

MS . NET



Mentor as a Service

Introduction to .Net Framework

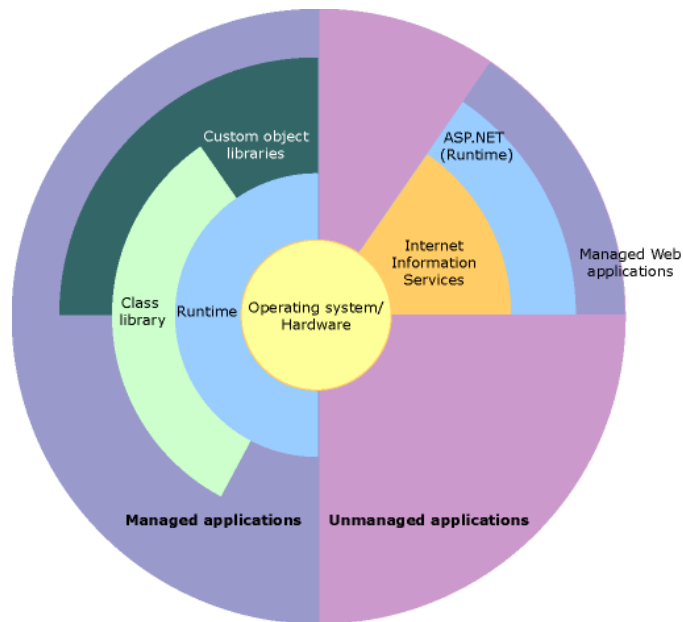
The .NET Framework is a development platform for building apps for Windows, Windows Phone, Windows Server, and Microsoft Azure.

Objectives of .NET Framework 4.5

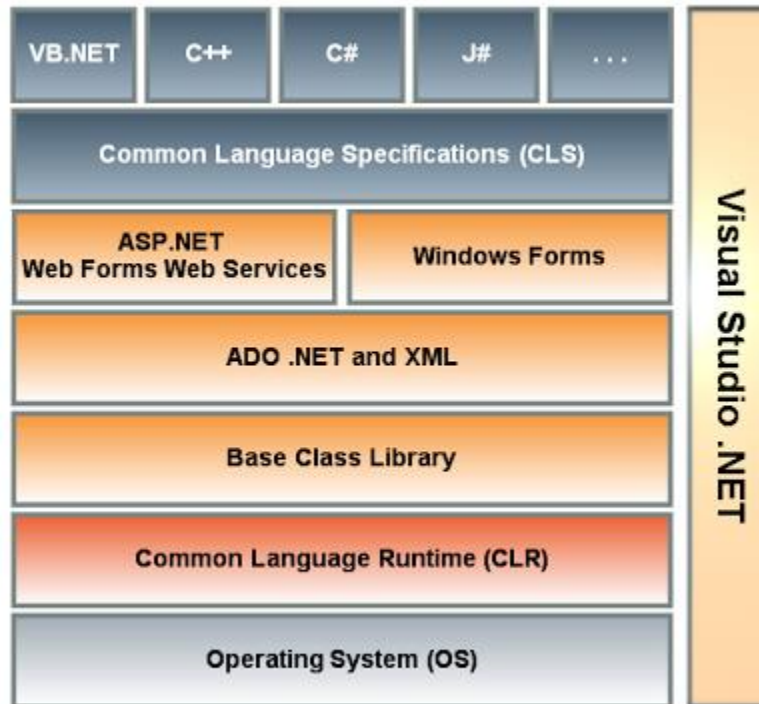
The .NET Framework is designed to fulfill the following objectives:

- To provide a consistent object-oriented programming environment whether object code is stored and executed locally, executed locally but Internet-distributed, or executed remotely.
- To provide a code-execution environment that minimizes software deployment and versioning conflicts.
- To provide a code-execution environment that promotes safe execution of code, including code created by an unknown or semi-trusted third party.
- To provide a code-execution environment that eliminates the performance problems of scripted or interpreted environments.
- To make the developer experience consistent across widely varying types of applications, such as Windows-based applications and Web-based applications.
- To build all communication on industry standards to ensure that code based on the .NET Framework can integrate with any other code.

.NET Framework Context



Dot Net Framework Components



Development in .NET Framework 4.5

Visual Studio is an Integrated Development Environment.

Application Life Cycle Management (ALM) can be managed using Visual Studio .NET

Design Develop Test Debug Deploy

- **Visual Studio .NET 2015 as IDE Tool**

The Design goals are:

- Maximize Developer Productivity
- Simplify Server based Development
- Deliver Powerful Design, Development Tools

RAD (Rapid Application Development Tool) for the next generation internet solutions
Enhanced RAD Support for creating Custom components for Business solutions.

- Tools for creating Rich Client Applications
- Tools for creating ASP.NET web sites
- Tools for Creating S-a-a-S modules
- Tools for connecting remote servers
- Tools for Cloud Connectivity
- Tools for Data Access
- Tools for Creating, Developing, Testing and Deploying Solutions.
- Help and Documentation
- Multiple .NET Language Support

Visual Studio Solutions and Projects

Visual Studio Solution consist of multiple Projects

Visual Studio Project consist of Source code, Resources.

- **C# as .NET Programming Language**

- C ++ Heritage
 - Namespaces, Pointers (in unsafe code),
 - Unsigned types, etc.
- Increased Productivity
 - Short Learning curve
- C# is a type safe Object Oriented Programming Language
- C# is case sensitive
- Interoperability
 - C# can talk to COM, DLLs and any of the .NET Framework languages

Structure of first C# program

```
using System;
// A "Hello World!" program in C#
public class HelloWorld
{ public static void Main ()
  {
    Console.WriteLine ("Hello, World");
  }
}
```

Passing Command Line Arguments

```
using System;

/* Invoke exe using console */

public class HelloWorld
{
  public static void Main (string [] args)
  {
    Console.WriteLine ("parameter count = {0}", args.Length);
    Console.WriteLine ("Hello {0}", args [0]);
    Console.ReadLine ();
  }
}
```

Execution of .NET Application

C# code is compiled by CSC.exe (C# compiler) into assembly as Managed code.

Managed code is targeted to Common Language Runtime of .NET Framework

Common Language Runtime converts MSIL code into Platform dependent executable code (native code) for targeted operating System.

Application is executed by Operating System as Process.

C# Types

A C# Program is a collection of types

Structure, Classes, Enumerations, Interfaces, Delegates, Events

C# provides a set of predefined types

e.g. int, byte, char, string, object, etc.

Custom types can be created.

All data and code is defined within a type.

No global variables, no global function.

Types can be instantiated and used by

Calling methods, setters and getters, etc.

Types can be converted from one type to another.

Types are organized into namespaces, files, and assemblies.

Types are arranged in hierarchy.

In .NET Types are of two categories

Value Type

Directly contain data on Stack.

Primitives:	int num; float speed;
Enumerations:	enum State {off, on}
Structures:	struct Point {int x, y ;}

Reference Types

Contain reference to actual instance on managed Heap.

Root	Object
String	string
Classes	class Line: Shape{ }
Interfaces	interface IDrawble {...}
Arrays	string [] names = new string[10];
Delegates	delegate void operation ();

Type Conversion

Implicit Conversion

No information loss

Occur automatically

Explicit Conversion

Require a cast

May not succeed

Information (precision) might be lost

```
int x=543454;  
long y=x;           //implicit conversion  
short z=(short)x;   //explicit conversion  
double d=1.3454545345;  
float f= (float) d;  //explicit conversion  
long l= (long) d     // explicit conversion
```

Constants and read only variables

```
// This example illustrates the use of constant data and readonly fields.

using System;
using System.Text;

namespace ConstData
{
    class MyMathClass
    {
        public static readonly double PI;
        static MyMathClass()
        { PI = 3.14; }
    }

    class Program
    {
        static void Main(string [] args)
        {
            Console.WriteLine ("***** Fun with Const *****\n");
            Console.WriteLine ("The value of PI is: {0}", MyMathClass.PI);

            // Error! Can't change a constant!
            // MyMathClass.PI = 3.1444;

            LocalConstStringVariable ();
        }

        static void LocalConstStringVariable()
        {
            // A local constant data point.
            const string fixedStr = "Fixed string Data";
            Console.WriteLine(fixedStr);

            // Error!
            //fixedStr = "This will not work!";
        }
    }
}
```


Enumerations

Enumerations are user defined data Type which consist of a set of named integer constants.

```
enum Weekdays { Mon, Tue, Wed, Thu, Fri, Sat}
```

Each member starts from zero by default and is incremented by 1 for each next member.

Using Enumeration Types

```
Weekdays day=Weekdays.Mon;  
Console.WriteLine("{0}", day);    //Displays Mon
```

Structures

Structure is a value type that is typically used to encapsulate small groups of related variables.

```
public struct Point  
{  
    public int x;  
    public int y;  
}
```

Arrays

Declare

```
int [] marks;
```

Allocate

```
int [] marks= new int [9];
```

Initialize

```
int [] marks=new int [] {1, 2, 3, 4, 5, 6, 7, 9};  
int [] marks={1,2,3,4,5,6,7,8,9};
```

Access and assign

```
Marks2[i] = marks[i];
```

Enumerate

```
foreach (int i in marks) {Console.WriteLine (i); }
```

Params Keyword

It defines a method that can accept a variable number of arguments.

```
static void ViewNames (params string [] names)
{
    Console.WriteLine ("Names: {0}, {1}, {2}",
                       names [0], names [1], names [2]);
}
public static void Main (string [] args)
{
    ViewNames("Nitin", "Nilesh", "Shrinivas");
}
```

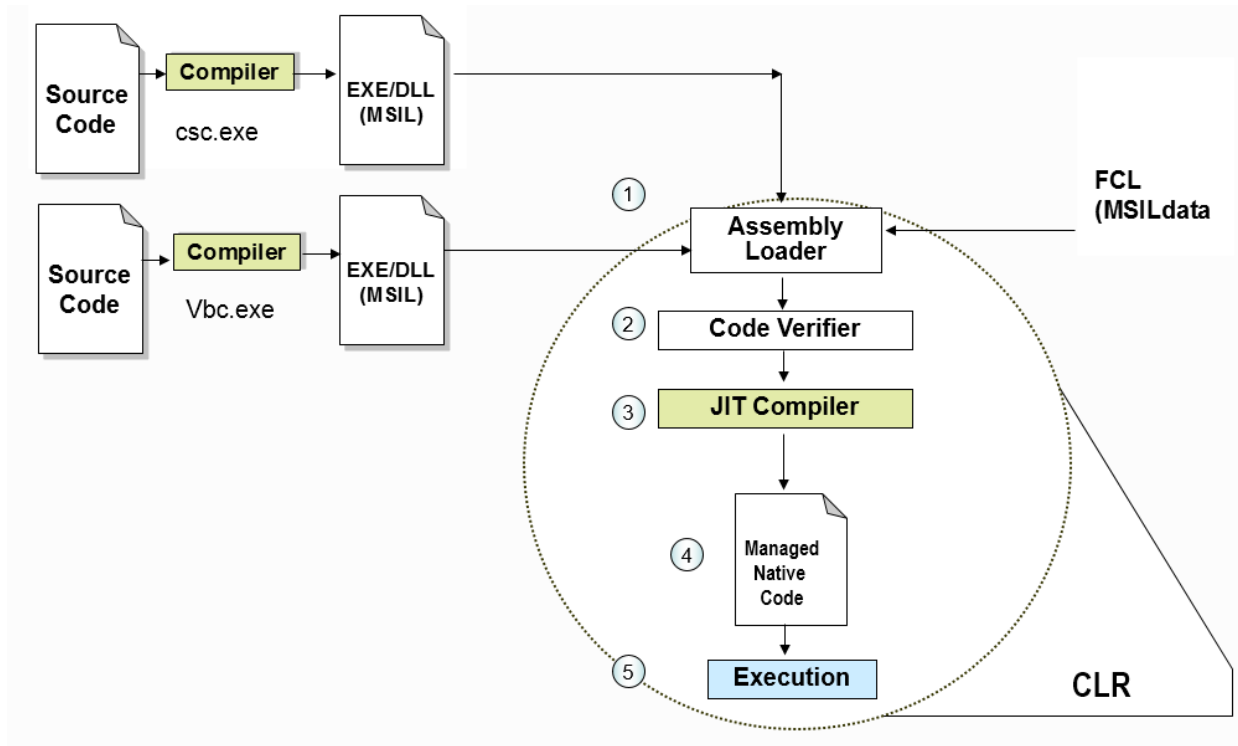
ref and out parameters

```
void static Swap (ref int n1, ref int n2)
{
    int temp =n1; n1=n2; n2=temp;
}

void static Calculate (float radius, out float area, out float circum)
{
    Area=3.14f * radius * radius;
    Circum=2*3.14f * radius;
}

public static void Main ()
{
    int x=10, y=20;
    Swap (ref x, ref y);
    float area, circum;
    Calculate (5, out area, out circum);
}
```

Execution Process in .NET Environment



.NET Assembly

A Logical unit of Deployment on .NET Platform.

Components of Assembly

- Manifest
- Metadata
- MSIL code
- Resources

Types of Assemblies

Private Assembly (bin folder)
 Shared Assembly (GAC)
 System.Data.dll
 System.Drawing.dll
 System.Web.dll
 Etc.

Windows vs. .NET executables

Windows Exe consist of native code
 .NET Exe consist of MSIL code

Inside .NET Framework

Common Language Runtime (CLR):

CLR is the heart of the .NET framework and it does 4 primary important things:

1. Garbage collection
2. CAS (Code Access Security)
3. CV (Code Verification)
4. IL to Native translation.

Common Type System (CTS): -

CTS ensure that data types defined in two different languages get compiled to a common data type. This is useful because there may be situations when we want code in one language to be called in other language. We can see practical demonstration of CTS by creating same application in C# and VB.Net and then compare the IL code of both applications. Here the data type of both IL code is same.

Common Language Specification (CLS):

CLS is a subset of CTS. CLS is a set of rules or guidelines. When any programming language adheres to these set of rules it can be consumed by any .Net language.

e.g. Lang must be object oriented, each object must be allocated on heap,

Exception handling supported.

Also each data type of the language should be converted into CLR understandable types by the Lang compiler.

All types understandable by CLR forms CTS (common type system) which includes:

System.Byte, System.Int16, System.UInt16,
System.Int32, System.UInt32, System.Int64,
System.UInt64, System.Char, System.Boolean, etc.

Assembly Loader:

When a .NET application runs, CLR starts to bind with the version of an assembly that the application was built with. It uses the following steps to resolve an assembly reference:

- 1) Examine the Configuration Files
- 2) Check for Previously Referenced Assemblies
- 3) Check the Global Assembly Cache
- 4) Locate the Assembly through Codebases or Probing

MSIL Code Verification and Just In Time compilation (JIT)

When .NET code is compiled, the output it produces is an intermediate language (MSIL) code that requires a runtime to execute the IL. So during assembly load at runtime, it first validates the metadata then examines the IL code against this metadata to see that the code is type safe. When MSIL code is executed, it is compiled Just-in-Time and converted into a platform-specific code that's called native code.

Thus any code that can be converted to native code is valid code. Code that can't be converted to native code due to unrecognized instructions is called Invalid code. During JIT compilation the code is verified to see if it is type-safe or not.

Garbage Collection

“Garbage” consists of objects created during a program’s execution on the managed heap that are no longer accessible by the program. Their memory can be reclaimed and reused with no adverse effects.

The garbage collector is a mechanism which identifies garbage on the managed heap and makes its memory available for reuse. This eliminates the need for the programmer to manually delete objects which are no longer required for program execution. This reuse of memory helps reduce the amount of total memory that a program needs to run.

IL dis-assembler

The IL Disassembler is a companion tool to the IL Assembler (**Ilasm.exe**). Ildasm.exe takes a portable executable (PE) file that contains intermediate language (IL) code and creates a text file suitable as input to Ilasm.exe.

.NET Framework Folder

C:\Program Files (x86)\Reference Assemblies\Microsoft\Framework

Object Orientation in C#

Object

A real world entity which has well defined structure and behavior.

Characteristics of an Object are:

- State
- Behavior
- Identity
- Responsibility

Pillars of Object Orientation

- Abstraction
- Encapsulation
- Inheritance
- Typing, Concurrency, Hierarchy, Persistence

Abstraction

Getting essential characteristic of a System depending on the perspective of on Observer.
Abstraction is a process of identifying the key aspects of System and ignoring the rest
Only domain expertise can do right abstraction.

Abstraction of Person Object

- Useful for social survey
- Useful for healthcare industry
- Useful for Employment Information

Encapsulation

Hiding complexity of a System.

Encapsulation is a process of compartmentalizing the element of an abstraction that constitute its structure and behavior.

Servers to separate interface of an abstraction and its implementation.

User is aware only about its interface: any changes to implementation does not affect the user.

Inheritance

Classification helps in handing complexity.

Factoring out common elements in a set of entities into a general entity and then making it more and more specific.

Hierarchy is ranking or ordering of abstraction.

Code and Data Reusability in System using is a relationship.

Typing

Typing is the enforcement of the entity such that objects of different types may not be interchanges, or at the most, they may be interchanged only in restricted ways.

Concurrency

Different objects responding simultaneously.

Persistence

Persistence of an object through which its existence transcends time and or space.

