareValidator Control
3. RangeValidator Control
4. RegularExpressionValidator Control
5. CustomValidator Control
6. ValidationSummary

The below table describes the controls and their work,

Validation Control	Description
RequiredFieldValidation	Makes an input control a required field
CompareValidator	Compares the value of one input control to the value of another input control or to a fixed value
RangeValidator	Checks that the user enters a value that falls between two values
RegularExpressionValidator	Ensures that the value of an input control matches a specified pattern
CustomValidator	Allows you to write a method to handle the validation of the value entered
ValidationSummary	Displays a report of all validation errors occurred in a Web page

All validation controls are rendered in form as <span> (label are referred as <span> on client by server)

Important points for validation controls

- ControlToValidate property is mandatory to all validate controls.
- One validation control will validate only one input control but multiple validate control can be assigned to a input control.

Validation Properties

Usually, Validation is invoked in response to user actions like clicking submit button or entering data. Suppose, you wish to perform validation on page when user clicks submit button.

Server validation will only performed when CauseValidation is set to true.

When the value of the CausesValidation property is set to true, you can also use the ValidationGroup property to specify the name of the validation group for which the Button control causes validation.

Page has a Validate() method. If it is true this methods is executed. Validate() executes each validation control.

To make this happen, simply set the CauseValidation property to true for submit button as shown below:

```
<asp:Button ID="Button2" runat="server" Text="Submit" CausesValidation=true />
```

Lets understand validation controls one by one with practical demonstration:

RequiredFieldValidation Control

The RequiredFieldValidator control is simple validation control, which checks to see if the data is entered for the input control. You can have a RequiredFieldValidator control for each form element on which you wish to enforce Mandatory Field rule.

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator3" runat="server"
Style="top: 98px; left: 367px; position: absolute; height: 26px; width: 162px"
ErrorMessage="password required" ControlToValidate="TextBox2">
</asp:RequiredFieldValidator>
```

CompareValidator Control

The CompareValidator control allows you to make comparison to compare data entered in an input control with a constant value or a value in a different control.

It can most commonly be used when you need to confirm password entered by the user at the registration time. The data is always case sensitive.

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator2" runat="server" Style=" top: 145px;
      left: 367px; position: absolute; height: 26px; width: 162px" ErrorMessage="password required"
      ControlToValidate="TextBox3"></asp:RequiredFieldValidator>
```

RangeValidator Control

The RangeValidator Server Control is another validator control, which checks to see if a control value is within a valid range. The attributes that are necessary to this control are: MaximumValue, MinimumValue, and Type.

```
<asp:RangeValidator ID="RangeValidator1" runat="server"
Style="top: 194px; left: 365px; position: absolute; height: 22px; width: 105px"
ErrorMessage="RangeValidator" ControlToValidate="TextBox4" MaximumValue="100" MinimumValue="18" Type="Integer"></asp:RangeValidator>
```

RegularExpressionValidator Control

A regular expression is a powerful pattern matching language that can be used to identify simple and complex characters sequence that would otherwise require writing code to perform.

Using RegularExpressionValidator server control, you can check a user's input based on a pattern that you define using a regular expression.

It is used to validate complex expressions. These expressions can be phone number, email address, zip code and many more. Using Regular Expression Validator is very simple. Simply set the ValidationExpression property to any type of expression you want and it will validate it.

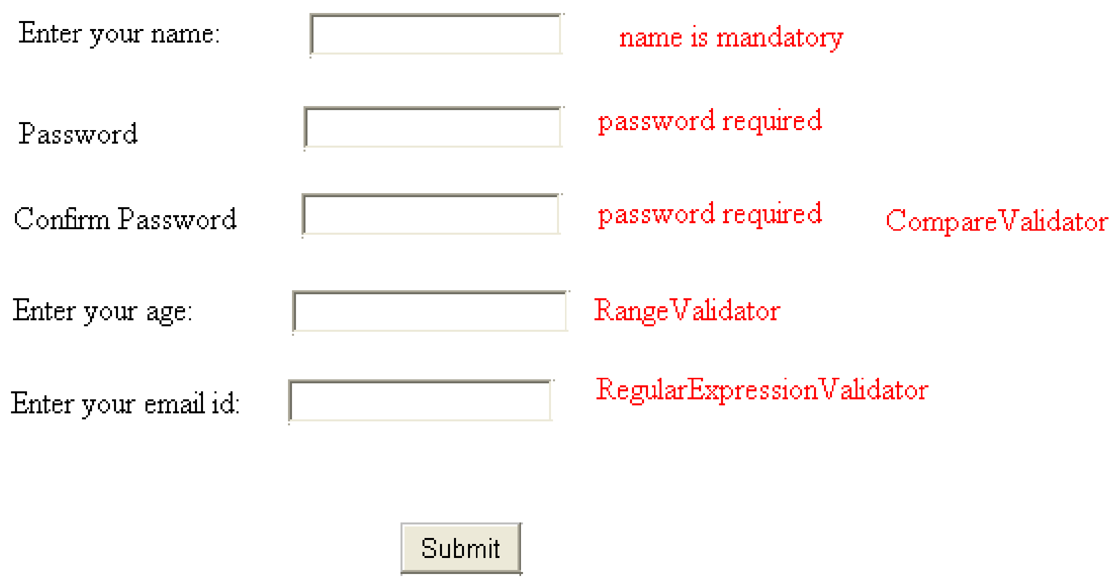
If you don't find your desired regular expression, you can create your custom one.