Namespace and Class

Namespace is a collection .NET Types such as structure, class, interfaces, etc.

```
namespace EmployeeApp
{
    public class Employee
    {
        private string empName;
        private int empID;
        private float currPay;
        private int empAge;
        private string empSSN;
        private static string companyName;
        public Employee ()

        { empID=18; currPay=15000; }

        public Employee (int id, float basicSal)
        { empID=id; currPay= basicSal; }

        public ~Employee()
        { //DeInitializtion }

        public void GiveBonus(float amount)
        { currPay += amount; }

        public void DisplayStats()
        {
            Console.WriteLine("Name: {0}", empName);
            Console.WriteLine("ID: {0}", empID);
            Console.WriteLine("Age: {0}", empAge);
            Console.WriteLine("SSN: {0}", empSSN);
            Console.WriteLine("Pay: {0}", currPay);
        }
    }
}
```

Partial class

A class can be spread across multiple source files using the keyword partial.
All source files for the class definition are compiled as one file with all class members.
Access modifiers used for defining a class should be consistent across all files.

Properties (smart fields)

Have two assessors:

Get retrieves data member values.

Set enables data members to be assigned

```
public int EmployeeID
{
    get {return _id;}
    set {_id=value ;}
}
```

Indexers (smart array)

```
public class Books
{
    private string [] titles= new string [100];
    public string this [int index]
    {
        get{ if (index <0 || index >=100)
            return 0;
            else
                return titles [index];
        }
        set{
            if (! index <0 || index >=100)
                return 0;
            else
                titles [index] =value;
        }
    }
}
public static void Main ()
{
    Books mybooks=new Books ();
    Mybooks [3] ="Mogali in Jungle";
}
```


Singleton class

```
public class OfficeBoy
{
    private static OfficeBoy _ref = null;
    private int _val;
    private OfficeBoy() { _val = 10; }
    public int Val { get { return _val; }
                  set { _val = value; }
    }

    public static OfficeBoy GetObject ()
    {
        if (_ref == null)
            _ref = new OfficeBoy ();
        return _ref;
    }
}

static void Main(string[] args)
{
    OfficeBoy sweeper, waiter;
    string s1; float f1;
    sweeper = OfficeBoy.GetObject(); waiter = OfficeBoy.GetObject();
    sweeper.Val = 60;
    Console.WriteLine("Sweeper Value : {0}", sweeper.Val);
    Console.WriteLine("Waiter Value : {0}", waiter.Val);
    s1 = sweeper.Val.ToString();
    f1 = (float)sweeper.Val;
    sweeper.Val = int.Parse(s1);
    sweeper.Val = Convert.ToInt32(s1);
}
```

Arrays

Multidimensional Arrays (Rectangular Array)

```
int [ , ] mtrx = new int [2, 3];  
    Can initialize declaratively  
int [ , ] mtrx = new int [2, 3] { {10, 20, 30}, {40, 50, 60} }
```

Jagged Arrays

An Array of Arrays

```
int [ ] [ ] mtrxj = new int [2] [ ];
```

Must be initialize procedurally.

Nullable Types

```
class DatabaseReader  
{  
    public int? numericValue = null;  
    public bool? boolValue = true;  
    public int? GetIntFromDatabase() { return numericValue; }  
    public bool? GetBoolFromDatabase() { return boolValue; }  
}  
public static void Main (string[] args)  
{  
    DatabaseReader dr = new DatabaseReader();  
    int? i = dr.GetIntFromDatabase();  
    if (i.HasValue)  
        Console.WriteLine("Value of 'i' is: {0}", i.Value);  
    else  
        Console.WriteLine("Value of 'i' is undefined.");  
    bool? b = dr.GetBoolFromDatabase();  
    int? myData = dr.GetIntFromDatabase() ?? 100;  
    Console.WriteLine("Value of myData: {0}", myData.Value);  
}  
static void LocalNullableVariables ()  
{  
    int? nullableInt = 10;  
    double? nullableDouble = 3.14;  
    bool? nullableBool = null;  
    int?[] arrayOfNullableInts = new int?[10];  
    // Define some local nullable types using Nullable<T>.  
    Nullable<int> nullableInt = 10;  
    Nullable<double> nullableDouble = 3.14;  
    Nullable<bool> nullableBool = null;  
    Nullable<int> [] arrayOfNullableInts = new int?[10];  
}}}
```

Overloading

Method Overloading

Overloading is the ability to define several methods with the same name, provided each method has a different signature

```
public class MathEngine
{
    public static double FindSquare (double number) { // logic defined }
    public static double FindSquare (int number) { // another logic defined }
}
public static void Main ()
{
    double res= MathEngine.FindSquare(12.5);
    double num= MathEngine.FindSquare(12);
}
```

Operator Overloading

Giving additional meaning to existing operators.

```
public static Complex Operator + (Complex c1, Complex c2)
{
    Complex temp= new Complex();
    temp.real = c1.real+ c2.real;
    temp.imag = c1.image + c2.imag;
    return temp;
}
public static void Main ()
{
    Complex o1= new Complex (2, 3);
    Complex o2= new Complex (5, 4);
    Complex o3= 1+ o2;
    Console.WriteLine (o3.real + " " + o3.imag);
}
```

Operator overloading restrictions

Following operators cannot be overloaded.

Conditional logical &&,	Array indexing operator [], Cast Operators ()	Array indexing operator [] Cast Operators ()
Assignment operators +=,-=,*=,/= etc	=,.,? :,->, new, is, sizeof, typeof	The comparison operator, if overloaded, must be overloaded in pairs. If == is overloaded then != must also be overloaded

C # Reusability

Inheritance

Provides code reusability and extensibility.

Inheritance is a property of class hierarchy whereby each derived class inherits attributes and methods of its base class.

Every Manager is Employee.

Every Wage Employee is Employee.

```
class Employee
{
    public double CalculateSalary ()
        {return basic_sal + hra+ da ;}
}

class Manager: Employee
{
    public double CalculateIncentives ()
    {
        //code to calculate incentives
        Return incentives;
    }
}

static void Main ()
{
    Manager mgr =new Manager ();
    double Inc=mgr. CalculateIncentives ();
    double sal=mgr. CalculateSalary ();
}
```

Constructors in Inheritance

```
class Employee
{
    public Employee ()
    {
        Console.WriteLine ("in Default constructor") ;
    }
    public Employee (int eid, ....)
    {
        Console.WriteLine ("in Parameterized constructor") ;
    }
}

class Manager: Employee
{
    public Manager (): base () {.....}
    public Manager (int id): base (id,...) {....}
}
```

Polymorphism

Ability of different objects to responds to the same message in different ways is called Polymorphism.

```
horse.Move();  
car.Move();  
aeroplane.Move();
```

Virtual and Override

Polymorphism is achieved using virtual methods and inheritance.

Virtual keyword is used to define a method in base class and override keyword is used in derived class.

```
class Employee  
{ public virtual double CalculateSalary ()  
    {return basic_sal+ hra + da ;}  
}  
class Manager: Employee  
{  
    public override double CalculateSalary ()  
        {return (basic_sal+ hra + da + allowances);}  
}  
static void Main ()  
{ Employee mgr= new Manager ();  
    Double salary= mgr. CalculateSalary ();  
    Console.WriteLine (salary);  
}
```

Shadowing

Hides the base class member in derived class by using keyword new.

```
class Employee
{public virtual double CalculateSalary ()
    {return basic_sal;}
}
class SalesEmployee:Employee
{ double sales, comm;
  public new double CalculateSalary ()
    {return basic_sal+ (sales * comm) ;}
}
static void Main ()
{ SalesEmployee sper= new SalesEmployee ();
  Double salary= sper.CalculateSalary ();
  Console.WriteLine (salary);
}
```

Sealed class

Sealed class cannot be inherited

```
sealed class SinglyList
{
    public virtual double Add ()
    { // code to add a record in the linked list }
}
public class StringSinglyList:SinglyList
{
    public override double Add ()
    { // code to add a record in the String linked list }
}
```


Concrete class vs. abstract classes

Concrete class

Class describes the functionality of the objects that it can be used to instantiate.

Abstract class

Provides all of the characteristics of a concrete class except that it does not permit object instantiation.

An abstract class can contain abstract and non-abstract methods.

Abstract methods do not have implementation.

```
abstract class Employee
{
    public virtual double CalculateSalary();
    { return basic +hra + da ;}
    public abstract double CalculateBonus();
}

class Manager: Employee
{
    public override double CalculateSalary();
    { return basic + hra + da + allowances ;}
    public override double CalaculateBonus ()
    { return basic_sal * 0.20 ;}
}

static void Main ()
{
    Manager mgr=new Manager ();
    double bonus=mgr. CalaculateBonus ();
    double Salary=mgr. CalculateSalary ();
}
```

Object class

Base class for all .NET classes

Object class methods

- **public bool Equals(object)**
- **protected void Finalize()**
- **public int GetHashCode()**
- **public System.Type GetType()**
- **protected object MemberwiseClone()**
- **public string ToString()**

Polymorphism using Object

The ability to perform an operation on an object without knowing the precise type of the object.

```
void Display (object o)
{
    Console.WriteLine (o.ToString ());
}
public static void Main ()
{
    Display (34);
    Display ("Transflower");
    Display (4.453655);
    Display (new Employee ("Ravi", "Tambade"));
}
```

Interface Inheritance

For loosely coupled highly cohesive mechanism in Application.

An interface defines a Contract

Text Editor uses Spellchecker as interfaces.

EnglishSpellChecker and FrenchSpellChecker are implementing contract defined by SpellChecker interface.

```
interface ISpellChecker
{ ArrayList CheckSpelling (string word) ;}
class EnglishSpellChecker:ISpellChecker
{
    ArrayList CheckSpelling (string word)
        { // return possible spelling suggestions}
}
class FrenchSpellChecker:ISpellChecker
{
    ArrayList CheckSpelling (string word)
        { // return possible spelling suggestions}
}
class TextEditor
{
    public static void Main()
    {
        ISpellChecker checker= new EnglishSpellChecker ();
        ArrayList words=checker. CheckSpelling ("Flower");
        ...
    }
}
```

Explicit Interface Inheritance

```
interface IOrderDetails    { void ShowDetails() ;}
interface ICustomerDetails { void ShowDetails() ;}
class Transaction: IOrderDetails, ICustomerDetails
{
    void IOrderDetails. ShowDetails()
    { // implementation for interface IOrderDetails ;}
    void ICustomerDetails. ShowDetails()
    { // implementation for interface IOrderDetails ;}
}
public static void Main()
{
    Transaction obj = new Transaction();
    IOrderDetails od = obj;
    od.ShowDetails();
    ICustomerDetails cd = obj;
    cd.ShowDetails();
}
```

Abstract class vs. Interface

	Abstract class	Interface
Methods	At least one abstract method	All methods are abstract
Best suited for	Objects closely related in hierarchy.	Contract based provider model
Multiple Inheritance	Not supported	Supported
Component Versioning	By updating the base class all derived classes are automatically updated.	Interfaces are immutable

Building cloned Objects

```
class StackClass: ICloneable
{
    int size; int [] sArr;

    public StackClass (int s) { size=s; sArr= new int [size]; }

    public object Clone()
    {
        StackClass s = new StackClass(this.size);
        this.sArr.CopyTo(s.sArr, 0);
        return s;
    }
}

public static void Main()
{
    StackClass stack1 = new StackClass (4);
    Stack1 [0] = 89;
    ....
    StackClass stack2 = (StackClass) stack1.Clone ();
}
```

Reflection

Reflection is the ability to examine the metadata in the assembly manifest at runtime.

Reflection is useful in following situations:

- Need to access attributes in your programs metadata.
- To examine and instantiate types in an assembly.
- To build new types at runtime using classes in **System.Reflection.Emit** namespace.
- To perform late binding, accessing methods on types created at runtime.

System. Type class

Type class provides access to metadata of any .NET Type.

System. Reflection namespace

Contains classes and interfaces that provide a managed view of loaded types, methods and fields

These types provide ability to dynamically create and invoke types.

Type	Description
Module	Performs reflection on a module
Assembly	Load assembly at runtime and get its type
MemberInfo	Obtains information about the attributes of a member and provides access to member metadata

Assembly class

```
MethodInfo method;

string methodName;

object result = new object ();

object [] args = new object [] {1, 2};

Assembly asm = Assembly.LoadFile (@"c:/transflowerLib.dll");

Type [] types= asm.GetTypes();

foreach (Type t in types)
{
    method = t.GetMethod(methodName);

    string typeName= t.FullName;

    object obj= asm.CreateInstance(typeName);

    result = t.InvokeMember (methodName, BindingFlags.Public |
        BindingFlags.InvokeMethod | BindingFlags.Instance, null, obj, args);

    break;
}string res = result.ToString();

Console.WriteLine ("Result is: {0}", res);
```

Garbage Collection

COM

Programmatically implement reference counting and handle circular references

C++

Programmatically uses the new operator and delete operator

Visual Basic

Automation memory management

Manual vs. Automatic Memory Management

Common problems with manual memory management

- Failure to release memory
- Invalid references to freed memory

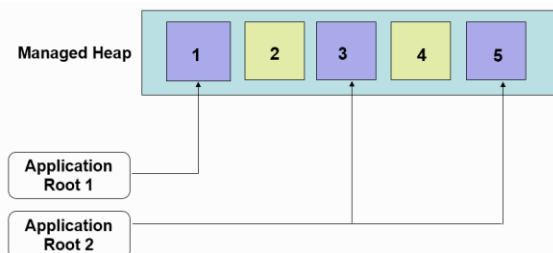
Automatic memory management provided with .NET Runtime

- Eases programming task
- Eliminates a potential source of bugs.

Garbage Collector

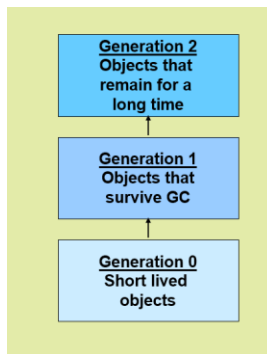
Manages the allocation and release of memory for your application.

Application Roots.



1. Identifies live object references or application roots and builds their graph
2. Objects not in the graph are not accessible by the application and hence considered garbage.
3. Finds the memory that can be reclaimed.
4. Move all the live object to the bottom of the heap, leaving free space at the top.
5. Looks for contiguous block objects & then shifts the non-garbage objects down in memory.
6. Updates pointers to point new locations.

Generational Garbage collection



Resource Management Types

Implicit Resource Management

With Finalize () method.

Will be required when an object encapsulates unmanaged resources like: file, window or network connection.

Explicit Resource Management

By implementing IDisposable Interface and writing Dispose method.

Implicit Resource Management with Finalization

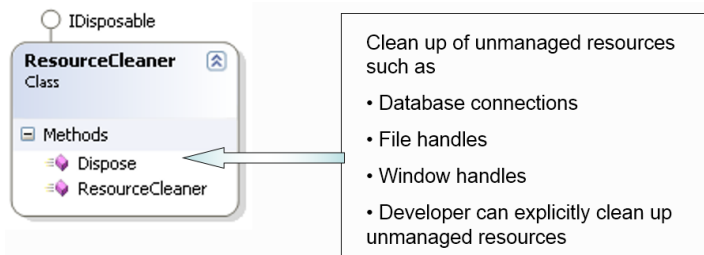
Writing Destructors in C#:

```
Class Car{  
    ~Car () //destructor { // cleanup statements}  
}
```


Explicit Resource Management

Implement IDisposable Interface.

Defines Dispose () method to release allocated unmanaged resources.



.NET Collection Framework

A Collection is a set of similarly typed objects that are grouped together.

System.Array class

The base class for all array Types.

```
int[] intArray= new int[5] { 22,11,33,44,55 };  
foreach (int i in intArray )  
{ Console.WriteLine( "\t {0}", i); }  
Array.Sort(intArray);  
Array.Reverse(intArray);
```

Collection Interfaces

Allow collections to support a different behavior

Interface	Description
IEnumertor	Supports a simple iteration over collection
IEnumerable	Supports foreach semantics
ICollection	Defines size, enumerators and synchronization methods for all collections.
IList	Represents a collection of objects that could be accessed by index
IComaprable	Defines a generalized comparison method to create a type-specific comparison
IComparer	Exposes a method that compares two objects.
IDictionary	Represents a collection of Key and value pairs

Implementing IEnumerable Interface

```
public class Team:IEnumerable
{
    private player [] players;
    public Team ()
    {
        Players= new Player [3];
        Players[0] = new Player("Sachin", 40000);
        Players[1] = new Player("Rahul", 35000);
        Players[2] = new Player("Mahindra", 34000);
    }

    public IEnumerator GetEnumerator ()
    {
        Return players.GetEnumerator();
    }
}

public static void Main()
{
    Team India = new Team();
    foreach(Player c in India)
    {
        Console.WriteLine (c.Name, c.Runs);
    }
}
```

Implementing ICollection Interface

To determine number of elements in a container.
Ability to copy elements into System.Array type

```
public class Team:ICollection
{
    private Players [] players;
    public Team() {.....}
    public int Count {get {return players.Count ;}
}

// other implementation of Team
}
```

```
public static void Main()
{
    Team India = new Team ();
    foreach (Player c in India)
    {
        Console.WriteLine (c.Name, c.Runs);
    }
}
```

Implementing IComparable Interface

```
public class Player:IComparable
{
    int IComparable.CompareTo(object obj)
    {
        Player temp= (Player) obj;
        if (this. Runs > temp.Runs)
            return 1;
        if (this. Runs < temp.Runs)
            return -1;
        else
            return 0;
    }
}
public static void Main()
{
    Team India = new Team();

    // add five players with Runs

    Arary.Sort(India);

    // display sorted Array
}
```

Using Iterator Method

```
public class Team
{
    private player [] players;
    public Team ()
    {
        Players= new Player [3];
        Players[0] = new Player("Sachin", 40000);
        Players[1] = new Player("Rahul", 35000);
        Players[2] = new Player("Mahindra", 34000);
    }

    public IEnumerator GetEnumerator()
    {
        foreach (Player p in players)
        {yield return p ;}
    }
}

public static void Main()
{
    Team India = new Team();
    foreach(Player c in India)
    {
        Console.WriteLine(c.Name, c.Runs);
    }
}
```

Collection Classes

ArrayList class

Represents list which is similar to a single dimensional array that can be resized dynamically.

```
ArrayList countries = new ArrayList ();
countries.Add("India");
countries.Add("USA");
countries.Add("UK");
countries.Add("China");
countries.Add("Nepal");

Console.WriteLine("Count: {0}", countries.Count);

foreach(object obj in countries)
{
    Console.WriteLine("{0}", obj);
}
```

Stack class

Represents a simple Last- In- First- Out (LIFO) collection of objects.

```
Stack numStack = new Stack();
numStack.Push(23);
numStack.Push(34);
numStack.Push(76);
numStack.Push(9);
Console.WriteLine("Element removed", numStack.Pop());
```

Queue class

Represents a first- in, first- out (FIFO) collection of objects.

```
Queue myQueue = new Queue ();
myQueue.Enqueue("Nilesh");
myQueue.Enqueue("Nitin");
myQueue.Enqueue("Rahul");
myQueue.Enqueue("Ravi");

Console.WriteLine("\t Capacity: {0}", myQueue.Capacity);
Console.WriteLine("(Dequeue) \t {0}", myQueue.Dequeue());
```

HashTable class

Represents a collection of Key/ value pairs that are organized based on the hash code of the key.

Each element is a key/ value pair stored in a DictionaryEntry object.

```
Hashtable h = new Hashtable();
h.Add("mo", "Monday");
h.Add("tu", "Tuesday");
h.Add("we", "Wednesday");
IDictionaryEnumerator e= h. GetEnumerator( );
    while(e.MoveNext( ))
    {
        Console.WriteLine(e.Key + "\t" + e.Value);
    }
```

Generics

Are classes, structures, interfaces and methods that have placeholders (type parameters) for one or more of the types they store or use.

```
class Hashtable<K,V>
{
    public Hashtable();
    public object Get(K);
    public object Add(K, V);
}

Hashtable <string,int> addressBook;
..
addressBook.Add("Amit Bhagat", 44235);
..
int extension = addressBook.Get("Shiv Khera");
```


List<T> class

Represents a strongly typed list of objects that can be accessed by index.

Generic equivalent of ArrayList class.

```
List<string> months = new List <string> ();
months.Add("January");
months.Add("February");
months.Add("April");
months.Add("May");
months.Add("June");
foreach(string mon in months)

Console.WriteLine(mon);

Months.Insert(2,"March");
```

List of user defined objects

```
class Employee
{
    int eid;//appropriate constructor and properties for Employee Entity
    string ename;
}
class EmpComparer:IComparer<Employee>
{
    public int Compare(Employee e1, Employee e2)
    { int ret = e1.Name.Length.CompareTo(e2.Name.Length); return ret;
    }
}

public static void Main ()
{
List<Employee>list1 = new List<Employee>();
List1.Add(new Employee(1, "Raghu");
List1.Add(new Employee(2, "Seeta");
List1.Add(new Employee(4, "Leela");
EmpComparer ec = new EmpComparer();
List1.Sort(ec);
foreach(Employee e in list1)
Console.WriteLine(e.Id + "-----"+ e.Name);
}
```

Stack<T> class

```
Stack<int>numStack = new Stack<int>();

numStack.Push(23);
numStack.Push(34);
numStack.Push(65);

Console.WriteLine("Element removed: ", numStack.Pop());
```

Queue<T> class

```
Queue<string> q = new Queue<string>();
q.Enqueue("Message1");
q.Enqueue("Message2");
q.Enqueue("Message3");

Console.WriteLine("First message: {0}", q.Dequeue());
Console.WriteLine("The element at the head is {0}", q.Peek());
IEnumerator<string> e = q.GetEnumerator();
while(e.MoveNext())
Console.WriteLine(e.Current);
```

LinkedList<T> class

Represents a doubly linked List

Each node is on the type LinkedListNode

```
LinkedList<string> l1= new LinkedList<string>();
l1.AddFirst(new LinkedListNode<string>("Apple"));
l1.AddFirst(new LinkedListNode<string>("Papaya"));
l1.AddFirst(new LinkedListNode<string>("Orange"));
l1.AddFirst(new LinkedListNode<string>("Banana"));

LinkedListNode<string> node=l1.First;
Console.WriteLine(node.Value);
Console.WriteLine(node.Next.Value);
```

Dictionary<K, V> class

Represents a collection of keys and values.

Keys cannot be duplicate.

```
Dictionary<int, string> phones= new Dictionary<int, string>();  
phones.Add(1, "James");  
phones.Add(35, "Rita");  
phones.Add(16, "Meenal");  
phones.Add(41, "jim");  
  
phones[16] = "Aishwarya";  
  
Console.WriteLine("Name {0}", phones [12]);  
if (!phone.ContainsKey(4))  
phones.Add(4,"Tim");  
Console.WriteLine("Name is {0}", phones [4]);
```

Custom Generic Types

Generic function

```
static void Main(string[] args)
{ // Swap 2 ints.
    int a = 10, b = 90;
    Console.WriteLine("Before swap: {0}, {1}", a, b);
    Swap<int>(ref a, ref b);
    Console.WriteLine("After swap: {0}, {1}", a, b);
    Console.WriteLine();

    // Swap 2 strings.
    string s1 = "Hello", s2 = "There";
    Console.WriteLine("Before swap: {0} {1}!", s1, s2);
    Swap<string>(ref s1, ref s2);
    Console.WriteLine("After swap: {0} {1}!", s1, s2);
    Console.WriteLine();

    // Compiler will infer System.Boolean.
    bool b1=true, b2=false;
    Console.WriteLine("Before swap: {0}, {1}", b1, b2);
    Swap (ref b1, ref b2);
    Console.WriteLine("After swap: {0}, {1}", b1, b2);
    Console.WriteLine();

    // Must supply type parameter if the method does not take params.
    DisplayBaseClass<int>();
    DisplayBaseClass<string>();
}

static void Swap<T>(ref T a, ref T b)
{
    Console.WriteLine("You sent the Swap() method a {0}", typeof(T));
    T temp; temp = a; a = b; b = temp;
}

static void DisplayBaseClass<T>()
{
    Console.WriteLine("Base class of {0} is: {1}.",
        typeof(T), typeof(T).BaseType);
}
```

Custom Structure

```
// A generic Point structure.
public struct Point<T>
{
    // Generic state data.
    private T xPos;
    private T yPos;

    // Generic constructor.
    public Point(T xVal, T yVal)
    {
        xPos = xVal; yPos = yVal;
    }

    // Generic properties.
    public T X
    {
        get { return xPos; } set { xPos = value; }
    }
    public T Y
    {
        get { return yPos; } set { yPos = value; }
    }
    public override string ToString()
    {
        return string.Format("[{0}, {1}]", xPos, yPos);
    }
    // Reset fields to the default value of the type parameter.
    public void ResetPoint()
    {
        xPos = default(T); yPos = default(T);
    }
}
```

```
static void Main(string[] args)
{
    // Point using ints.
    Point<int> p = new Point<int>(10, 10);
    Console.WriteLine("p.ToString()={0}", p.ToString());
    p.ResetPoint();
    Console.WriteLine("p.ToString()={0}", p.ToString());

    // Point using double.
    Point<double> p2 = new Point<double>(5.4, 3.3);
    Console.WriteLine("p2.ToString()={0}", p2.ToString());
    p2.ResetPoint();
    Console.WriteLine("p2.ToString()={0}", p2.ToString());
}
```

Custom Generic collection class

```
public class Car
{
    public string PetName;
    public int Speed;
    public Car(string name, int currentSpeed)
    {
        PetName = name;
        Speed = currentSpeed;
    }
    public Car() { }
}

public class SportsCar : Car
{
    public SportsCar(string p, int s): base(p, s) { }
    // Assume additional SportsCar methods.
}

public class MiniVan : Car
{
    public MiniVan(string p, int s) : base(p, s) { }
    // Assume additional MiniVan methods.
}

// Custom Generic Collection
public class CarCollection<T> : IEnumerable<T> where T : Car
{
    private List<T> arCars = new List<T>();
    public T GetCar(int pos) { return arCars[pos]; }
    public void AddCar(T c) { arCars.Add(c); }
    public void ClearCars() { arCars.Clear(); }
    public int Count { get { return arCars.Count; }
}
```

```
// IEnumerable<T> extends IEnumerable,
//therefore we need to implement both versions of GetEnumerator().
    IEnumerator<T> IEnumerable<T>.GetEnumerator()
    {return arCars.GetEnumerator(); }
    IEnumerator IEnumerable.GetEnumerator()
    { return arCars.GetEnumerator(); }
// This function will only work because of our applied constraint.
public void PrintPetName(int pos)
{    Console.WriteLine(arCars[pos].PetName);    }
}

static void Main(string[] args)
{ // Make a collection of Cars.
    CarCollection<Car> myCars = new CarCollection<Car>();
    myCars.AddCar(new Car("Alto", 20));
    myCars.AddCar(new Car("i20", 90));
    foreach (Car c in myCars)
    { Console.WriteLine("PetName: {0}, Speed: {1}",
                        c.PetName, c.Speed);
    }
// CarCollection<Car> can hold any type deriving from Car.
    CarCollection<Car> myAutos = new CarCollection<Car>();
    myAutos.AddCar(new MiniVan("Family Truckster", 55));
    myAutos.AddCar(new SportsCar("Crusher", 40));
    foreach (Car c in myAutos)
    { Console.WriteLine("Type: {0}, PetName: {1}, Speed: {2}",
                        c.GetType().Name, c.PetName, c.Speed);
    }
}
```


Exceptions Handling

Abnormalities that occur during the execution of a program (runtime error).

.NET framework terminates the program execution for runtime error.

e.g. divide by Zero, Stack overflow, File reading error, loss of network connectivity

Mechanism to detect and handle runtime error.

```
int a, b=0;

Console.WriteLine("My program starts");

try
{
    a= 10/b;
}
catch(Exception e)
{
    Console.WriteLine(e.Message) ;
}

Console.WriteLine("Remaining program");
```

.NET Exception classes

SystemException	FormatException
ArgymentException	IndexOutOfRangeException
ArgumentNullException	InvalidCastException
ArrayTypeMismatchException	InvalidOperationException
CoreException	NullReferenceException
DivideByZeroException	OutOfMemoryException
StackOverflowException	

User Defined Exception classes

Application specific class can be created using ApplicationException class.

```
class StackFullException:ApplicationException
{
    public string message;
    public StackFullException(string msg)
    {
        Message = msg;
    }
}

public static void Main(string [] args)
{
    StackClass stack1= new StackClass();

    try
    {
        stack1.Push(54);
        stack1.Push(24);
        stack1.Push(53);
        stack1.Push(89);
    }

    catch(StackFullException s)
    {
        Console.WriteLine(s.Message);
    }
}
```

Attributes

Declarative tags that convey information to runtime.

Stored with the metadata of the Element

.NET Framework provides predefined Attributes

The Runtime contains code to examine values of attributes and to act on them

Types of Attributes

Standard Attributes

Custom Attributes

Standard Attributes

.NET framework provides many pre-defined attributes.

- General Attributes

- COM Interoperability Attributes

- Transaction Handling Attributes

- Visual designer component- building attributes

```
[Serializable]
public class Employee
{
    [NonSerialized]
    public string name;
}
```

Custom Attributes

User defined class which can be applied as an attribute on any .NET compatibility Types like:

- Class

- Constructor

- Delegate

- Enum

- Event

- Field

- Interface

- Method

- Parameter

- Property

- Return Value

- Structure

Attribute Usage

AttributeUsageAttribute step 1

It defines some of the key characteristics of custom attribute class with regards to its application , inheritance, allowing multiple instance creation, etc.

```
[AttributeUsage(AttributeTargets.All, Inherited= false,
AllowMultiple=true) ]
```

AttributeUsageAttribute step 2

Designing Attribute class

Attribute classes must be declared as public classes.
All Attribute classes must inherit directly or indirectly from **System.Attribute** .

```
[AttributeUsage(AttributeTargets.All, Inherited= false,
AllowMultiple=true) ]

public class MyAttribute: System.Attribute
{
    ...
}
```

AttributeUsageAttribute step 3

Defining Members

Attributes are initialized with constructors in the same way as traditional classes.

```
public MyAttribute (bool myValue)
{
    this.myValue = myValue;
}
```

Attribute properties should be declared as public entities with description of the data type that will be returned.

```
public bool MyProperty
{
    get { return this. MyValue ;}
    set {this. MyValue= value ;}
}
```

Applying Custom Attribute

Custom attribute is applied in following way.

Retrieving Custom Attributes

```
[Developer("Ravi Tambade", "1")]
public class Employee
{
}
public static void Main()
{
    Employee tflemp = new Employee();
    Type t = tflemp.GetType();
    foreach(Attribute a in t.GetCustomAttributes(true))
    {
        Developer r= (Developer) a;
        //Access r.Name and r.Level
    }
}
```

Delegates

A delegate is a reference to a method.

All delegates inherit from the `System.delegate` type

It is foundation for Event Handling.

Delegate Types

Unicast (Single cast)

Multicast Delegate

Unicast (Single cast) Delegate

Steps in using delegates

- i. Define delegate
- ii. Create instance of delegate
- iii. Invoke delegate

```
delegate string strDelegate(string str);  
  
strDelegate strDel =new strDelegate(strObject.ReverseStr);  
  
string str=strDel("Hello Transflower");  
  
// or use this Syntax  
  
string str =strDel.Invoke("Hello Transflower");
```

Multicast Delegate

A Multicast delegate derives from `System.MulticastDelegate` class.

It provides synchronous way of invoking the methods in the invocation list.

Generally multicast delegate has void as their return type.

```
delegate void strDelegate(string str);  
  
strDelegate delegateObj;  
  
strDelegate Upperobj = new strDelegate(obj.UppercaseStr);  
strDelegate Lowerobj = new strDelegate(obj.LowercaseStr);  
  
delegateObj=Upperobj;  
delegateObj+=Lowerobj;  
  
delegateObj("Welcome to Transflower");
```

Delegate chaining

Instances of delegate can be combined together to form a chain

Methods used to manage the delegate chain are

- Combine method
- Remove method

```
//calculator is a class with two methods Add and Subtract
Calculator obj1 = new Calculator();
CalDelegate [] delegates = new CalDelegate[];
    { new CalDelegate(obj1.Add) ,
      new CalDelegate(Calculator. Subtract)};
CalDelegate chain = (CalDelegate)delegate.Combine(delegates);
Chain = (CalDelegate)delegate.Remove(chain, delegates [0]);
```

Asynchronous Delegate

It is used to invoke methods that might take long time to complete.

Asynchronous mechanism more based on events rather than on delegates.

```
delegate void strDelegate(string str);
public class Handler
{
    public static string UpperCase(string s) {return s. ToUpper() ;}
}
strDelegate caller = new strDelegate(handler. UpperCase);
IAsyncResult result = caller.BeginInvoke("transflower", null, null);
// . . .
String returnValue = caller.EndInvoke(result);
```

Anonymous Method

It is called as inline Delegate.

It is a block of code that is used as the parameter for the delegate.

```
delegate string strDelegate(string str);  
public static void Main()  
{  
    strDelegate upperStr = delegate(string s) {return s.ToUpper() ;};  
}
```

Events

- An Event is an automatic notification that some action has occurred.
- An Event is built upon a Delegate

```
public delegate void AccountOperation();  
public class Account  
{  
    private int balance;  
    public event AccountOperation UnderBalance;  
    public event AccountOperation OverBalance;  
    public Account() {balance = 5000 ;}  
    public Account(int amount) {balance = amount ;}  
    public void Deposit(int amount)  
    {  
        balance = balance + amount;  
        if (balance > 100000) { OverBalance(); }  
    }  
    public void Withdraw(int amount)  
    {  
        balance=balance-amount;  
        if(balance < 5000) { UnderBalance () ;}  
    }  
}}
```


Event Registrations using Event Handlers

```
class Program
{
    static void Main(string [] args)
    {
        Account axisBanktflAccount = new Account(15000);
        //register Event Handlers
        axisBanktflAccount.UnderBalance+=PayPenalty;
        axisBanktflAccount.UnderBalance+=BlockBankAccount;
        axisBanktflAccount.OverBalance+=PayProfessionalTax;
        axisBanktflAccount.OverBalance+= PayIncomeTax;
        //Perform Banking Operations
        axisBanktflAccount.Withdraw(15000);
        Console.ReadLine();
    }
    //Event handlers
    static void PayPenalty()
    {
        Console.WriteLine("Pay Penalty of 500 within 15 days");
    }
    static void BlockBankAccount()
    {
        Console.WriteLine("Your Bank Account has been blocked");
    }
    static void PayProfessionalTax()
    {
        Console.WriteLine("You are requested to Pay Professional Tax");
    }
    static void PayIncomeTax()
    {
        Console.WriteLine("You are requested to Pay Income Tax as TDS");
    }
}
```

Windows Forms

Winforms Object Model (API) is used to build Rich Client traditional Desktop applications.

Microsoft has supplied Winforms API under the **System.Windows.Forms** namespace in assembly System.Windows.Forms.dll.

Event Driven Applications

- Individual user actions are translated into "events".
- Events are passed one by one to application for processing.

GUI based Events

- Mouse Move, Mouse Click, Mouse double Click, Key press, Button click, Menu Selection Change in Focus, Window Activation
- Events are handled by methods that live behind visual interface

Developer job is to program these events.

GUI (Graphics User Interface) applications are based on the notion of forms and controls.