In the example I have checked the email id format:

```
<asp:RegularExpressionValidator ID="RegularExpressionValidator1" runat="server" Style="top: 234px;
        left: 366px; position: absolute; height: 22px; width: 177px"
        ErrorMessage="RegularExpressionValidator" ControlToValidate="TextBox5"

        ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*"></asp:RegularExpressionValidator>
```

The complete code for the above 4 controls is as,

Default.aspx Design



Default.aspx Source code

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title>Untitled Page</title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
      <asp:Label ID="Label3" runat="server" Style="top: 241px; left: 70px; position: absolute;
```

```
            height: 22px; width: 128px; bottom: 282px;" Text="Enter your email id:"><
/asp:Label>
        <asp:Label ID="Label1" runat="server" Style="top: 54px; left: 74px; position:
absolute;
            height: 22px; width: 128px" Text="Enter your name:"></asp:Label>
        <asp:TextBox ID="TextBox1" runat="server" Style="top: 54px; left: 221px; p
osition: absolute;
            height: 22px; width: 128px; right: 396px;"></asp:TextBox>
        <asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server" St
yle="top: 56px;
            left: 378px; position: absolute; height: 22px; width: 128px" ErrorMessage="
RequiredFieldValidator"
            ControlToValidate="TextBox1">name is
        mandatory </asp:RequiredFieldValidator>
    </div>
    <p>
        <asp:Button ID="Button1" runat="server" Style="top: 311px; left: 267px; posi
tion: absolute;
            height: 26px; width: 61px" Text="Submit" />
    </p>
    <asp:TextBox ID="TextBox3" runat="server" Style="top: 145px; left: 217px; po
sition: absolute;
        height: 22px; width: 131px" TextMode="Password"></asp:TextBox>
    <p>
        <asp:TextBox ID="TextBox2" runat="server" Style="top: 101px; left: 218px;
position: absolute;
            height: 22px; width: 131px" TextMode="Password"></asp:TextBox>
        <asp:Label ID="Label4" runat="server" Style="top: 105px; left: 74px; position
: absolute;
            height: 22px; width: 128px" Text="Password"></asp:Label>
        <asp:TextBox ID="TextBox5" runat="server" Style="top: 239px; left: 210px;
position: absolute;
            height: 22px; width: 134px"></asp:TextBox>
    </p>
    <asp:RequiredFieldValidator ID="RequiredFieldValidator3" runat="server" Style
="top: 98px;
        left: 367px; position: absolute; height: 26px; width: 162px" ErrorMessage="pa
ssword required"
        ControlToValidate="TextBox2"></asp:RequiredFieldValidator>
    <asp:RequiredFieldValidator ID="RequiredFieldValidator2" runat="server" Style
="top: 145px;
        left: 367px; position: absolute; height: 26px; width: 162px" ErrorMessage="pa
ssword required"
        ControlToValidate="TextBox3"></asp:RequiredFieldValidator>
    <asp:CompareValidator ID="CompareValidator1" runat="server" Style="top: 14
9px; left: 512px;
```

```
        position: absolute; height: 26px; width: 162px" ErrorMessage="CompareValidator"
        ControlToValidate="TextBox3" ValueToCompare="hello"></asp:CompareValidator>
    <p>
        <asp:Label ID="Label5" runat="server" Style="top: 148px; left: 71px; position: absolute;
            height: 22px; width: 128px; bottom: 375px;" Text="Confirm Password"></asp:Label>
        <asp:TextBox ID="TextBox4" runat="server" Style="top: 194px; left: 212px; position: absolute;
            height: 22px; width: 140px"></asp:TextBox>
        <asp:Label ID="Label6" runat="server" Style="top: 194px; left: 71px; position: absolute;
            height: 22px; width: 128px; bottom: 329px;" Text="Enter your age:"></asp:Label>
    </p>
    <asp:RangeValidator ID="RangeValidator1" runat="server" Style="top: 194px; left: 365px;
        position: absolute; height: 22px; width: 105px" ErrorMessage="RangeValidator"
        ControlToValidate="TextBox4" MaximumValue="100" MinimumValue="18" Type="Integer"></asp:RangeValidator>
    <asp:RegularExpressionValidator ID="RegularExpressionValidator1" runat="server" Style="top: 234px;
        left: 366px; position: absolute; height: 22px; width: 177px"
        ErrorMessage="RegularExpressionValidator" ControlToValidate="TextBox5"
        ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*"></asp:RegularExpressionValidator>
    </form>
</body>
</html>
```

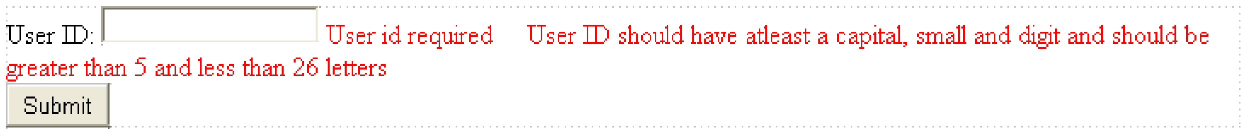
CustomValidator Control

You can solve your purpose with ASP.NET validation control. But if you still don't find solution you can create your own custom validator control.

The CustomValidator Control can be used on client side and server side. JavaScript is used to do client validation and you can use any .NET language to do server side validation.

I will explain you CustomValidator using server side. You should rely more on server side validation.

To write CustomValidator on server side you override ServerValidate event.

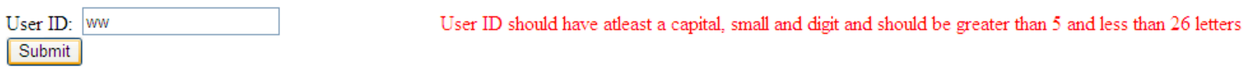

User ID: User id required User ID should have atleast a capital, small and digit and should be greater than 5 and less than 26 letters

Source Code

```
<%@ Page Language="C#" AutoEventWireup="true"  CodeFile="Default.aspx.cs"
Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-
//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Label ID="Label1" runat="server" Text="User ID:"></asp:Label>
 <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
 <asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server"
            ControlToValidate="TextBox1" ErrorMessage="User id required"></asp:Re
quiredFieldValidator>

        <asp:CustomValidator ID="CustomValidator1" runat="server" OnServerValid
ate="UserCustomValidate"
            ControlToValidate="TextBox1"
            ErrorMessage="User ID should have atleast a capital, small and digit and sh
ould be greater than 5 and less
than 26 letters"
            SetFocusOnError="True"></asp:CustomValidator>
        </div>
    <asp:Button ID="Button1" runat="server" onclick="Button1_Click" Text="Subm
it" />
    </form>
</body>
</html>
```

Code behind file

```
using System;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;

public partial class _Default : System.Web.UI.Page
{
    protected void UserCustomValidate(object source, ServerValidateEventArgs args)
    {
        string str = args.Value;
        args.IsValid = false;
       //checking for input length greater than 6 and less than 25 characters
        if (str.Length < 6 || str.Length > 25)
        {
            return;
        }
        //checking for a atleast a single capital letter
        bool capital = false;
        foreach (char ch in str)
        {
            if (ch >= 'A' && ch <= 'Z')
            {
                capital = true;
                break;
            }
        }
        if (!capital)
        {
            return;
        }
       //checking for a atleast a single lower letter
        bool lower = false;
         foreach (char ch in str)
        {
            if (ch >= 'a' && ch <= 'z')
            {
                lower = true;
                break;
            }
        }
        if (!lower)
        {
            return;
        }
        bool digit = false;
        foreach (char ch in str)
        {
            if (ch >= '0' && ch <= '9')
            {
                digit = true;
                break;
            }
        }
        if (!digit)
        {
            return;
        }
        args.IsValid = true;
    }
    protected void Page_Load(object sender, EventArgs e)
    {
    }
    protected void Button1_Click(object sender, EventArgs e)
    {
    }
}
```



User ID should have atleast a capital, small and digit and should be greater than 5 and less than 26 letters

The server side validation you write does not need to provide the exact same validation as that of the client side validation. The client side validation can check for the user input data for range and type and server side validation can check for matching of data with database. Both server side and client side validation can be used for total solution.

ValidationSummary

ASP.NET has provided an additional control that complements the validator controls.

The ValidationSummary control is reporting control, which is used by the other validation controls on a page.

You can use this validation control to consolidate errors reporting for all the validation errors that occur on a page instead of leaving this up to each and every individual validation control.

The validation summary control will collect all the error messages of all the non-valid controls and put them in a tidy list.

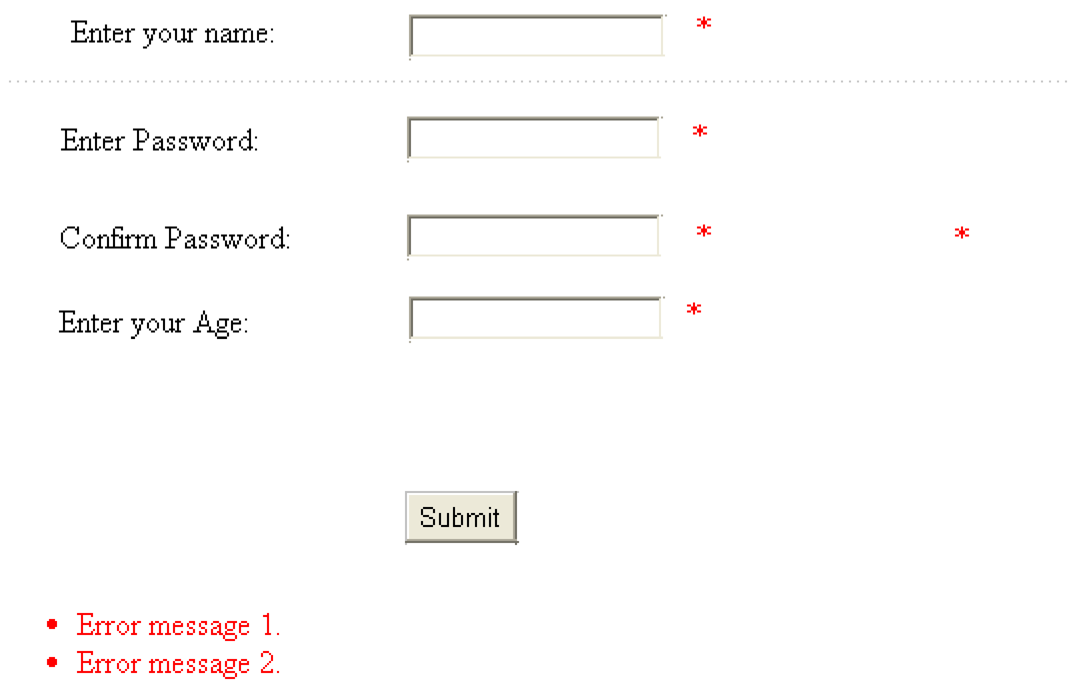
```
<asp:ValidationSummary ID="ValidationSummary1" runat="server"
        style="top: 390px; left: 44px; position: absolute; height: 38px; width: 625px" />
```

Both ErrorMessage and Text properties are used to display error messages. Text error message have precedence.

If you are using ValidationSummary than only ErrorMessage and Text property is used.

The complete code for the above ValidationSummary is as:

Default.aspx Design



- Error message 1.
- Error message 2.

Default.aspx Source code

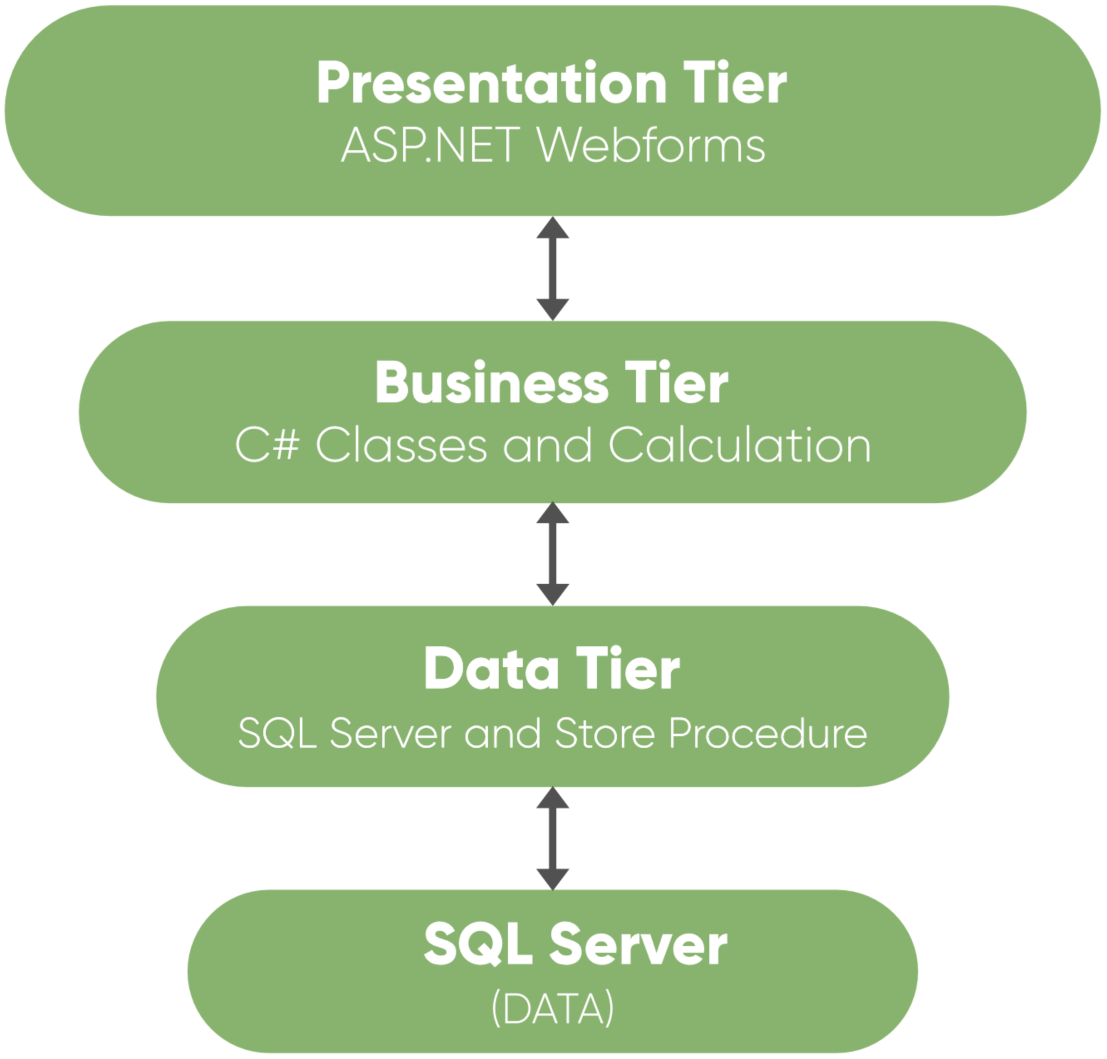
```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="_Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Label ID="Label1" runat="server" Style="top: 239px; left: 75px; position: absolute;
            height: 22px; width: 128px" Text="Enter your Age:"></asp:Label>