- A form represents a window
- A form contains zero or more controls

First WinForm

```
using System.Windows.Forms ;

public class Form1 : Form

{ public static void Main()

    { //Run the Application

        Application.Run(new Form1());

    }

}
```

GDI +

GDI+ resides in **System.Drawing.dll** assembly.

All GDI+ classes are reside in the System.Drawing, System.Text, System.Printing, System.Internal, System.Imaging, System.Drawing2D and System.Design namespaces.

The Graphics Class

The Graphics class encapsulates GDI+ drawing surfaces. Before drawing any object (for example circle, or rectangle) we have to create a surface using Graphics class. Generally, we use Paint event of a Form to get the reference of the graphics. Another way is to override OnPaint method.

```
private void form1_Paint(object sender, PaintEventArgs e)
{
    Graphics g = e.Graphics;
}
```

Graphics class's methods:

DrawArc	Draws an arc from the specified ellipse.
DrawBezier	Draws a cubic bezier curve.
DrawBeziers	Draws a series of cubic Bezier curves.
DrawClosedCurve	Draws a closed curve defined by an array of points.
DrawCurve	Draws a curve defined by an array of points.
DrawEllipse	Draws an ellipse.
DrawImage	Draws an image.
DrawLine	Draws a line.
DrawPath	Draws the lines and curves defined by a GraphicsPath.
DrawPie	Draws the outline of a pie section.
DrawPolygon	Draws the outline of a polygon.
DrawRectangle	Draws the outline of a rectangle.
DrawString	Draws a string.
FillEllipse	Fills the interior of an ellipse defined by a bounding rectangle.
FillPath	Fills the interior of a path.
FillPie	Fills the interior of a pie section.
FillPolygon	Fills the interior of a polygon defined by an array of points.
FillRectangle	Fills the interior of a rectangle with a Brush.
FillRectangles	Fills the interiors of a series of rectangles with a Brush.
FillRegion	Fills the interior of a Region.

GDI Objects

After creating a Graphics object, you can use it draw lines, fill shapes, draw text and so on. The major objects are:

Brush	Used to fill enclosed surfaces with patterns, colors, or bitmaps.
Pen	Used to draw lines and polygons, including rectangles, arcs, and pies
Font	Used to describe the font to be used to render text
Color	Used to describe the color used to render a particular object. In GDI+ color can be alpha blended

```
Pen pn = new Pen( Color.Blue ); or Pen pn = new Pen( Color.Blue, 100 );
```

The Font Class

The Font class defines a particular format for text such as font type, size, and style attributes. You use font constructor to create a font.

Initializes a new instance of the Font class with the specified attributes.

```
public Font(string, float);
```

Initializes a new instance of the Font class from the specified existing Font and FontStyle.

```
Font font = new Font("Times New Roman", 26);
```

The Brush Class

The Brush class is an abstract base class and cannot be instantiated. We always use its derived classes to instantiate a brush object, such as SolidBrush, TextureBrush, RectangleGradientBrush, and LinearGradientBrush.

```
LinearGradientBrush lBrush = new LinearGradientBrush(rect, Color.Red,
Color.Yellow, LinearGradientMode.BackwardDiagonal);
```

OR

```
Brush brsh = new SolidBrush(Color.Red), 40, 40, 140, 140);
```

The SolidBrush class defines a brush made up of a single color. Brushes are used to fill graphics shapes such as rectangles, ellipses, pies, polygons, and paths.

The TextureBrush encapsulates a Brush that used to fill the interior of a shape with an image.

The LinearGradientBrush encapsulates both two-color gradients and custom multi-color gradients.

The Rectangle Structure

```
public Rectangle(Point, Size); or public Rectangle(int, int, int, int);
```

The Point Structure

```
Point pt1 = new Point( 30, 30);
```

Drawing a rectangle

```
protected override void OnPaint(PaintEventArgs pe)
{
    Graphics g = pe.Graphics ;
    Rectangle rect = new Rectangle(50, 30, 100, 100);
    LinearGradientBrush lBrush = new LinearGradientBrush(rect, Color.Red,
    Color.Yellow,LinearGradientMode.BackwardDiagonal);
    g.FillRectangle(lBrush, rect);
}
```

Drawing an Arc

```
Rectangle rect = new Rectangle(50, 50, 200, 100);
protected override void OnPaint(PaintEventArgs pe)
{
    Graphics g = pe.Graphics ;
    Pen pn = new Pen( Color.Blue );
    Rectangle rect = new Rectangle(50, 50, 200, 100);
    g.DrawArc( pn, rect, 12, 84 );
}
```

Drawing a Line

```
protected override void OnPaint(PaintEventArgs pe)
{
    Graphics g = pe.Graphics ;
    Pen pn = new Pen( Color.Blue );
    Point pt1 = new Point( 30, 30);
    Point pt2 = new Point( 110, 100);
    g.DrawLine( pn, pt1, pt2 );
}
```

Drawing an Ellipse

```
protected override void OnPaint(PaintEventArgs pe)
{
    Graphics g = pe.Graphics ;
    Pen pn = new Pen( Color.Blue, 100 );
    Rectangle rect = new Rectangle(50, 50, 200, 100);
    g.DrawEllipse( pn, rect );
}
```

The FillPath

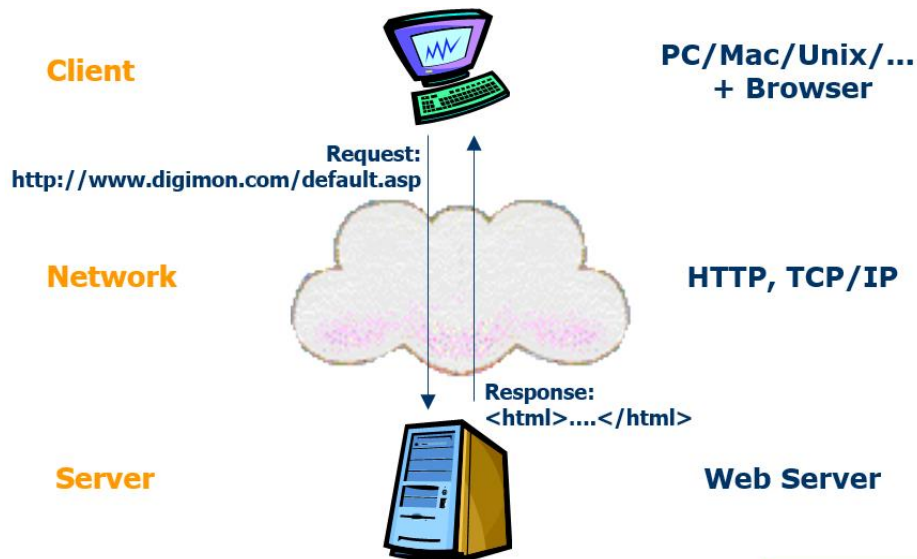
```
protected override void OnPaint(PaintEventArgs pe)
{
    Graphics g = pe.Graphics;
    g.FillRectangle(new SolidBrush(Color.White), ClientRectangle);
    GraphicsPath path = new GraphicsPath(new Point[] {
        new Point(40, 140), new Point(275, 200),
        new Point(105, 225), new Point(190, 300),
        new Point(50, 350), new Point(20, 180), },
        new byte[] {
            (byte)PathPointType.Start,
            (byte)PathPointType.Bezier,
            (byte)PathPointType.Bezier,
            (byte)PathPointType.Bezier,
            (byte)PathPointType.Line,
            (byte)PathPointType.Line,
        });
    PathGradientBrush pgb = new PathGradientBrush(path);
    pgb.SurroundColors = new Color[] { Color.Green,Color.Yellow,Color.Red,
        Color.Blue,
        Color.Orange, Color.White, };
    g.FillPath(pgb, path);
}
```

Drawing Text and Strings

```
protected override void OnPaint(PaintEventArgs pe)
{
    Font fnt = new Font("Verdana", 16);
    Graphics g = pe.Graphics;
    g.DrawString("GDI+ World", fnt, new SolidBrush(Color.Red), 14,10);
}
```

ASP.NET Web Forms

Background Web Architecture

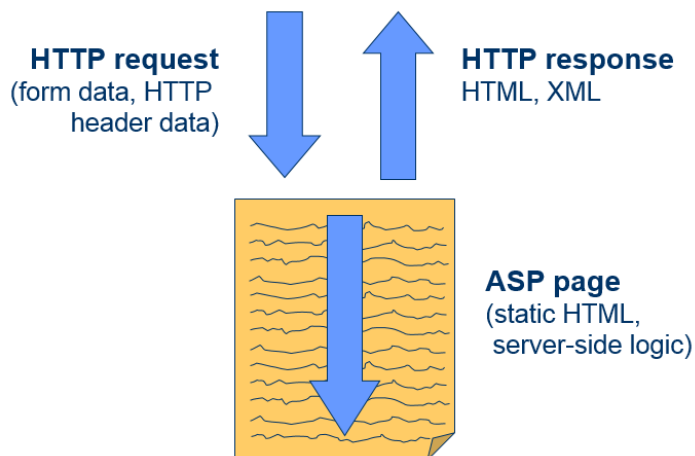


Background Web Technologies

Client-side technologies
HTML, DHTML, JavaScript
Server-side technologies
ASP (Active Server Pages)
ASP.NET is the next generation of ASP

Background what is ASP?

Server-side programming technology
Consists of static HTML interspersed with script
ASP intrinsic objects (Request, Response, Server, Application, and Session) provide services
Commonly uses ADO to interact with databases
Application and session variables
Application and session begin/end events
ASP manages threads, database connections...



Hello World.asp

```
<html>
<head><title>HelloWorld.asp</title></head>
<body>
<form method="post">
<input type="submit" id=button1 name=button1 value="Push Me" />
<%
  if (Request.Form("button1") <> "") then
Response.Write "<p>Hello, the time is " & Now()
end if
%>
</form>
</body>
</html>
```

ASP Success

- Simple procedural programming model
- Access to COM components
- ActiveX Data Objects (ADO)
- File System Object
- Custom components
- Script-based: no compiling, just edit, save & run
- VBScript, JScript – leverages existing skills
- Support for multiple scripting languages
- ASP has been very popular

Background ASP Challenges

- Coding overhead (too much code)
 - Everything requires writing code!
- Code readability (too complex; code and UI intermingled)
- Maintaining page state requires more code
- Reuse is difficult
- Supporting many types of browsers is difficult
- Deployment issues (e.g. DLL locking)
- Session state scalability and availability
- Limited support for caching, tracing, debugging, etc.
- Performance and safety limitations of script.

ASP.NET Overview

- ASP.NET provides services to allow the creation, deployment, and execution of Web Applications and Web Services
- Like ASP, ASP.NET is a server-side technology
- Web Applications are built using Web Forms
- Web Forms are designed to make building web-based applications as easy as building Visual Basic applications

ASP.NET Overview Goals

Keep the good parts of ASP and improve the rest

Simplify: less code, easier to create and maintain

Multiple, compiled languages

Fast

Scalable

Manageable

Available

Customizable and extensible

Secure

Tool support

ASP.NET Overview Key Features

Web Forms

Web Services

Built on .NET Framework

Simple programming model

Maintains page state

Multi-browser support

XCOPY deployment

XML configuration

Complete object model

Session management

Caching

Debugging

Extensibility

Separation of code and UI

Security

ASPX, ASP side by side

Simplified form validation

Cookie less sessions

ASP.NET Overview Demo: Hello World.aspx

```

<%@ Page language="c#" %>
<html>
  <head></head>
  <script runat="server">
    public void B_Click (object sender, System EventArgs e) {
      Label1.Text = "Hello, the time is " + DateTime.Now;
    }
  </script>
  <body>
    <form method="post" runat="server">
      <asp:Button onclick="B_Click" Text="Push Me"
        runat="server" /> <p>
      <asp:Label id=Label1 runat="server" />
    </form>
  </body>
</html>

```

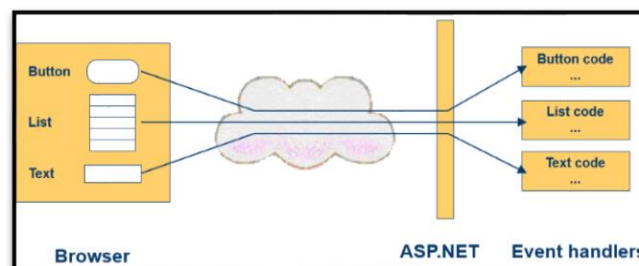
ASP.NET Overview Architecture

ASP.NET is built upon

- .NET Framework
- Internet Information Server (IIS)
- IIS MMC Snap-In (Internet Services Manager)
 - Tool to manage IIS
- Virtual Directories
 - Provides a mapping between URL and file path
 - E.g., on my machine the URL:
 - <http://localhost/CS594>
 - maps to the file path:
 - C:_CS594Fall2001

Programming Model Controls and Events

- Server-side programming model
- Based on controls and events
- Just like Visual Basic
- Not "data in, HTML out"
- Higher level of abstraction than ASP
- Requires less code
- More modular, readable, and maintainable



Programming Model ASP.NET object Model

User code executes on the web server in page or control event handlers

Controls are objects, available in server-side code

Derived from System.Web.UI.Control

The web page is an object too

Derived from System.Web.UI.Page which is a descendant of System.Web.UI.Control

A page can have methods, properties, etc.

Programming Model Postbacks

A postback occurs when a page generates an

HTML form whose values are posted back to the same page

A common technique for handling form data

In ASP and other server-side technologies the state of the page is lost upon postback...

Unless you explicitly write code to maintain state

This is tedious, bulky and error-prone.

Programming Model Postbacks maintain states

By default, ASP.NET maintains the state of all server-side controls during a postback

Can use method="post" or method="get"

Server-side control objects are automatically populated during postback

No state stored on server

Works with all browsers

Programming Model Server-Side Controls

Multiple sources of controls

Built-in

3rd party

User-defined

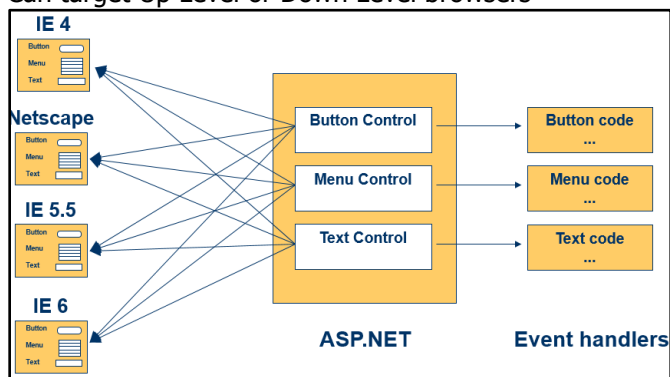
Controls range in complexity and power: button, text, drop down, calendar, data grid, ad rotator, validation

Can be populated via data binding.

Programming Model Automatic Browser Compatibility

Controls can provide automatic browser compatibility

Can target Up Level or Down Level browsers



Programming Model Code behind pages

Two styles of creating ASP.NET pages

Controls and code in .aspx file

Controls in .aspx file, code in code-behind page

Supported in Visual Studio.NET

Code-behind pages allow you to separate the user interface design from the code

Allows programmers and designers to work independently.

```
<%@ Codebehind="WebForm1.cs" Inherits=WebApplication1.WebForm1" %>
```

Programming Model Automatic Compilation

Just edit the code and hit the page

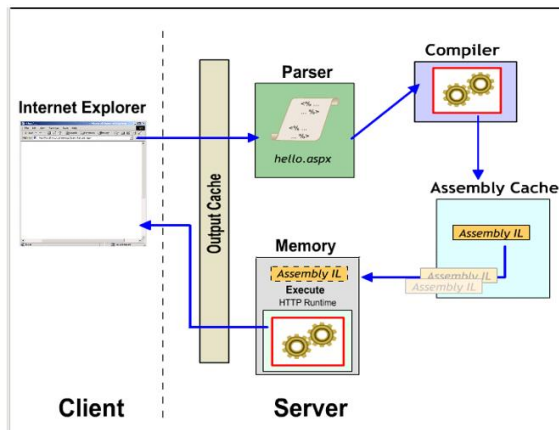
ASP.NET will automatically compile the code into an assembly

Compiled code is cached in the CLR Assembly Cache

Subsequent page hits use compiled assembly

If the text of the page changes then the code is recompiled

Works just like ASP: edit, save and run



Programming Basics Page Syntax

The most basic page is just static text

Any HTML page can be renamed .aspx

Pages may contain:

Directives: `<%@ Page Language="C#" %>`

Server controls: `<asp:Button runat="server">`

Code blocks: `<script runat="server">...</script>`

Data bind expressions: `<%# %>`

Server side comments: `<%-- --%>`

Render code: `<%= %>` and `<% %>`

Use is discouraged; use `<script runat=server>` with code in event handlers

Instead

Programming Basics Page Directive

Lets you specify page-specific attributes, e.g.