        <asp:Label ID="Label2" runat="server" Style="top: 94px; left: 81px; position: absolute;
            height: 22px; width: 128px" Text="Enter your name:"></asp:Label>
    </div>
    <asp:TextBox ID="TextBox1" runat="server" Style="top: 95px; left: 250px; position: absolute;
        height: 22px; width: 128px"></asp:TextBox>
    <p>
        <asp:TextBox ID="TextBox4" runat="server" Style="top: 195px; left: 249px; position: absolute;
            height: 22px; width: 127px"></asp:TextBox>
    </p>
    <p>
        <asp:Label ID="Label3" runat="server" Style="top: 148px; left: 76px; position: absolute;
            height: 22px; width: 128px" Text="Enter Password:"></asp:Label>
    </p>
    <p>
        <asp:TextBox ID="TextBox3" runat="server" Style="top: 146px; left: 249px; position: absolute;
            height: 22px; width: 127px" TextMode="Password"></asp:TextBox>
    </p>
   <p>
        <asp:Label ID="Label4" runat="server" Style="top: 197px; left: 75px; position: absolute;
            height: 22px; width: 128px" Text="Confirm Password:"></asp:Label>
    </p>
    <asp:TextBox ID="TextBox2" runat="server" Style="top: 236px; left: 250px; position: absolute;
        height: 22px; width: 127px" TextMode="Password"></asp:TextBox>
    <asp:CompareValidator ID="CompareValidator1" runat="server" Style="top: 197px; left: 522px;
        position: absolute; height: 22px; width: 17px" ErrorMessage="CompareValidator"
        ControlToCompare="TextBox2" ControlToValidate="TextBox3">*</asp:CompareValidator>
    <p>
        <asp:Button ID="Button1" runat="server" Style="top: 333px; left: 248px; position: absolute;
            height: 26px; width: 56px" Text="Submit" />
        <asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server" Style="top: 196px;
            left: 393px; position: absolute; height: 22px; width: 22px" ErrorMessage="Confirm Password mandatory & should match password"
            ControlToValidate="TextBox3">*</asp:RequiredFieldValidator>
        <asp:RangeValidator ID="RangeValidator1" runat="server" Style="top: 235px; left: 388px;
            position: absolute; height: 22px; width: 156px; bottom: 288px;" ErrorMessage="age between 18-100"
            ControlToValidate="TextBox4" MaximumValue="100" MinimumValue="18" Type="Integer">*</asp:RangeValidator>
        <asp:RequiredFieldValidator ID="RequiredFieldValidator2" runat="server" Style="top: 92px;
            left: 393px; position: absolute; height: 22px; width: 156px" ErrorMessage="Name is required"
            ControlToValidate="TextBox1">*</asp:RequiredFieldValidator>
        <asp:RequiredFieldValidator ID="RequiredFieldValidator3" runat="server" Style="top: 146px;
            left: 391px; position: absolute; height: 22px; width: 156px" ErrorMessage="Password mandatory"
            ControlToValidate="TextBox2">*</asp:RequiredFieldValidator>
    </p>
    <asp:ValidationSummary ID="ValidationSummary1" runat="server" Style="top: 390px;
        left: 44px; position: absolute; height: 38px; width: 625px" />
    </form>
</body>
</html>
```

9 b . Three-layer architecture is dividing the project into three layers that are *User interface layer*, *business layer* and data(database) layer where we separate UI, logic, and data in three divisions. Suppose the user wants to change the UI from windows to the phone than he has to only make change in UI layer, other layers are not affected by this change. Similarly, if the user wants to change the database then he has to only make a change in the data layer, rest everything remains the same.

Why to use 3-Tier Architecture in Projects?

We use 3 Tier architecture to manage large projects. We can take an example: Suppose you have created 2000 entity code and you have coded all your code in only one layer. Now suppose you want to change from windows application to mobile or web application then again you have to write code for all 2000 entities or you want to change database provider than again you have to change database connection in all 2000 entities. Hence writing such a large amount of code is a time and money wasting. Also writing a large amount of code in one layer only makes new people difficult to understand that code.

So, to provide maintenance, flexibility, updation flexibility without affecting other parts, clean and understandable code we should use 3-Tier Architecture in Projects. This architecture basically contains **three layers**:



1. Presentation Layer

This is the top layer of architecture. The topmost level of application is the user interface. It is related to the user interface that is what the user sees. The main function of this layer is to translate tasks and results in something which the user can understand. It contains pages like web forms, windows form where data is presented to the user and use to take input from the user. The presentation layer is the most important layer because it is the one that the user sees and good UI attracts the user and this layer should be designed properly.

2. Business Layer

This is the middle layer of architecture. This layer involves C# classes and logical calculations and operations are performed under this layer. It processes the command, makes logical decisions and perform calculations. It also acts as a middleware between two surrounded layers that is presentation and data layer. It processes data between these two layers. This layer implements business logic and calculations. This layer also validates the input conditions before calling a method from the data layer. This ensures the data input is correct before proceeding, and can often ensure that the outputs are correct as well. This validation of input is called business rules.

3. Data Layer

This layer is used to connect the business layer to the database or data source. It contains methods which are used to perform operations on database like insert, delete, update, etc. This layer contains stored procedures which are used to query database. Hence this layer establishes a connection with the database and performs functions on the database.

Advantages:

- Updation to new graphical environments is easier and faster.
- Easy to maintain large and complex projects.
- It provides logical separation between presentation layer, business layer and data layer.
- By introducing application layer between data and presentation layer, it provides more security to database layer hence data will be more secure.
- You can hide unnecessary methods from the business layer in the presentation layer.
- Data provider queries can be easily updated and OOP’s concept can be easily applied on project.

Disadvantages:

- It is more complex than simple client-server architecture.
- It is difficult to build and time consuming.
- User should have good knowledge of object-oriented concepts.

10 a .

HTTP, the protocol that takes care of the communication between a server and a client on the web, is known as a stateless protocol. In other words, if a user requests two pages on a server, no information will be shared between these two requests automatically. Instead, a developer will have to rely on something called cookies (or sessions - more on those later) to share information between the requests. This is extremely useful in a lot of situations, e.g. to keep a user logged in between several requests etc.

Pretty much all server-side technologies have built-in support for handling cookies and of course the ASP.NET MVC framework does as well. In fact, dealing with cookies is pretty easy, thanks to the functionality found on the HttpContext class. In this article, you'll learn how to set a cookie and then read it again afterwards, so that you will be able to share information about a specific visitor between his/hers requests.

What is a cookie?

A cookie is basically a physical, plain-text file stored by the client (usually a browser), tied to a specific website. The client will then allow this specific website to read the information stored in this file on subsequent requests, basically allowing the server (or even the client itself) to store information for later use.

Setting and reading cookies

Setting a cookie, and reading it again later on, with ASP.NET MVC is very, very easy. Here's how you can send a cookie to the client, in its most basic form:

```
HttpContext.Response.Cookies.Append("user_id", "1");
```

Notice how I use the **Response** property on the **HttpContext** class, where I can access the **Cookies** property. Using the Append() method, I can add a Cookie to the output, by supplying a name and a value for it. The name (user_id) is what we'll use later on to get the value (1) back. And getting the actual value back is just as easy:

```
var userId = HttpContext.Request.Cookies["user_id"];
```

Notice how I now use the **Request** property instead of the *Response*, when reading the value back - the reason is that setting a cookie is done by adding information to the response sent to the client, while reading it means that you are pulling information from the request made by the client (a normal browser will automatically send all relevant cookies with each request).

So, now that you know how to write and read cookie information, let's combine that into a very simple example, where we add a piece of information as a cookie if it's not already present - this will be a great way to know if a user has visited the page before or not:

```
public class CookiesController : Controller
{
    public IActionResult Index()
    {
    if(!HttpContext.Request.Cookies.ContainsKey("first_request"))
    {
        HttpContext.Response.Cookies.Append("first_request", DateTime.Now.ToString());
        return Content("Welcome, new visitor!");
    }
    else
    {
        DateTime firstRequest = DateTime.Parse(HttpContext.Request.Cookies["first_request"]);
        return Content("Welcome back, user! You first visited us on: " + firstRequest.ToString());
    }
    }
}
```

Our Controller action now produces response based on whether the visitor has visited the page before, by always checking for the presence of a cookie with the name of "first_request" - if this cookie is not present, we add it, while setting the value to the current date and time. If the cookie is present, we can assume that the visitor has visited the page before and we can even tell the visitor when the first visit occurred.

So, this is basic cookies in a nutshell and now you know how to write and read them. But there's a bit more you need to know, if you want to take full advantage of cookies.

CookieOptions

As an optional third parameter to the *Append()* method we just used, you can pass an instance of the **CookieOptions** class. It allows you to adjust several important aspects of your cookie, e.g. how long it should stay alive and stuff like domain(s) and path. Allow me to go through the most important properties, but first, make sure that you have included the *Microsoft.AspNetCore.Http* namespace, where the CookieOptions class lives:

```
using Microsoft.AspNetCore.Http;
```

Now you can create an instance of the CookieOptions class, which will use the default settings, and then pass it to the Append() method:

```
CookieOptions cookieOptions = new CookieOptions();
HttpContext.Response.Cookies.Append("first_request", DateTime.Now.ToString(),
cookieOptions);
```

Before passing it though (so, between the two lines of code), you likely want to change some of the options. Here are the most relevant ones:

CookieOptions.Expires

By default, your cookie will be a so-called session cookie, meaning that it will only live as long as the browser remains open - once the browser is closed, the cookie is deleted by the browser. However, you are free to change this behavior using the **Expires** property. This property is a DateTimeOffset instance, which makes it easy to configure an expiry time, like this:

```
cookieOptions.Expires = new DateTimeOffset(DateTime.Now.AddDays(7));
```

This will make the cookie expire in 7 days. You can of course adjust this using the methods like AddDays(), AddHours() and so on.

CookieOptions.Domain

By default, the cookie will be set for the domain which made the request. So, if your page is accessed by the domain mywebsite.com, the cookie will be set for mywebsite.com. If your page is accessed using a subdomain, this subdomain will be used, and this is important, because a subdomain could be "www". So, if your page is accessed by www.mywebsite.com, the cookie will, by default, **only be accessible from www.mywebsite.com and NOT mywebsite.com**. Therefore, it can be a good idea to set the Domain property to the base-domain of your website, prefixed with a dot, like this:

```
cookieOptions.Domain = ".mywebsite.com";
```

Now your cookie will be accessible from mywebsite.com as well as all possible subdomains. On the other hand, if you don't have full control over the domain, you may want to limit the domain of the cookie to the specific (sub)domain you control.

CookieOptions.Path

By default, the Path of the cookie will be set to "/", which basically means that the cookie will be valid for all pages of the website. However, under certain conditions, you may need a cookie that's valid only for a specific page or folder. This is easily accomplished with the Path property:

```
cookieOptions.Path = "/users/";
```

With that in place, the cookie will now only be visible and readable to pages in the "users" folder, as well as sub-folders to it.

More properties of CookieOptions

There are several other interesting properties to be found on CookieOptions, like IsEssential, HttpOnly and Secure. If you want to know more about them, I suggest that you have a look at the documentation for CookieOptions.

Using Cookies to Maintain Sessions in ASP

- Article
- 06/16/2017
- 7 minutes to read

A cookie is a token that the Web server embeds in a user's Web browser to identify the user. The next time the same browser requests a page, it sends the cookie it received from the Web server. Cookies allow a set of information to be associated with a user. ASP scripts can both get and set the values of cookies by using the Response.Cookies Collection collection of the **Response** and **Request** objects.

About SessionID and Cookies

The first time a user requests an .asp file within a given application, ASP generates a SessionID. A number produced by a complex algorithm, the SessionID uniquely identifies each user's session. At the beginning of a new session, the server stores the Session ID in the user's Web browser as a cookie.

The SessionID cookie is similar to a locker key in that, as the user interacts with an application during a session, ASP can store information for the user in a "locker" on the server. The user's SessionID cookie, transmitted in the HTTP request header, enables access to this information in the way that a locker key enables access to a locker's contents. Each time that ASP receives a request for a page, it checks the HTTP request header for a SessionID cookie.

After storing the SessionID cookie in the user's browser, ASP reuses the same cookie to track the session, even if the user requests another .asp file, or requests an .asp file running in other application. Likewise, if the user deliberately abandons or lets the session timeout, and then proceeds to request another .asp file, ASP begins a new session using the same cookie. The only time a user receives a new SessionID cookie is when the server administrator restarts the server, thus clearing the SessionID settings stored in memory, or the user restarts the Web browser.

By reusing the SessionID cookie, ASP minimizes the number of cookies sent to the browser. Additionally, if you determine that your ASP application does not require session management, you can prevent ASP from tracking session and sending SessionID cookies to users.

ASP will not send the session cookies under the following conditions:

- If an application has session state disabled.
- If an ASP page is defined as sessionless, that is, a page containing the

  Copy

  ```
  <%@ EnableSessionState=False %>
  ```

  tag. For more information, see Sessionless ASP Pages.

You should also note that SessionID cookies are not intended to provide a permanent or secure means for tracking users across multiple visits to a Web site. The SessionID information stored in the server computer's memory can be easily lost or impersonated by a malicious user. If you want track users who visit your Web application over a longer periods, you must create a user identification by storing a special cookie in a user's Web browser and saving the cookie information to a database. If you do so, configure your application to use SSL in order to encrypt the SessionID and protect it from malicious users. For more information, see Using Cookies and see "Secure Sockets Layer" in IIS Help, which is accessible from IIS Manager..

Setting Cookies

To set the value of a cookie, use **Response.Cookies**. If the cookie does not already exist, **Response.Cookies** creates a new one. For example, to send a cookie named ("VisitorID") with an associated value ("49") to the browser, use the following command, which must appear on your Web page before the <HTML> tag:

Copy

```
<% Response.Cookies("VisitorID") = 49 %>
```

Copy

Copy

If you want a cookie to be used only during the current user session, then sending the cookie to the browser is all you need to do. However, if you want to identify a user even after the user has stopped and restarted the browser, you must force the browser to store the cookie in a file on the client computer's hard disk. To save the cookie, use the **Expires** attribute for **Response.Cookies** and set the date to some date in the future:

Copy

```
<%
```

Copy

```
  Response.Cookies("VisitorID") = 49
  Response.Cookies("VisitorID").Expires = "December 31, 2001"
%>
```

Copy

A cookie can have multiple values; such a cookie is called an indexed cookie. An indexed cookie value is assigned a key; you can set a particular cookie key value. For example:

Copy

```
<% Response.Cookies("VisitorID")("49") = "Travel" %>
```

Copy

Copy

If an existing cookie has key values but **Response.Cookies** does not specify a key name, then the existing key values are deleted. Similarly, if an existing cookie does not have key values but **Response.Cookies** specifies key names and values, the existing value of the cookie is deleted and new key-value pairs are created.

Getting Cookies

To get the value of a cookie, use the **Request.Cookies** collection. For example, if the user HTTP request sets

Copy

```
VisitorID=49
```

Copy

, then the following statement retrieves the value

Copy

49
Copy

:

Copy

```
<%= Request.Cookies("VisitorID") %>
```
Copy

Copy

Similarly, to retrieve a key value from an indexed cookie, use the key name. For example, if a user's browser sends the following information in the HTTP request header:

Copy

```
Cookie: VisitorID=49=Travel
```
Copy

The following statement would then return the value

Copy

Travel
Copy

:

Copy