AspCompat: Compatibility with ASP

Buffer: Controls page output buffering

CodePage: Code page for this .aspx page

ContentType: MIME type of the response

ErrorPage: URL if unhandled error occurs

Inherits: Base class of Page object

Language: Programming language

Trace: Enables tracing for this page

Transaction: COM+ transaction setting

We do have only one page directive for each asp.net Webform (.aspx)

Programming Basics Server Control Syntax

Controls are declared as HTML tags with runat="server" attribute

```
<input type=text id=text2 runat="server" />  
<asp:calendar id=myCal runat="server" />
```

Tag identifies which type of control to create

Control is implemented as an ASP.NET class

The id attribute provides programmatic identifier

It names the instance available during postback

Just like Dynamic HTML

Programming Basics Server Control Properties

Tag attributes map to control properties

```
<asp:button id="c1" Text="Foo" runat="server">  
<asp:ListBox id="c2" Rows="5" runat="server">
```

Tags and attributes are case-insensitive

Control properties can be set programmatically

```
c1.Text = "Foo";  
c2.Rows = 5;
```

Programming Basics Maintaining State

By default, controls maintain their state across multiple postback requests

Implemented using a hidden HTML field: __VIEWSTATE

Works for controls with input data (e.g. TextBox, CheckBox), non-input controls (e.g. Label, DataGrid), and hybrids (e.g. DropDownList, ListBox)

Can be disabled per control or entire page

Set EnableViewState="false"

Lets you minimize size of __VIEWSTATE

Programming Basics Server Code Blocks

Server code lives in a script block marked **runat="server"**

```
<script language="C#" runat=server>
<script language="VB" runat=server>
<script language="JScript" runat=server>
```

Script blocks can contain

Variables, methods, event handlers, properties

They become members of a custom Page object

Programming Basics Page Events

Pages are structured using events

Enables clean code organization

Avoids the "Monster IF" statement

Less complex than ASP pages

Code can respond to page events

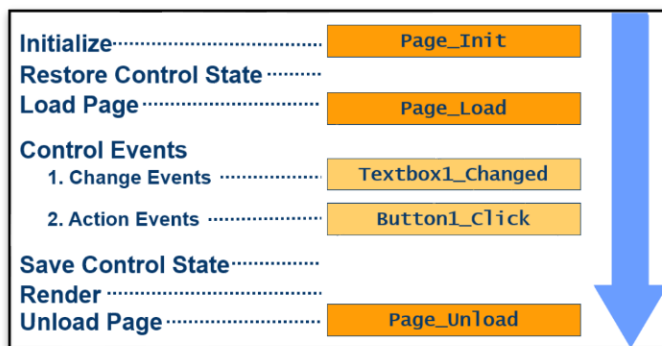
e.g. Page_Load, Page_Unload

Code can respond to control events

Button1_Click

Textbox1_Changed

Programming Basics Page Event Lifecycle



Programming Basics Page Loading

Page_Load fires at beginning of request after controls are initialized
Input control values already populated

```
protected void Page_Load(Object s, EventArgs e)
{
    message.Text = textbox1.Text;
}
```

Page Load fires on every request
Use Page.IsPostBack to execute conditional logic
If a Page/Control is maintaining state then need only initialize it when IsPostBack is false.

```
protected void Page_Load(Object s, EventArgs e)
{
    if (! Page.IsPostBack)
    {
        // Executes only on initial page load
        Message.Text = "initial value";
    }
    // Rest of procedure executes on every request
}
```

Programming Basics Server Control Event

Change Events

By default, these execute only on next action event
E.g. OnTextChanged, OnCheckedChanged
Change events fire in random order

Action Events

Cause an immediate postback to server
E.g. OnClick

Works with any browser

No client script required, no applets, no ActiveX® Controls!

Programming Basics Wiring up Control Events

Control event handlers are identified on the tag

```
<asp: button onclick="btn1_click"runat=server>
<asp: textbox onchange="text1_changed"runat=server>
```

Event handler code

```
protected void btn1_Click(Object s, EventArgs e)
{
    Message.Text = "Button1 clicked";
}
```


Programming Basics Event Arguments

Events pass two arguments:

- The sender, declared as type object
 - Usually the object representing the control that generated the event
 - Allows you to use the same event handler for multiple controls
- Arguments, declared as type EventArgs
 - Provides additional data specific to the event
 - EventArgs itself contains no data; a class derived from EventArgs will be passed

Programming Basics Page Unloading

Page_Unload fires after the page is rendered

Don't try to add to output

Useful for logging and clean up

```
protected void Page_Unload(Object s, EventArgs e)
{
    MyApp.LogPageComplete();
}
```

Programming Basics Import Directive

Adds code namespace reference to page

Avoids having to fully qualify .NET types and class names

Equivalent to the C# using directive.

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Net" %>
<%@ Import Namespace="System.IO" %>
```

Programming Basics Page Class

The Page object is always available when handling server-side events

Provides a large set of useful properties and methods, including:

Application, Cache, Controls, EnableViewState,
EnableViewStateMac, ErrorPage, IsPostBack, IsValid,
Request, Response, Server, Session,
Trace, User, Validators

DataBind (), LoadControl (), MapPath (), Validate ()

ASP.NET Server Controls

ASP.NET webform provides more than 50 built-in controls.

These controls are organized into logical families

- **HTML controls**
 - Controls / properties map 1:1 with HTML
- **Server controls**
 - Richer functionality
 - More consistent object model

HTML Controls

Derived from System.Web.UI.HtmlControls.HtmlControl

Supported controls have custom class, others derive from HtmlGenericControl

Supported Controls

```
<a>
<img>
<form>
<table>
<tr>
<td>
<th>
<select>
<textarea>
<button>
<input type=text>
<input type=file>
<input type=submit>
<input type=button>
<input type=reset>
<input type=hidden>
```

Can use controls two ways:

Handle everything in action events (e.g. button click)

Event code will read the values of other controls (e.g. text, check boxes, radio buttons, select lists)

Handle change events as well as action events

Server Controls HTML Controls Web Controls

```
Label1.BackColor = Color.Red;
Table.BackColor = Color.Blue;

<asp:button onclick="button1_click" runat=server>
<asp:textbox onchange="text1_changed" runat=server>
```

Defined in the System.Web.UI.WebControls namespace

Four types of Webserver Controls

- Intrinsic controls
- List controls
- Rich controls
- Validation controls

Server Controls Intrinsic controls

Correspond to HTML controls

Supported controls

<asp:button>	<asp:imagebutton>	<asp:linkbutton>	<asp:hyperlink>	<asp:textbox>
<asp:checkbox>	<asp:radiobutton>	asp:image>	<asp:label>	<asp:panel>
<asp:table>				

TextBox, ListControl, CheckBox and their subclasses don't automatically do a postback when their controls are changed
Specify AutoPostBack=true to make change events cause a postback

Server Controls List Controls

These Controls handle repetition

<asp:dropdownlist>	<asp:listbox>	<asp:radiobuttonlist>	<asp:checkboxlist>	<asp:repeater>
<asp:datalist>	<asp:datagrid>			

These Server Controls are:

- Powerful, customizable list controls
- Expose templates for customization
- Can contain other controls
- Provide event bubbling through their OnItemCommand event
- More about these controls and templates later

Server Controls CheckBoxList and RadioButtonList

Provides a collection of check box or radio button controls
Can be populated via data binding

```
<asp:CheckBoxList id=Check1 runat="server">
  <asp:ListItem>Big Data</asp:ListItem>
  <asp:ListItem>Artificial Intelligence</asp:ListItem>
  <asp:ListItem>Internet of Things</asp:ListItem>
  <asp:ListItem>Cloud Computing</asp:ListItem>
</asp:CheckBoxList>
```

Server Control Validation Control Features

- Rich, declarative validation
- Validation declared separately from input control
- Extensible validation framework
- Supports validation on client and server
- Automatically detects uplevel clients
- Avoids roundtrips for uplevel clients
- Server-side validation is always done
- Prevents users from spoofing Web Forms

<asp:RequiredFieldValidator>	Ensures that a value is entered
<asp:RangeValidator>	Checks if value is within minimum and maximum values
<asp:CompareValidator>	Compares value against constant, another control or data type
<asp:RegularExpressionValidator>	Tests if value matches a predefined pattern
<asp:CustomValidator>	Lets you create custom client- or server-side validation function
<asp:ValidationSummary>	Displays list of validation errors in one place

Validation controls are derived from System.Web.UI.WebControls.BaseValidator, which is derived from the Label control.
Validation controls contain text which is displayed only if validation fails
Text property is displayed at control location
ErrorMessage is displayed in summary

Validation controls are associated with their target control using the ControlToValidate property

```
<asp:TextBox id=TextBox1 runat=server />
<asp:RequiredFieldValidator id="Req1"
ControlToValidate="TextBox1"
Text="Required Field" runat=server />
```

Can create multiple validation controls with the same target control
Page.IsValid indicates if all validation controls on the page succeed

```
void Submit_click(object s, EventArgs e)
{
  if (Page.IsValid) { Message.Text = "Hi Transflower,Page is valid!"; }
}
```

ADO.NET

A rich set of classes, interface, structures and enumerated types that manage data access from different types of data stores

ADO.NET Features

- A robust connected, disconnected Data Access Model
- Integrated XML support
- Data from varied Data Sources
- Familiarity to ADO programming model (unmanaged environment)
- Enhanced Support

Connected vs. Disconnected Architecture

	Connected	Disconnected
State of Connection	Constantly kept Opened	Closed once data is fetched in cache at client side.
Scalability	Limited	More
Current Data	Always available	Not up to date

ADO.NET components

.NET Data Providers

Allow users to interact with different types of data sources.

ODBC Providers

OleDb Providers

SQL Data Providers

Oracle Data Providers

DataSets

Explicitly designed for disconnected architecture

ADO.NET Interfaces

IDbConnection	Represents an open connection to a data source.
IDbCommand	Represents an SQL statement that is executed while connected to a data source.
IDataReader	Provides a means of reading one or more forward-only streams of result sets obtained by executing a command at a data source.
IDataAdapter	Provides loosely coupled Data Access with Data Sources.

Connection Object

Has the responsibility of establishing connection with the data source.

Connection has to be explicitly closed in finally block.

```
SqlConnection conSql= new SqlConnection();  
conSql.ConnectionString = "server=DatabaseServer; Initial Catalog= Transflower; user  
                           id= sa; password=sa";  
conSql.Open();
```

Command Object

Used to specify the type of interaction to perform with the database like select, insert, update and delete.

Exposes properties like:

CommandText

CommandType

Connection

Exposes several execute methods like

ExecuteScalar()

ExecuteReader()

ExecuteNonQuery()

Inserting Data

```
string insertString = "insert into dept (deptId, deptName, loc) values  
(10,'Mktg', 'Mumbai') ;  
string updateString = "update dept set deptName='Marketing' where  
deptName='Mktg' ;  
string deleteString = "delete from dept where deptName='ABC' " ;  
conSql.Open() ;  
SqlCommand cmd= new SqlCommand(insertString, conSql);  
cmd.ExecuteNonQuery() ;
```

Getting Single Value

```
SqlCommand cmdSql = new SqlCommand();  
cmdSql.Connection = conSql;  
cmdSql.CommandText = "Select Count(*) from emp";  
int count = (int) cmdSql.ExecuteScalar();  
MessageBox.Show(count.ToString());
```

The DataReader Object

Used to only read data in forward only sequential manner.

```
string queryStr = "Select deptName, loc from dept";  
conSql.Open();  
SqlCommand cmdSql = new SqlCommand(queryStr, conSql);  
SqlDataReader dataReader= cmdSql.ExecuteReader();  
while(dataReader.Read())  
{  
    MessageBox.Show("Last Name is "+ dataRead[0].ToString());  
    MessageBox.Show("First Name is "+ dataRead[1].ToString());  
}  
dataReader.Close();
```

Adding parameters to Command

```
conSql.Open();
SqlCommand cmd = new SqlCommand("select * from emp where empNo =@eno", conSql);
SqlParameter param = new SqlParameter();
param.ParameterName = "@eno";
param.Value = 100;
Cmd.Parameters.Add(param);
SqlDataReader reader =cmd.ExecuteReader();
while(reader.Read())
{
    //Display data
}
```

Calling a Stored Procedure

```
//Stored Procedure in SQL Server
CREATE PROCEDURE DeleteEmpRecord (@eno)
AS delete from emp where empno = @eno;
RETURN
//C# Code

SqlCommand cmd = new SqlCommand ("DeleteEmpRecord", conSql);
cmd.CommandType = CommandType.StoredProcedure;
cmd.Parameters.Add( new SqlParameter("@eno",100);
```

Multiple Queries

Multiple queries can be executed using a single command object.

```
SqlCommand cmd = new SqlCommand("select * from dept; select * from emp", conSql);
SqlDataReader reader =cmd.ExecuteReader();
...
//code to access first result set
bool result = dr.NextResult();
...
// code to access next result set
```


Disconnected Data Access

Data Adapter

Represents a set of data commands and a database connection that are used to fill the dataset and update a sql server database.

Forms a bridge between a disconnected ADO.NET objects and a data source.

Supports methods

- Fill();
- Update ();

```
string sqlStr= "SELECT * FROM Orders";  
SqlAdapter da = new SqlAdapter (sqlStr, con);
```

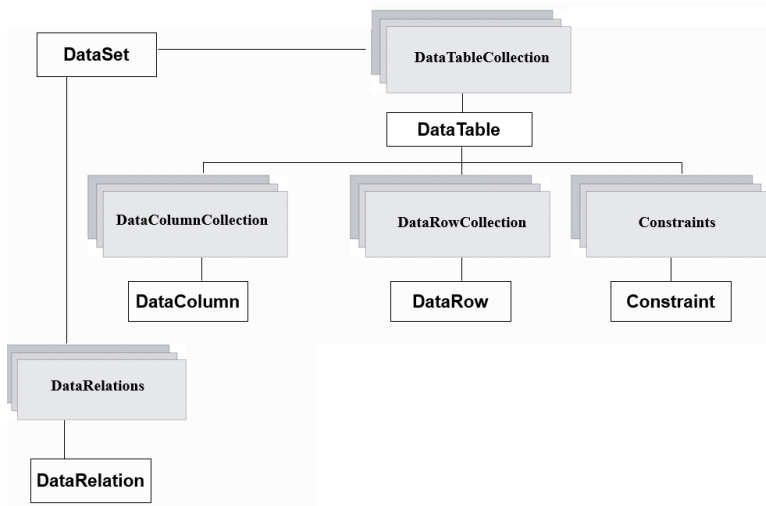
DataAdapter Properties

- **SelectCommand**
- **Insertcommand**
- **DeleteCommand**
- **UpdateCommand**

```
da.SelectCommand.CommandText = "SELECT customerID , ContactName FROM Customers";  
SqlCommand cmd = new SqlCommand ("INSERT into Customers (CustomerID, CompanyName)  
VALUES (898, 'seed');  
da.InsertCommand = command;
```

DataSet

Main object of disconnected architecture
Can store tables with their schema at client side.



Role of CommandBuilder Object

Automatically generates Insert, Update, delete queries by using SelectCommand property of DataAdapter

```

SqlConnection con = new SqlConnection("server=databaseServer; Initial Catalog=
Transflower; userid=sa; password=sa");
SqlDataAdapter da= new SqlDataAdapter("Select * from Customers", con);
SqlCommandBuilder cmdBuilder= new SqlCommandBuilder(da);
DataSet ds= new DataSet();
Da.Fill(ds, "Cusomters");
  
```

Constraints

Constraints restrict the data allowed in a data column or set of data columns.

Constraint classes in the System.Data namespace

UniqueConstraint

ForeignKeyConstraint

Using existing primary Key constraint

```

da.FillSchema(ds, schematype.Source, "Customers");
Or
da.MissingSchemaAction = AddWithKey;
da.Fill(ds,"Customers");
  
```

ADO.NET and XML

With ADO.NET it is easy to

- Convert data into XML
- Generate a matching XSD schema
- Perform an XPath search on a result set.
- Interact with an ordinary XML document through the ADO.net

DataSet XML Methods

- Getxml()
- GetXmlSchema()
- ReadXml()
- ReadXmlSchema()
- WriteXml()
- WriteXmlSchema()
- InferXmlSchema()

Concurrency in Disconnected Architecture

- Disadvantage of disconnected architecture
- Conflict can occur when two or more users retrieve and then try to update data in the same row of a table
- The second user's changes could overwrite the changes made by the first user.

ASP.NET DataBinding

Data Binding what it is?

- Provides a single simple yet powerful way to populate Web Form controls with data
- Enables clean separation of code from UI
- Supports binding to any data source Properties, expressions, method calls, Collections (Array, Hashtable, etc.)
Dataset, DataTable, DataView, DataReader
XML
- One-way snapshot model
- Requires code to reapply to data model
- Allows you to specify an expression

When the DataBind method of the control is called, the expression is evaluated and bound DataBind for a single control (and subcontrols)

Page.DataBind binds all controls on a page

Data Binding Scalar Expressions

Data binding expression: <%# expression %>

Expression is evaluated when DataBind() is called

```
<asp:Label id=label1
Text=<%# "The result is " + (1 + 2) + " , the time is " +
DateTime.Now.ToLongTimeString() %> runat="server" />

public void Page_Load(object s, EventArgs e) {
    if (! Page.IsPostBack)
        Page.DataBind();
}
```

Data Binding Simple List

Data binding a list creates a user interface element for each item in the list
Each item contains text (displayed to user) and an optional value (not displayed)
The simple list controls:

```
<asp:ListBox>
Single or multiple select
<asp:DropDownList>
<asp:RadioButtonList>
<asp:CheckBoxList>
```

Steps to data bind a list control

1. Declare the list control
2. Optionally set DataValueField and DataTextField
3. Set its DataSource
4. Call DataBind() method

Data Binding Database

Data binding can be used to populate server controls with data from a database

Data Binding Data Source Example

```

DataView GetSampleData()
{
    DataSet ds;
    SqlConnection cxn;
    SqlDataAdapter adp;
    cxn = new SqlConnection("server=localhost; " + "uid=sa;pwd=;database=Northwind");
    adp = new SqlDataAdapter("select CategoryID, CategoryName from Categories", cxn);
    ds = new DataSet();
    adp.Fill(ds, "Categories");
    return ds.Tables["Categories"].DefaultView;
}

```

Data Binding List Binding Example