```
<%= Request.Cookies("VisitorID")("49") %>
```
Copy

Copy

Setting Cookie Paths

Each cookie stored by ASP on the user's Web browser contains path information. When the browser requests a file stored in the same location as the path specified in the cookie, the browser automatically forwards the cookie to the server. By default, cookie paths correspond to the name of the application containing the .asp file that originally generated the cookie. For example, if an .asp file, residing in an application called UserApplication, generates a cookie, then each time a user's Web browser retrieves any file residing in that application, the browser will forward the cookie, in addition to any other cookies containing the path /UserApplication.

To specify a path for a cookie other than the default application path, you can use the ASP **Response.Cookies** collection's **Path** attribute. For example, the following script assigns the path SalesApp/Customer/Profiles/ to a cookie called

Copy

Purchases
Copy

:

Copy

```
<%
```

Copy

```
  Response.Cookies("Purchases") = "12"
  Response.Cookies("Purchases").Expires = "January 1, 2001"
  Response.Cookies("Purchases").Path = "/SalesApp/Customer/Profiles/"
%>
```

Copy

Whenever the Web browser containing the

Copy

Purchases
Copy

cookie requests a file residing in the path /SalesApp/Customer/Profiles/ or in any of it subdirectories, the browser forwards the cookie to the server.

Many Web browsers, including Microsoft Internet Explorer version 4.0, or later, and Netscape browsers, preserve the case of the cookie path. This means that if the case of the path of a requested file differs from the case of the stored cookie path, the browser will not send the cookie to the server. For example, to ASP, the virtual directories /TRAVEL and /travel are the same ASP application, but to a browser that preserves the case of a URL, /TRAVEL and /travel are two different applications. Make sure all URLs to .asp files have the same case to ensure that the user's browser forwards stored cookies.

You can use the following statement to set the cookie path so that the user's Web browser will forward a cookie whenever the browser requests a file from your server, regardless of application or path:

Copy

```
Response.Cookies("Purchases").Path = "/"
```

Copy

Copy

Note, however, that forwarding cookies to the server, without distinguishing between applications, raises a potential security concern if the cookies contain sensitive information that should not be accessible outside of a specific application.

Preserving State without Cookies

Not all browsers support cookies. Even with browsers that do support cookies, some users prefer to turn off cookie support. If your application needs to be responsive to browsers that don't support cookies, you cannot use ASP session management.

In this case, you must write your own mechanism to pass information from page to page in your application. There are two general ways to do this:

- Add parameters to a URL's query string. For example:

  Copy

  ```
  http://MyServer/MyApp/start.asp?name=Jeff
  ```

  Some browsers, however, will discard any explicit parameters passed in a query string if a form is submitted with the GET method.

- Add hidden values to a form. For example, the following HTML form contains a hidden control, which does not appear on the actual form and remains invisible in the user's Web browser. The form passes a user identification value, in addition to the information supplied by the user, by using the HTTP POST method.

  Copy

  ```
  <FORM METHOD="POST" ACTION="/scripts/inform.asp">
  <INPUT TYPE="text" NAME="city" VALUE="">
  <INPUT TYPE="text" NAME="country_region" VALUE ="">
  <INPUT TYPE="hidden" NAME="userid" VALUE= <%= UserIDNum(i) %>
  <INPUT TYPE="submit"  VALUE="Enter">
  ```

  This method requires all link destinations that pass user information to be coded as HTML forms.

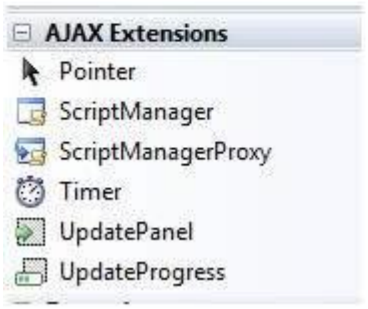
If you are not using ASP session management, you should turn off session support for your application. When sessions are enabled, ASP sends a SessionID cookie to each browser that requests a page. To turn off session support, clear the **Enable Session State** check box on the **Application Options** property sheet in the Internet Information Services snap-in.

10 b .

AJAX stands for Asynchronous JavaScript and XML. This is a cross platform technology which speeds up response time. The AJAX server controls add script to the page which is executed and processed by the browser.

However like other ASP.NET server controls, these AJAX server controls also can have methods and event handlers associated with them, which are processed on the server side.

The control toolbox in the Visual Studio IDE contains a group of controls called the 'AJAX Extensions'



The ScriptManager Control

The ScriptManager control is the most important control and must be present on the page for other controls to work.

It has the basic syntax:

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
</asp:ScriptManager>
```

If you create an 'Ajax Enabled site' or add an 'AJAX Web Form' from the 'Add Item' dialog box, the web form automatically contains the script manager control. The ScriptManager control takes care of the client-side script for all the server side controls.

The UpdatePanel Control

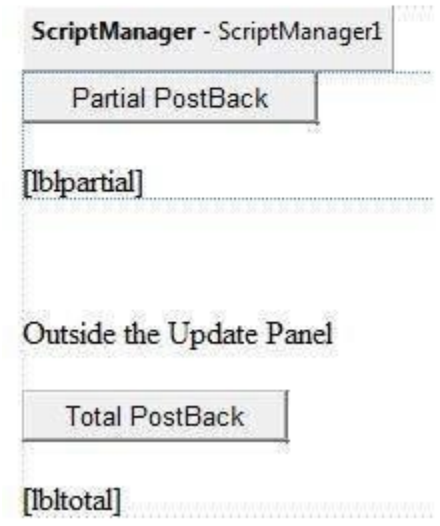
The UpdatePanel control is a container control and derives from the Control class. It acts as a container for the child controls within it and does not have its own interface. When a control inside it triggers a post back, the UpdatePanel intervenes to initiate the post asynchronously and update just that portion of the page.

For example, if a button control is inside the update panel and it is clicked, only the controls within the update panel will be affected, the controls on the other parts of the page will not be affected. This is called the partial post back or the asynchronous post back.

Example

Add an AJAX web form in your application. It contains the script manager control by default. Insert an update panel. Place a button control along with a label control within the update panel control. Place another set of button and label outside the panel.