```

void Page_Load(object s, EventArgs e)
{
    ListBox1.DataSource = GetSampleData();
    ListBox1.DataValueField = "CategoryID";
    ListBox1.DataTextField = "CategoryName";
    ListBox1.DataBind();
}

<asp : ListBox id="ListBox1" runat="server" />
void Page_Load(object s, EventArgs e)
{
    ListBox1.DataBind();
}

```

```

<asp:ListBox id="ListBox1" runat="server" DataSource=<%#GetSampleData()%>
DataValueField="CategoryID"
DataTextField="CategoryName" />

```

Data Binding DataGrid

- Full-featured list output
- Default look is a grid
- Default is to show all columns, though you can specify a subset of columns to display
- Columns can be formatted with templates
- Optional paging
- Updateable

Data Binding binding to all Columns

Binding all columns in the datasource

1. Declare an <asp:DataGrid>
2. Set its DataSource
3. Call DataBind()

```
void Page_Load(object s, EventArgs e)
{
    myDataGrid.DataSource = GetSampleData();
    myDataGrid.DataBind();
}

<asp:datagrid id=myDataGrid runat="server" />
```

Data Binding DataGrid Paging

When there is too much data to display in one screen, a DataGrid can provide automatic paging

1. Set AllowPaging="true"
2. Set PageSize=5
3. Handle OnPageIndexChanged event
4. Set page index
 - i. Fetch data
 - ii. Re-bind data

Data Binding Template

- Templates provide a powerful way to customize the display of a server control
- Customize structure – not just style
- Can use controls or other HTML within a template
- 3rd party controls can expose new templates
- With data binding, templates specify a set of markup (HTML or server controls) for each bound piece of data
- Not just specifying formatting and style for a column
- However, templates are not limited to data binding
- No fixed set of templates
- Controls may define their own and expose any number of them

Standard templates for list-bound controls

HeaderTemplate: rendered once before all data bound rows

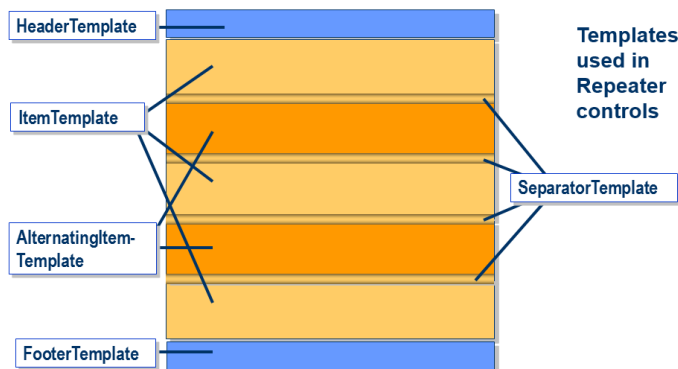
ItemTemplate: rendered once for each row in the data source

AlternatingItemTemplate: like ItemTemplate, but when present is used for every other row

SeparatorTemplate: rendered between each row

FooterTemplate: rendered once, after all data bound rows

Data Binding Template



Data Binding in Template

- Templates need to access the bound data
- Container is an alias for the template's containing control
- DataItem is an alias for the current row of the datasource
- DataBinder.Eval is a utility function provided to retrieve and format data within a template

```
<%# DataBinder.Eval(Container.DataItem, "price", "$ {0}") %>
```

Data Binding Repeater Control

- Provides simple output of a list of items
- No inherent visual form
- Templates provide the visual form
- No paging
- Can provide templates for separators
- Not updateable

```
<asp:Repeater id="repList" runat="server">
  <template name="HeaderTemplate">
    <table>
      <tr><td>Title</td><td>Type</td></tr>
    </table>
  </template>
  <template name="ItemTemplate">
    <tr>
      <td><%# DataBinder.Eval(Container.DataItem,"title_id") %></td>
      <td><%# DataBinder.Eval(Container.DataItem,"type") %></td>
    </tr>
  </template>
  <template name="FooterTemplate">
  </table>
</template>
</asp:Repeater>
```

Data Binding DataList Control

- Provides list output with editing
- Default look is a table
- Customized via templates
- Directional rendering (horizontal or vertical)
- Single and multiple selection
- Alternate item
- Updateable
- No paging

```
void Page_Load(object s, EventArgs e) {  
    myDataGrid.DataSource = GetSampleData();  
    myDataGrid.DataBind();  
}  
<asp:datalist id=myDataList runat=server>  
    <template name="itemtemplate">  
        <b>Title id:</b>  
        <%# DataBinder.Eval(Container.DataItem, "title_id") %>  
        <br> <b>Title:</b>  
        <%# DataBinder.Eval(Container.DataItem, "title") %>  
    </template>  
</asp:datalist>
```


Building Data-Driven ASP.NET Application

Simplified Data Binding

Data binding expressions are now simpler and support hierarchical (XML) data binding

```
<!-- ASP.NET 1.x data binding expression -->
<%# DataBinder.Eval (Container.DataItem, "Price") %>
<!-- Equivalent ASP.NET 2.0 data binding expression -->
<%# Eval ("Price") %>
<!-- XML data binding -->
<%# XPath ("Price") %>
```

Data Source Controls: Declarative (no-code) data binding

Name	Description
SQLDataSource	Connect data-binding controls to SQL database
AccessDataSource	Connect data-binding controls to Access database
XMLDataSource	Connect data-binding controls to XML
ObjectDataSource	Connect data-binding controls to data components
SiteMapDataSource	Connects site navigation controls to site map data

SqlData Source

Declarative data binding to SQL databases

Any database served by a managed provider

Two-way data binding

SelectCommand defines query semantics

InsertCommand, UpdateCommand, and DeleteCommand define update semantics

Optional caching of query results

Parameterized operation

Using SQL Data Source

```
<asp:SqlDataSource ID="Titles" RunAt="server"
  ConnectionString="server=localhost;database=pubs;integrated security=true"
  SelectCommand="select title_id, title, price from titles" />
<asp:DataGrid DataSourceID="Titles" RunAt="server" />
```

Name	Description
ConnectionString	Connection string used to connect to data source
SelectCommand	Command used to perform queries
InsertCommand	Command used to perform inserts
UpdateCommand	Command used to perform updates
DeleteCommand	Command used to perform deletes
DataSourceMode	Specifies whether DataSet or DataReader is used(default=DataSet)
ProviderName	Specifies provider (default=SQL Server .NET provider)

Calling Stored Procedures

```
<asp:SqlDataSource ID="Countries" RunAt="server"
  ConnectionString="server=localhost;database=northwind;..."
  SelectCommand="proc_GetCountries" />
<asp:SqlDataSource ID="Customers" RunAt="server"
  ConnectionString="server=localhost;database=northwind;..."
  SelectCommand="proc_GetCustomers">
  <SelectParameters>
    <asp:ControlParameter Name="Country" ControlID="MyDropDownList"
      PropertyName="SelectedValue" />
  </SelectParameters>
</asp:SqlDataSource>
<asp:DropDownList ID="MyDropDownList" DataSourceID="Countries"
  DataTextField="country" AutoPostBack="true" RunAt="server" />
<asp:DataGrid DataSourceID="Customers" RunAt="server" />
```

Stored Procedure examples

```
CREATE PROCEDURE proc_GetCustomers
@Country nvarchar (32) AS
  SELECT * FROM Customers
  WHERE Country = @Country
GO

CREATE PROCEDURE proc_GetCountries AS
  SELECT DISTINCT Country
  FROM Customers
  ORDER BY Country
GO
```

XMLDataSource

- Declarative data binding to XML data
- Supports caching and XSL transformations
- One-way data binding only; no updating

```
<asp:XmlDataSource ID="Rates" DataFile="Rates.xml" RunAt="server" />
<asp:TreeView ID="MyTreeView" DataSourceID="Rates" RunAt="server" />
```

ObjectDataSource

- Declarative binding to data components
 - Leverage middle-tier data access components
 - Keep data access code separate from UI layer
- Two-way data binding
 - SelectMethod, InsertMethod, UpdateMethod, and DeleteMethod
- Optional caching of query results
- Parameterized operation

Key ODS Properties

Name	Description
TypeName	Type name of data component
SelectMethod	Method called on data component to perform queries
InsertMethod	Method called on data component to perform insert
UpdateMethod	Method called on data component to perform update
DeleteMethod	Method called on data component to perform delete
EnableCaching	Specifies whether caching is enabled(default=false)

Key ODS Properties Count

Name	Description
CacheDomain	Length of time in seconds data should be cached
SqlCacheDependency	Creates dependency on specified database entity
SelectParameter	Specifies parameter for SelectMethod
InsertParameter	Specifies parameter for InsertMethod
UpdateParameter	Specifies parameter for UpdateMethod
DeleteParameter	Specifies parameter for DeleteMethod

Initialization and Clean-Up

ObjectDataSource.SelectMethod et al can identify static methods or instance methods

If instance methods are used:

ODS creates new class instance on each call

Class must have public default constructor

Use ObjectCreated and ObjectDisposing events to perform specialized initialization or clean-up work on data components

The GridView Control

- Enhanced DataGrid control
 - Renders sets of records as HTML tables
- Built-in sorting, paging, selecting, updating, and deleting support
- Supports rich assortment of field types, including CheckBoxFields
 - Declared in <Columns> element
- Highly customizable UI

GridView Example

```
<asp:SqlDataSource ID="Employees" RunAt="server"
  ConnectionString="server=localhost;database=northwind;..."
  SelectCommand="select lastname, firstname, title from employees" />
<asp:GridView DataSourceID="Employees" Width="100%" RunAt="server" />
```

Output

lastname	firstname	title
Davolio	Nancy	Sales Representative
Fuller	Andrew	Vice President, Sales
Leverling	Janet	Sales Representative
Peacock	Margaret	Sales Representative
Buchanan	Steven	Sales Manager
Suyama	Michael	Sales Representative
King	Robert	Sales Representative
Callahan	Laura	Inside Sales Coordinator
Dodsworth	Anne	Sales Representative

GridView Field Type

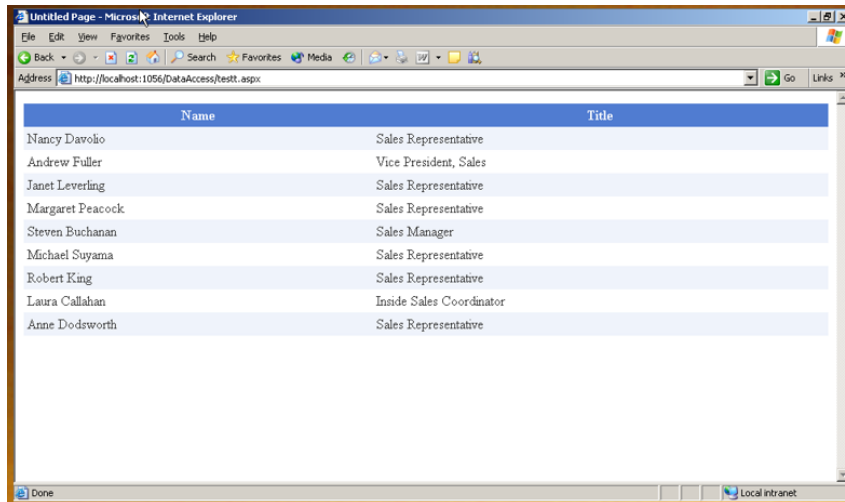
Name	Description
BoundField	Renders columns of text fields in data source
ButtonField	Renders columns of button (push button, image, or link
CheckBoxField	Renders Booleans as check boxes
CommandField	Renders columns of selecting and editing GridView data
HyperLinkField	Renders columns of hyperlinks
ImageField	Renders columns of images from URL text
TemplateField	Renders columns using HTML templates

Specifying Field Type

```
<asp:SqlDataSource ID="Employees" RunAt="server"
    ConnectionString="server=localhost;database=northwind;..."
    SelectCommand="select photo, lastname, firstname, title from employees" />

<asp:GridView DataSourceID="Employees" Width="100%" RunAt="server"
    AutoGenerateColumns="false" >
    <Columns>
        <asp:TemplateField HeaderText="Name">
            <ItemTemplate>
                <%# Eval ("firstname") + " " + Eval ("lastname") %>
            </ItemTemplate>
        </asp:TemplateField>
        <asp:BoundField HeaderText="Title" DataField="title" />
    </Columns>
</asp:GridView>
```

Output



Name	Title
Nancy Davolio	Sales Representative
Andrew Fuller	Vice President, Sales
Janet Leverling	Sales Representative
Margaret Peacock	Sales Representative
Steven Buchanan	Sales Manager
Michael Suyama	Sales Representative
Robert King	Sales Representative
Laura Callahan	Inside Sales Coordinator
Anne Dodsworth	Sales Representative

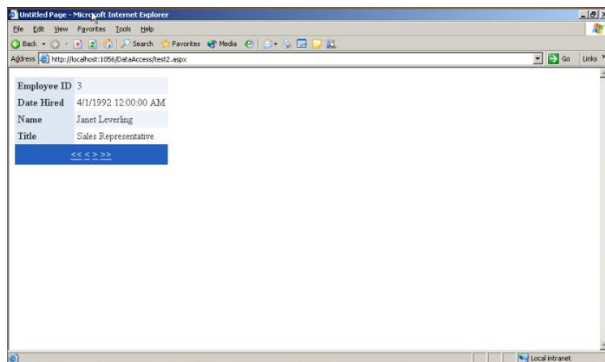
The DetailsView Control

- Renders individual records
 - Pair with GridView for master-detail views
 - Or use without GridView to display individual records
- Built-in paging, inserting, updating, deleting
- Uses same field types as GridView
 - Declared in <Fields> element
- Highly customizable UI

DetailsView Example

```
<asp:SqlDataSource ID="Employees" RunAt="server"
ConnectionString="server=localhost;database=northwind;..."
SelectCommand="select employeeid, photo, ... from employees" />
<asp:DetailsView DataSourceID="Employees" RunAt="server"
AllowPaging="true" AutoGenerateRows="false"
PagerSettings-Mode="NextPreviousFirstLast">
<Fields>
<asp:BoundField HeaderText="Employee ID" DataField="employeeid" />
<asp:BoundField HeaderText="Date Hired" DataField="hiredate" />
<asp:TemplateField HeaderText="Name">
<ItemTemplate>
<%# Eval ("firstname") + " " + Eval ("lastname") %>
</ItemTemplate>
</asp:TemplateField>
<asp:BoundField HeaderText="Title" DataField="title" />
</Fields>
</asp:DetailsView>
```

Output



Inserting Updating and Deleting

- Data controls supply editing UIs
 - AutoGenerateXxxButton properties
 - Insert/EditRowStyle properties
- Data source controls supply editing logic
 - Insert/Update/DeleteCommand properties
 - Insert/Update/DeleteParameters properties
 - Inserting/ed, Updating/ed, Deleting/ed events
 - Visual Studio supplies the glue

Editing with GridView

```
<asp:SqlDataSource ID="Employees" RunAt="server"
    ConnectionString="server=localhost;database=northwind;..."
    SelectCommand="select employeeid, lastname, firstname from employees"
    UpdateCommand="update employees set lastname=@lastname, firstname=
    @firstname where employeeid=@original_employeeid">
    <UpdateParameters>
    <asp:Parameter Name="EmployeeID" Type="Int32" />
    <asp:Parameter Name="lastname" Type="String" />
    <asp:Parameter Name="firstname" Type="String" />
    </UpdateParameters>
</asp:SqlDataSource>
<asp:GridView DataSourceID="Employees" Width="100%" RunAt="server"
    DataKeyNames="EmployeeID" AutoGenerateEditButton="true" />
```

Managing Look, Feel, and Layout in ASP.NET Webforms

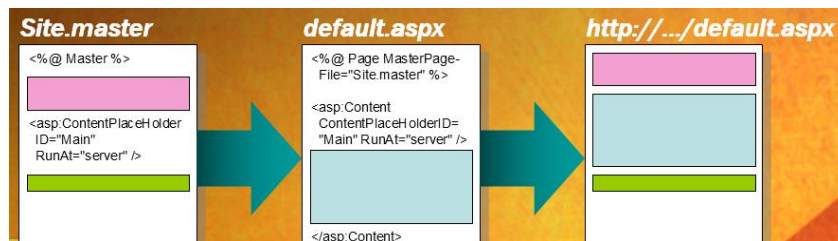
Master Pages



Master Page Basics

Masters define common content and placeholders (<asp:ContentPlaceholder>)

Content pages reference masters and fill placeholders with content (<asp:Content>)



Defining a Master Page

```
<%@ Master %>
<html>
<body>
  <!-- Banner shown on all pages that use this master -->
  <table width="100%">
    <tr>
      <td bgcolor="darkblue" align="center">
        <span style="font-size: 36pt; color: white">ACME Inc.</span>
      </td>
    </tr>
  </table>
  <!-- Placeholder for content below banner -->
  <asp:ContentPlaceholder ID="Main" RunAt="server" />
</body>
</html>
```


Applying a Master Page

```
<%@ Page MasterPageFile="~/Site.master" %>
<asp:Content ContentPlaceHolderID="Main" RunAt="server">
    This content fills the place holder "Main" defined in the master page
</asp:Content>
```

Applying a Master Page to a Site

```
<configuration>
  <system.web>
    <pages masterPageFile="~/Site.master" />
  </system.web>
</configuration>
```

Applying a Master Page Programmatically

```
void Page_PreInit (Object sender, EventArgs e)
{
    Page.MasterPageFile = "~/Site.master";
}
```

Default Content

ContentPlaceHolder controls can define content of their own ("default content")
Default content is displayed ONLY if not overridden by content page

```
<%@ Master %>
...
<asp:ContentPlaceHolder ID="Main" RunAt="server">
    This is default content that will appear in the absence of a
    matching Content control in a content page
</asp:ContentPlaceHolder>
```

The Page.Master Property

- Retrieves reference to master page
- Instance of class derived from System.Web.UI.MasterPage
- Null if page doesn't have a master
- Used to programmatically access content defined in the master page
 - Use FindControl for weak typing
 - Use public property in master page for strong typing (preferred)

Accessing a Control in the Master Page (Weak Typing)

In the Master Page

```
<asp:Label ID="Title" RunAt="server" />
```

In the Content Page

```
((Label) Master.FindControl ("Title")).Text = "Orders";
```

Accessing a Control in the Master Page (Strong Typing)

In the Master Page

```
<asp:Label ID="Title" RunAt="server" />
<script language="C#" runat="server">
    public string TitleText
    {
        get { return Title.Text; }
        set { Title.Text = value; }
    }
</script>
```

In the Content page

```
Master.TitleText = "Orders";
```

Nesting Master Pages

- Master pages can be nested
- Master pages that have masters must contain only Content controls, but Content controls can contain ContentPlaceHolders

```
<!-- Orders.Master -->
<%@ Master MasterPageFile="~/Site.Master" %>
<asp:Content ContentPlaceHolderID="..." RunAt="server">
    <asp:ContentPlaceHolder ID="..." RunAt="server">
        ...
    </asp:ContentPlaceHolder>
</asp:Content>
```

Themes and Skins

Mechanism for theming controls, pages, and sites by group-initializing control properties

Skin = Visual attributes for control(s)

Physically stored in .skin files

Default skins and named skins

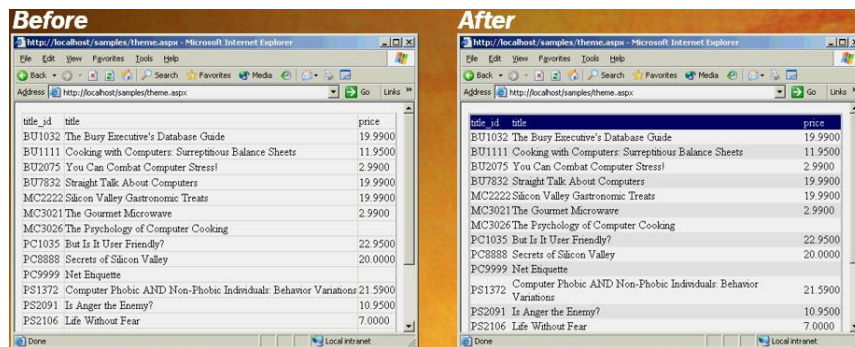
Theme = Collection of one or more skins

Physically stored in Themes subfolders

Global themes and local themes

Applying a Theme to a Page

```
<%@ Page Theme="BasicBlue"%>
```



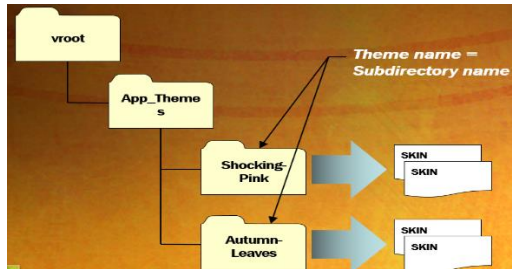
Applying a Theme to a Site

```
<configuration>
  <system.web>
    <pages theme="BasicBlue" />
  </system.web>
</configuration>
```

Applying a Theme Programmatically

```
void Page_PreInit (Object sender, EventArgs e)
{
    Page.Theme = "BasicBlue";
}
```

Themes



Defining Skins

```

<!-- Default look for DropDownList controls -->

<asp:DropDownList runat="server" BackColor="hotpink" ForeColor="white" />

<!-- Default look for DataGrid controls -->

<asp:DataGrid runat="server" BackColor="#CCCCCC" BorderWidth="2pt"
  BorderStyle="Solid" BorderColor="#CCCCCC" GridLines="Vertical"
  HorizontalAlign="Left">
  <HeaderStyle ForeColor="white" BackColor="hotpink" />
  <ItemStyle ForeColor="black" BackColor="white" />
  <AlternatingItemStyle BackColor="pink" ForeColor="black" />
</asp:DataGrid>
...

```

Named Skins

Defining Named Skins

```

<!-- Default look for DropDownList controls -->

<asp:DropDownList runat="server" BackColor="blue" ForeColor="white"
  SkinID="Blue" />

<!-- Default look for DataGrid conotrols -->

<asp:DataGrid runat="server" BackColor="#CCCCCC" BorderWidth="2pt"
  BorderStyle="Solid" BorderColor="#CCCCCC" GridLines="Vertical"
  HorizontalAlign="Left" SkinID="Blue">
  <HeaderStyle ForeColor="white" BackColor="blue" />
  <ItemStyle ForeColor="black" BackColor="white" />
  <AlternatingItemStyle BackColor="lightblue" ForeColor="black" />
</asp:DataGrid>
...

```

Using a Named Skin

```
<asp:DropDownList ID="Countries" SkinID="Blue" RunAt="server" />
```

The EnableTheming Property

Supported by all pages and controls

Defaults to true

Set EnableTheming to false to disable theming for individual controls or entire pages

```
<asp:DropDownList ID="Countries" EnableTheming="false" RunAt="server" />
```

IIS (Web Server)

Internet Information Server

- Microsoft's web server
- Foundation for ASP.NET
- Also FTP, NNTP, SMTP
- Shared resources
 - Default location c:\inetpub\wwwroot
 - Internet Services Manager
- A Microsoft Management Console (MMC) snap-in

IIS Virtual Directories

Provides a level of indirection from URL to actual file locations on the server

For example, the file for the url: <http://myServer/myApplication/foo.asp>
could be mapped to the physical location: d:\myFolder\myAppFolder\foo.asp

WebApplication

- All resources (**files, pages, handlers, modules, executable code**, etc.) within a virtual directory and its subdirectories
 - **Configuration files**
 - **Shared data (application variables)**
 - **global.asax**
 - **Scopes user sessions**
- A session is a series of web page hits by a single user within a block of time
- Shared data (session variables)

WebApplication global.asax

Located at application root

Can define and initialize application variables and session variables

Specify object to create with class, COM ProgID or COM ClassID

Be careful: use of shared objects can cause concurrency errors or blocking!

```
<object id="items" runat="server"
  scope="application"
  class="System.Collections.ArrayList" />
```

Can contain user-created code to handle application and session events (just like ASP)

Application_OnStart, Application_OnEnd

Session_OnStart, Session_OnEnd

```
void Application_OnStart () {
    Application ["startTime"]=DateTime.Now.ToString();
}

void Session_OnStart() {
    Session["startTime"]=DateTime.Now.ToString();
}
```

Application Configuration

Goal

Provide extensible configuration for admins & developers to hierarchically apply settings for an application

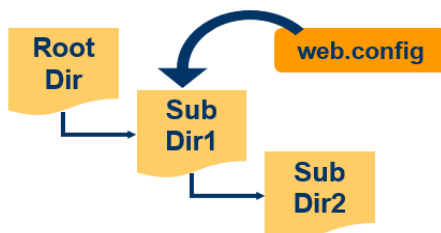
Solution

- Store configuration data in XML text files
 - Format is readable and writable by people and machines
 - Settings specified in configuration sections, e.g.
 - Security, SessionState, Compilation, CustomErrors, ProcessModel, HTTPHandlers, Globalization, AppSettings, WebServices, WebControls, etc.
- Configuration information stored in web.config
It is just a file, no DLL registration, no Registry settings, no Metabase settings

<!-- web.config can have comments -->

Configuration Hierarchy

Configuration files can be stored in application folders
Configuration system automatically detects changes
Hierarchical configuration architecture
Applies to the actual directory and all subdirectories



Configuration web.config Sample

```
<configuration>
  <configsections>
    <add names="httpmodules"
          type="System.Web.Config.HttpModulesConfigHandler"/>
    <add names="sessionState"
          type="..."/>
  </configsections>

  <httpModules>
    <!-- http module subelements go here -->
  </httpModules>
  <sessionState>
    <!-- sessionstate subelements go here -->
  </sessionState>
</configuration>
```

Configuration Hierarchy

Standard machine-wide configuration file
Provides standard set of configuration section handlers
Is inherited by all Web Applications on the machine

C:\WINDOWS\Microsoft.NET\Framework64\...\config\machine.config

Configuration User Defined Settings

Create web.config in appropriate folder

```
<configuration>
<appSettings>
  <add key="CxnString"
    value="localhost;uid=sa;pwd=;Database=foo"/>
</appSettings>
</configuration>
```

Retrieve settings at run-time

```
string cxnStr = ConfigurationSettings.AppSettings["CxnString"];
```

Custom Configuration Handlers

- Extend the set of section handlers with your own
- Implement the interface:
 System.Web.Configuration.IConfigurationSectionHandler
- Add to web.config or machine.config

Tracing

ASP.NET supports tracing

- Easy way to include “debug” statements
 - No more messy Response.Write() calls!
 - Debug statements can be left in, but turned off
- Great way to collect request details
 - Server control tree
 - Server variables, headers, cookies
 - Form/Query string parameters
 - Tracing provides a wealth of information about the page
- Can be enabled at page- or application- level

Tracing Methods and Properties

- Methods
 - Trace.Write: Writes category and text to trace
 - Trace.Warn: Writes category and text to trace in red
- Properties
 - Trace.IsEnabled: True if tracing is turned on for the application or just that page
 - Trace.Mode: SortByTime, SortByCategory

Page Level Tracing

To enable tracing for a single page:

1. Add trace directive at top of page
 - a. <%@ Page Trace="True" %>
2. Add trace calls throughout page
 - a. Trace.Write("MyApp", "Button Clicked");
 - b. Trace.Write("MyApp", "Value: " + value);
3. Access page from browser

Application Level Tracing

To enable tracing across multiple pages:
Create web.config file in application root

```
<configuration>
<trace enabled="true" requestlimit="10"/>
</configuration>
```

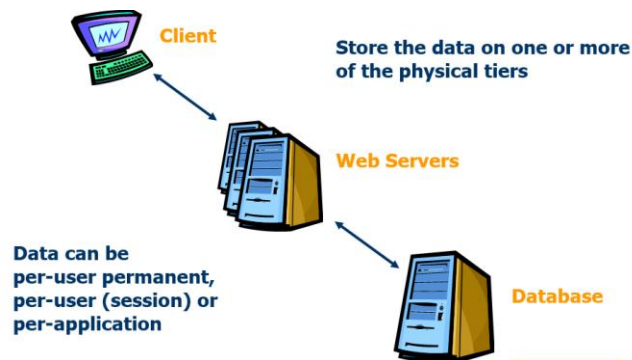
Hit one or more pages in the application
Access tracing URL for the application

```
http://localhost/MyApp/Trace.axd
```

State Management The Problem in Web Application

How/where to store data?
How can you pass data from page to page?
How do we get around HTTP statelessness?

State Management Three-Tier Architecture



State Management Client

Client requests an initial page
The server generates a HTTP/HTML response that is sent back to the client
This response includes data (state)
User looks at the response and makes a selection, causing another request to the server
This second request contains the data that was sent in the first response
The server receives and processes the data
Could be same server or different server

URL in a hyperlink (<a>)

Query string
Very visible to users
This can be good or bad

Hidden form elements

Like __VIEWSTATE

Cookies

Limited to 4K
May be blocked by users

State Management Web Server Middle-Tier

Application variables

Shared by all sessions (users)

Session variables

Still need to pass session id through the client
ASP.NET State Service or database

Caching

Similar to application variables, but can be updated periodically or based upon dependencies

State Management Database

Application-level

Part of application database design

Session-level

Custom session state management in database
ASP.NET database session state support

State Management in ASP.NET

ASP.NET supports both Application-level and Session-level state management
Allows you to store state (data) in middle tier

State Management Application Variables

- Application state is stored in an instance of `HttpApplicationState`
- Accessed from `Page.Application` property
- Can lock Application object for concurrent use
- Needed only when changing application variable
- Again, use this wisely
 - Use in "read-mostly" style
 - Initialize in `global.asax`
 - Avoid serializing your pages

State Management Sessions

- What is a session?
 - Context in which a user communicates with a server over multiple HTTP requests
 - Within the scope of an ASP.NET Application
- HTTP is a stateless, sessionless protocol
- ASP.NET adds the concept of "session"
 - Session identifier: 120 bit ASCII string
 - Session events: `Session_OnStart`, `Session_OnEnd`
 - Session variables: store data across multiple requests
 - ASP.NET improves upon ASP sessions

State Management Session Identifier

- By default, session id stored in a cookie
- Can optionally track session id in URL
- New in ASP.NET
- Requires no code changes to app
- All relative links continue to work

```
<configuration>
<sessionState cookieless="true"/>
</configuration>
```

State Management Session Variables

- ASP stores session state in IIS process
 - State is lost if IIS crashes
 - Can't use session state across machines
- ASP.NET stores session state:
 - In process
 - In another process: ASP State NT service
 - In SQL Server database

```
<sessionstate inproc="false"
server="AnotherServer" port="42424" />

<sessionstate inproc="false"
server="AnotherServer" port="42424"
usesqlserver="true" />
```

- "Live" objects are not stored in session state
 - Instead, ASP.NET serializes objects out between requests
- ASP.NET approach provides:
 - Ability to recover from application crashes
 - Ability to recover from IIS crash/restart
 - Can partition an application across multiple machines (called a Web Farm)
 - Can partition an application across multiple processes (called a Web Garden)

Transferring Controls between Pages

- Link to a page
- Postback
- Response.Redirect
- Causes HTTP redirect
- Tells browser to go to another URL
 - Server.Transfer
 - Like a redirect but entirely on one server
 - Server.Execute
 - Execute one page from another then return control
 - Both pages processed on same server

Caching

- Many sites spend considerable effort generating the same web pages over and over
 - For example, a product catalog changes overnight, but is accessed tens of thousands of times a day
- Server-side caching can vastly improve performance and scalability
 - ASP.NET provides support for
 - Page output caching
 - Data caching

Page out Caching

- Entire web page output (HTML) is cached
 - Must specify life of cached page (in seconds)
- Can cache multiple versions of a page, by:
 - GET/POST parameters; use VaryByParam
 - HTTP header; use VaryByHeader
 - E.g. Accept-Language
- Browser type or custom string; use VaryByCustom

Partial Pageoutput Caching

- Can cache a portion of a page by placing it in a User Control
- Can cache multiple versions of a User Control on a page, by property; use VaryByControl

Caching in the Browser

- Don't confuse server-side page output caching with how the browser and proxy servers cache the page
 - Use Response.Cache to specify HTTP cache headers
- Contains a HttpCachePolicy object

Data Caching

- Data cache is similar to application variables
- Can cache objects, HTML fragments, etc.
- Usage pattern:
 - Try to retrieve data
 - If null then create data and insert into cache

```
DataView Source = (DataView)Cache["MyData"];
if (Source == null) {
    Source = new DataView(ds.Tables["Authors"]);
    Cache["MyData"] = Source;          // Save in cache
}
```

- Cache object is stored on the Page and is an instance of **System.Web.Caching.Cache**
- Cache may be scavenged: when memory runs low it will be automatically reclaimed
- Can specify data expiration: absolute time (e.g. midnight), relative (in 1 hour)
- Cached data can be dependent upon a file or other cache item

```
Cache.Insert("MyData", Source, null, // Expire in 1 hour
    DateTime.Now.AddHours(1), TimeSpan.Zero);
Cache.Insert("MyData", Source,          // Dependent on file
    new CacheDependency(Server.MapPath("authors.xml")));
```

- Populating a data cache has an inherent race condition: if hit almost concurrently, multiple pages may try to populate the same cache
- This probably doesn't matter at all; it's only significant if the cost to create the data is prohibitive or if there are side effects
- If it does matter, there are two solutions:
 - Populate the cache in Application_OnStart
 - Synchronize access to the cache

```
private static String cacheSynchronize = "myKey";
DataView Source = (DataView)Cache["MyDataSet"];
if (Source == null) {
    lock (cacheSynchronize) {
        Source = (DataView)Cache["MyDataSet"];
        if (Source == null) {           // Have to test again
            // Open database ...
            Source = new DataView(ds.Tables["Authors"]);
            Cache["MyDataSet"] = Source;    // Save in cache
        }
    }
}
```

- ASP.NET page state maintenance is great, but __VIEWSTATE can get quite large
 - Why store constant data in __VIEWSTATE?
 - E.g. dropdowns listing countries, states, days of the week, months, product categories, SKUs, etc.
 - Instead, set EnableViewState=false, cache that data on the server, and populate the control from the cache in Page_Load
 - Can cache data or even HTML
 - Use Control.Render() to obtain a control's HTML

Error Handling

- .NET Common Language Runtime provides a unified exception architecture
- Runtime errors done using exceptions
- VB now supports try/catch/finally
- ASP.NET also provides declarative application custom error handling
- Automatically redirect users to error page when unhandled exceptions occur
- Prevents ugly error messages from being sent to users

Error Handling Custom Error pages

Can specify error pages for specific HTTP status codes in web.config

```
<configuration>
  <customerrors mode="remoteonly"
    defaultredirect="error.htm">
    <error statuscode="404"
      redirect="adminmessage.htm"/>
    <error statuscode="403"
      redirect="noaccessallowed.htm"/>
  </customerrors>
</configuration>
```

Error Handling Error Events

- Can override **Page.HandleError** to deal with any unhandled exception on that page
- What do you actually do when an error occurs?
 - Use new EventLog class to write custom events to log when errors occur
 - Use new SmtpMail class to send email to administrators

Error Handling writing to Event Log