The design view looks as follows:



The source file is as follows:

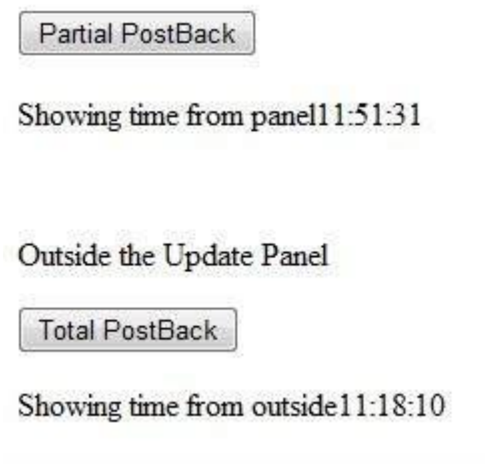
```
<form id="form1" runat="server">
   <div>
      <asp:ScriptManager ID="ScriptManager1" runat="server" />
   </div>

   <asp:UpdatePanel ID="UpdatePanel1" runat="server">
      <ContentTemplate>
         <asp:Button ID="btnpartial" runat="server" onclick="btnpartial_Click" Text="Partial
PostBack"/>
         <br />
         <br />
         <asp:Label ID="lblpartial" runat="server"></asp:Label>
      </ContentTemplate>
   </asp:UpdatePanel>

   <p> </p>
   <p>Outside the Update Panel</p>
   <p>
      <asp:Button ID="btntotal" runat="server" onclick="btntotal_Click" Text="Total PostBack"
/>
   </p>

   <asp:Label ID="lbltotal" runat="server"></asp:Label>
</form>
```

Both the button controls have same code for the event handler:

```
string time = DateTime.Now.ToLongTimeString();
lblpartial.Text = "Showing time from panel" + time;
```

```
lbltotal.Text = "Showing time from outside" + time;
```

Observe that when the page is executed, if the total post back button is clicked, it updates time in both the labels but if the partial post back button is clicked, it only updates the label within the update panel.



Showing time from panel11:51:31

Outside the Update Panel

Showing time from outside11:18:10

A page can contain multiple update panels with each panel containing other controls like a grid and displaying different part of data.

When a total post back occurs, the update panel content is updated by default. This default mode could be changed by changing the UpdateMode property of the control. Let us look at other properties of the update panel.

Properties of the UpdatePanel Control

The following table shows the properties of the update panel control:

Properties	Description
ChildrenAsTriggers	This property indicates whether the post backs are coming from the child controls, which cause the update panel to refresh.
ContentTemplate	It is the content template and defines what appears in the update panel when it is rendered.
ContentTemplateContainer	Retrieves the dynamically created template container object and used for adding child controls programmatically.
IsInPartialRendering	Indicates whether the panel is being updated as part of the partial post back.

RenderMode	Shows the render modes. The available modes are Block and Inline.
UpdateMode	Gets or sets the rendering mode by determining some conditions.
Triggers	Defines the collection trigger objects each corresponding to an event causing the panel to refresh automatically.

Methods of the UpdatePanel Control

The following table shows the methods of the update panel control:

Methods	Description
CreateContentTemplateContainer	Creates a Control object that acts as a container for child controls that define the UpdatePanel control's content.
CreateControlCollection	Returns the collection of all controls that are contained in the UpdatePanel control.
Initialize	Initializes the UpdatePanel control trigger collection if partial-page rendering is enabled.
Update	Causes an update of the content of an UpdatePanel control.

The behavior of the update panel depends upon the values of the UpdateMode property and ChildrenAsTriggers property.

UpdateMode	ChildrenAsTriggers	Effect
Always	False	Illegal parameters.
Always	True	UpdatePanel refreshes if whole page refreshes or a child control on it posts back.
Conditional	False	UpdatePanel refreshes if whole page refreshes or a triggering control outside it initiates a refresh.

Conditional	True	UpdatePanel refreshes if whole page refreshes or a child control on it posts back or a triggering control outside it initiates a refresh.

The UpdateProgress Control

The UpdateProgress control provides a sort of feedback on the browser while one or more update panel controls are being updated. For example, while a user logs in or waits for server response while performing some database oriented job.

It provides a visual acknowledgement like "Loading page...", indicating the work is in progress.

The syntax for the UpdateProgress control is:

```
<asp:UpdateProgress ID="UpdateProgress1" runat="server" DynamicLayout="true"
AssociatedUpdatePanelID="UpdatePanel1" >

  <ProgressTemplate>
    Loading...
  </ProgressTemplate>

</asp:UpdateProgress>
```

The above snippet shows a simple message within the ProgressTemplate tag. However, it could be an image or other relevant controls. The UpdateProgress control displays for every asynchronous postback unless it is assigned to a single update panel using the AssociatedUpdatePanelID property.

Properties of the UpdateProgress Control

The following table shows the properties of the update progress control:

Properties	Description
AssociatedUpdatePanelID	Gets and sets the ID of the update panel with which this control is associated.
Attributes	Gets or sets the cascading style sheet (CSS) attributes of the UpdateProgress control.
DisplayAfter	Gets and sets the time in milliseconds after which the progress template is displayed. The default is 500.

DynamicLayout	Indicates whether the progress template is dynamically rendered.
ProgressTemplate	Indicates the template displayed during an asynchronous post back which takes more time than the DisplayAfter time.

Methods of the UpdateProgress Control

The following table shows the methods of the update progress control:

Methods	Description
GetScriptDescriptors	Returns a list of components, behaviors, and client controls that are required for the UpdateProgress control's client functionality.
GetScriptReferences	Returns a list of client script library dependencies for the UpdateProgress control.

The Timer Control

The timer control is used to initiate the post back automatically. This could be done in two ways:

(1) Setting the Triggers property of the UpdatePanel control:

```
<Triggers>
  <asp:AsyncPostBackTrigger ControlID="btnpanel2" EventName="Click" />
</Triggers>
```

(2) Placing a timer control directly inside the UpdatePanel to act as a child control trigger. A single timer can be the trigger for multiple UpdatePanels.

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server" UpdateMode="Always">

  <ContentTemplate>
    <asp:Timer ID="Timer1" runat="server" Interval="1000">
      </asp:Timer>

    <asp:Label ID="Label1" runat="server" Height="101px" style="width:304px" >
      </asp:Label>
  </ContentTemplate>

</asp:UpdatePanel>
```