```
<%@ Import Namespace="System.Diagnostics" %>
<%@ Assembly name="System.Diagnostics" %>
<script language="C#" runat=server>
public void Application_Error(object Sender, EventArgs E) {
    string LogName = "MyCustomAppLog";
    string Message = "Url " + Request.Path + " Error: " +
        this.Error.ToString()
    // Create event log if it doesn't exist
    if (! EventLog.SourceExists(LogName)) {
        EventLog.CreateEventSource(LogName, LogName);
    }
    // Fire off to event log
    EventLog Log = new EventLog();
    Log.Source = LogName;
    Log.WriteEntry(Message, EventLogEntryType.Error);
}
</script>
```

Error Handling Sending SMTP Mail

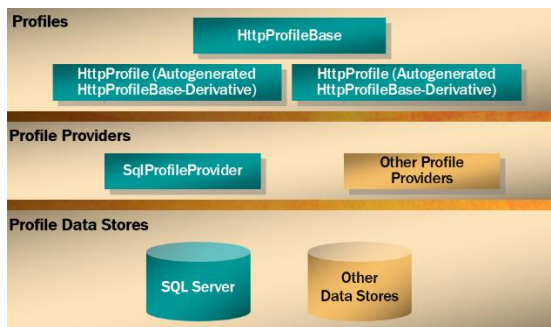
```
<%@ Import Namespace="System.Web.Util" %>
<%@ Assembly name="System.Diagnostics" %>
<script language="C#" runat=server>
public void Application_Error(object Sender, EventArgs E) {
    MailMessage MyMessage = new MailMessage();
    MyMessage.To = "scottgu@microsoft.com";
    MyMessage.From = "MyAppServer";
    MyMessage.Subject = "Unhandled Error!!!";
    MyMessage.BodyFormat = MailFormat.Html;
    MyMessage.Body = "<html><body><h1>" + Request.Path +
        "</h1>" + Me.Error.ToString() + "</body></html>";
    SmtpMail.Send(MyMessage);
}
</script>
```


Personalization

Profile Service

- Stores per-user data persistently
 - Strongly typed access (unlike session state)
 - On-demand lookup (unlike session state)
 - Long-lived (unlike session state)
 - Supports authenticated and anonymous users
- Accessed through dynamically compiled `HttpProfileBase` derivatives (`HttpProfile`)
- Provider-based for flexible data storage

Profile Schema



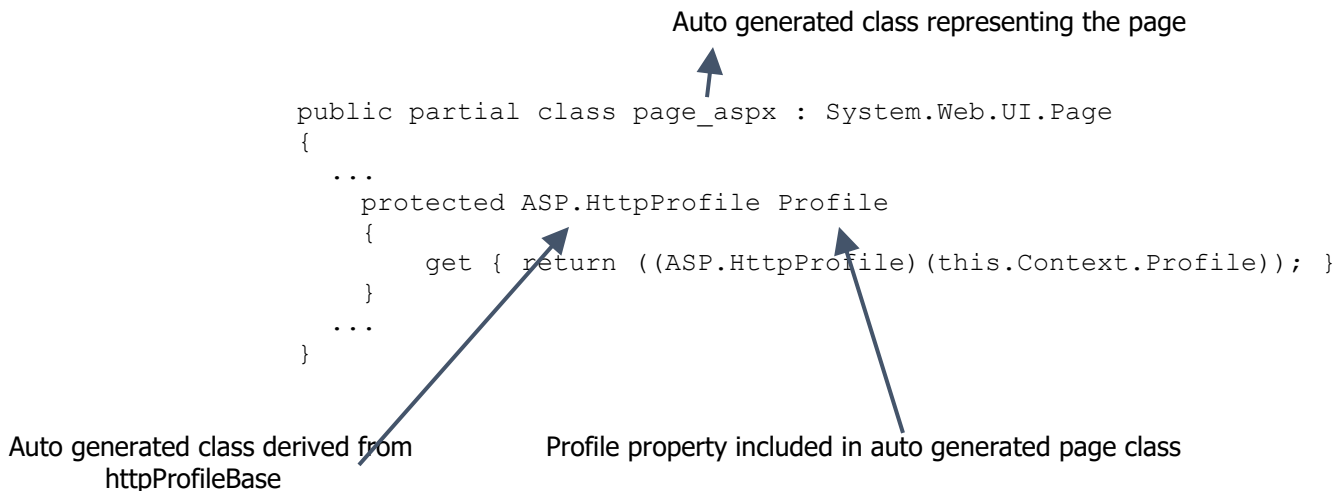
Defining a Profile

```
<configuration>
  <system.web>
    <profile>
      <properties>
        <add name="ScreenName" />
        <add name="Posts" type="System.Int32" defaultValue="0" />
        <add name="LastPost" type="System.DateTime" />
      </properties>
    </profile>
  </system.web>
</configuration>
```

Using a Profile

```
// Increment the current user's post count
Profile.Posts = Profile.Posts + 1;
// Update the current user's last post date
Profile.LastPost = DateTime.Now;
```

How Profile Works



Accessing another User's Profile

- `Profile.propertyname` refers to current user
- Use `Profile.GetProfile (username)` to access profiles for other users

```
// Get a reference to Fred's profile
HttpProfile profile = Profile.GetProfile ("Fred");
// Increment Fred's post count
profile.Posts = profile.Posts + 1;
// Update Fred's last post date
profile.LastPost = DateTime.Now;
```

Accessing a profile from External Component

- "Profile" property is only valid in classes generated by ASP.NET (ASPX, ASAX, etc.)
- Use `HttpContext.Profile` property to access profiles from external components

```
// Read the current user's ScreenName property in an ASPX file
string name = Profile.ScreenName;
// Read the current user's ScreenName property in an external component
string name = (string) HttpContext.Current.Profile["ScreenName"];
```

Profile Groups

- Properties can be grouped
- `<group>` element defines groups

```
<profile>
  <properties>
    <add ... />
    <group name="...">
      <add ... />
    </group>
  </properties>
</profile>
```

Defining a Profile Group

```
<configuration>
  <system.web>
    <profile>
      <properties>
        <add name="ScreenName" />
        <group name="Forums">
          <add name="Posts" type="System.Int32" defaultValue="0" />
          <add name="LastPost" type="System.DateTime" />
        </group>
      </properties>
    </profile>
  </system.web>
</configuration>
```

Accessing a Profile Group

```
// Increment the current user's post count
Profile.Forums.Posts = Profile.Forums.Posts + 1;

// Update the current user's last post date
Profile.Forums.LastPost = DateTime.Now;
```

Custom DataType

Profiles support base types: String, Int32, Int64, DateTime, Decimal, etc.

Profiles support custom types: Use type attribute to specify type

Use serializeAs attribute to specify serialization mode: Binary, Xml (default), or String

serializeAs="Binary" types must be serializable

serializeAs="String" types need type converters

Using a Custom DataType

```
<configuration>
  <system.web>
    <profile>
      <properties>
        <add name="Cart" type="ShoppingCart" serializeAs="Binary" />
      </properties>
    </profile>
  </system.web>
</configuration>
```

Anonymous User Profiles

- By default, profiles aren't available for anonymous (unauthenticated) users
 - Data keyed by authenticated user IDs
- Anonymous profiles can be enabled
 - Step 1: Enable anonymous identification
 - Step 2: Specify which profile properties are available to anonymous users
- Data keyed by user anonymous IDs

Profiles for Anonymous Users

```
<configuration>
  <system.web>
    <anonymousIdentification enabled="true" />
    <profile>
      <properties>
        <add name="ScreenName" allowAnonymous="true" />
        <add name="Posts" type="System.Int32" defaultValue="0" />
        <add name="LastPost" type="System.DateTime" />
      </properties>
    </profile>
  </system.web>
</configuration>
```

Anonymous Identification

Anonymous identification can be cookieless or cookieless (URL munging)
Cookies can be configured

```
<anonymousIdentification
  enabled=" [true | false] "
  cookieName=" .ASPXANONYMOUS "
  cookieTimeout="100000"
  cookiePath="/"
  cookieRequireSSL=" [true | false] "
  cookieSlidingExpiration=" [true | false] "
  cookieProtection=" [None | Validation | Encryption | All] "
  cookieless=" [UseCookies | UseUri | AutoDetect | UseDeviceProfile] "
/>
```

Profile Events

Profile service and anonymous identification service fire global events

Global.asax Handler Name	Description
Anonymousidentification_Create	Called when anonymous ID is issued
Anonymousidentification_Remove	Called when authentication request contains anonymous ID and ID is therefore deleted
Profile_MigrateAnonymous	Called other anonymousidentification_Remove to migrate settings for anonymous users

Localization

- @ Page (UI)Culture "auto" keyword
 - Declarative mapping of Accept-Language headers to relevant thread properties
- Simplified resource handling
 - Declarative mapping of control properties to resources using <%\$... %> expressions
- Strongly typed programmatic resource loading
 - <localize runat="server"> and more

Culture Handling (ASP.NET 1.x)

Code required to map Accept-Language headers to Current (UI) Culture properties of current thread

Global.asax

```
<script language="C#" runat="server">
    void Application_BeginRequest (Object sender, EventArgs e)
    {
        Thread.CurrentThread.CurrentCulture =
            CultureInfo.CreateSpecificCulture (Request.UserLanguages [0] );
        Thread.CurrentThread.CurrentUICulture =
            Thread.CurrentThread.CurrentCulture;
    }
</script>
```

Culture Handling (ASP.NET)

"auto" keyword maps Accept-Language headers to Current(UI)Culture properties of current thread
Single Page (ASPX)

```
<%@ Page Culture="auto" UICulture="auto" %>
```

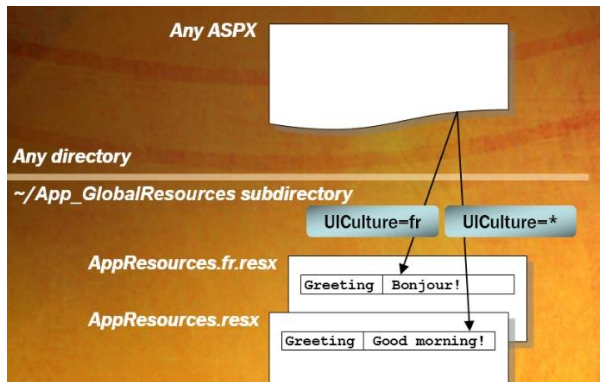
All pages (Web.config)

```
<globalization culture="auto" uiCulture="auto" />
```

Localization Resources

- Application resources
 - Available to all pages in application
 - RESX files in ~/App_GlobalResources subdirectory
- Local resources
 - Available to a specific page
 - RESX files in App_LocalResources subdirectories
- Use filename.culture.resx naming schema for localization based on CurrentUICulture

Application Resources



Loading Application Resources

Declarative

```
<asp:Label ID="Output" RunAt="server"
    Text='<%$ Resources:AppResources, Greeting %>' />
```

(RESX Name: AppResources) (Resource Name: Greeting)

Programmatic

```
// Strong typing
Output.Text = Resources.AppResources.Greeting;
// Weak typing
Output.Text = (string) GetAppResourceObject ("AppResources", "Greeting");
```

Loading Local Resources

Declarative

```
<asp:Label ID="Output" RunAt="server" Text='<%$ Resources:Greeting%>' />
```

Resource Name

Programmatic

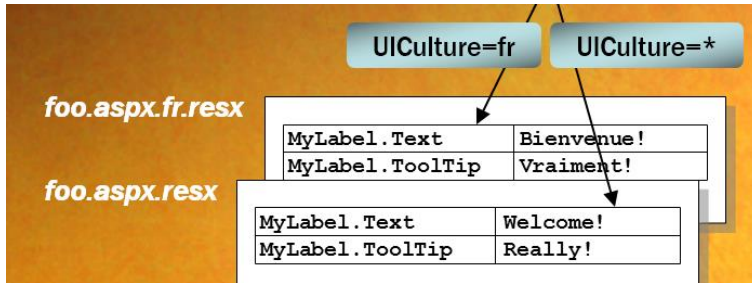
```
// Weak typing only
Output.Text = (string) GetPageResourceObject ("Greeting");
```

Implicit Expressions

Batch-initialize control properties
Applicable to local resources only

foo.aspx

```
<asp:Label ID="Output" RunAt="server" meta:ResourceKey="MyLabel" />
```



The Localize Control

Loads static content from resources based on Thread.CurrentThread.CurrentUICulture
foo.aspx

```
<asp:Localize Runat="server" Text="<%$ Resources:WelcomeMessage %>" />
```



Panel. Direction

- Direction="RightToLeft" right-justifies content generated by child controls
- Use it to right-justify right-to-left text
- Supported at design time by VS IDE

```
<asp:Panel Direction="RightToLeft" RunAt="server">
  Imagine this were in Hebrew<br />
  <asp:Login RunAt="server" />
</asp:Panel>
```

ASP.NET WebApplication Deployment

- **XCOPY deployment**
 - Components are placed in .\bin folder
- **No DLL deployment, registration**
 - Unless you're using COM or other DLLs
- **No locked DLLs**
 - DLLs are "shadow copied" into a hidden folder
 - .aspx files are automatically compiled
 - Not true for codebehind
- Update code (.aspx and assemblies) while server is running
- No need to stop/bounce the server
- **Applications are isolated**
 - Each can have their own version of components
- **Uninstall = delete /s *.***

Security

- Reasons for Security
 - Prevent access to areas of your Web server
 - Record and store secure relevant user data
- Security Configuration
 - <security> tag in web.config file
- Authentication, Authorization, Impersonation
- Code Access Security
 - Are you the code you told me you are?
 - Protect your server from bad code

Security Authentication

- Who are you?
- Server must authenticate client
- Client should authenticate server
 - Kerberos does
- Need a directory to store user accounts
 - NT: Security Accounts Manager
 - Good for intranet usage up to about 40,000 accounts
 - Windows 2000: Active Directory
 - Good for intranet and Internet usage

Security IIS Authentication

- Anonymous
 - A single W2K/NT account is used for all visitors
- Basic authentication: Standard, commonly supported, Password sent in clear text
- Digest authentication: Standard, but not yet common
- Integrated Windows Authentication: NTLM, Kerberos (Windows 2000 only)
- Client certificates: Mapped to W2K/NT account

Security ASP.NET Authentication

- Custom, **forms-based authentication**
- Easy to use, with cookie token tracking
- Enables custom login screen (no popup dialogs)
- Supports custom credential checks against database, exchange, etc.

Security ASP.NET Authorization

- ASP.NET supports authorization using either users or roles
- ASP.NET uses Membership and Roles Management API for Authentication and Authorization in web Applications.
- Roles map users into logical groups

Example: "User", "Manager", "VP", etc.

Provides nice developer/admin separation

Developers can perform runtime role checks in code

```
if (User.IsInRole("Admin")) { }
```

Security Impersonation

- IIS authenticates the "user"
- A token is passed to the ASP.NET application
- ASP.NET impersonates the given token
- Access is permitted according to NTFS settings

ASP.NET MVC

ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that enables a clean separation of concerns and that gives you full control over markup for enjoyable, agile development. ASP.NET MVC includes many features that enable fast, TDD-friendly development for creating sophisticated applications that use the latest web standards.

ASP.NET: One Web application framework to rule them all...

Caching	Modules	Globalization
Pages	Controls	Master Pages
Profile	Roles	Membership
Handlers	Etc.	

There are two major considerations for choosing between Webforms and ASP.NET MVC:

- **Test Driven Development** – life would be MUCH easier using MVC while following TDD.
- **Data Driven Application** – life would be MUCH easier using WebForms if the application is data heavy.

There are no rights or wrongs, and every application can be written in both frameworks. In fact, you can even have a hybrid approach, where you can write some part in WebForms and some in MVC.

SWOT Analysis Webforms and ASP.NET MVC

Asp.Net WebForms

Strengths

- Provides very good RAD development capabilities, Great designer support in Visual Studio.
- Ease of development for data-heavy LOB applications, Very rich control libraries and third party vendor support.
- A familiar event-driven model when compared to Windows Forms development, and so easy for developers to pick up.

Weaknesses

- UI logic coupled with the code, and thus is hard to separate.
- Harder to unit test, so difficult to employ TDD.
- Heavy page sizes due to view state management.

Opportunities

- Great at creating quick prototypes for business applications. This comes in very handy when trying to show quick Proof of Concepts to clients.

Threats

- Harder to adopt to various UI views despite the various frameworks available (master pages, Themes, etc.).

WebForms is great, but options are good.....

Classic ASP.NET (Web Forms)

- High level abstraction over HTML / HTTP
- Simplified state management
- ViewState and the post-back model
- Control model
- Data binding
- Simple event-driven mechanism
- Simple Page Controller pattern

What makes it lazy web forms

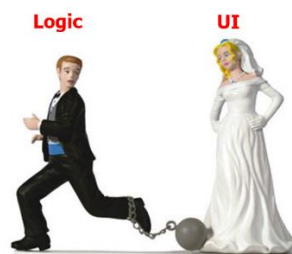
- Sub-optimal URLs
- `blog.aspx?date=21032008`
- `Form runat="server"`
- ViewState
- Hard to test
- All sorts of code in the page
- Requirement to test with an `HttpContext`

No real role responsibility...

Pages, Master pages, UI logic, Business logic, Data Access everything together with the help of powerful Webserver controls.

Control abstractions can be negative...

It isn't easy enough to test...

[illegible]

Asp.Net MVC

Strengths

- Provides fine control over rendered HTML.
- Cleaner generated HTML.
- Superior separation between UI and code.
- Easier to unit test.
- Can support multiple view engines.
- By default uses RESTful interfaces for URLs – so better SEO.
- No ViewState (this may also be a weakness).
- Typical size of page is small.
- Easy integration with frameworks like JQuery.

Weaknesses

- Not event driven, so maybe difficult for people who know only Asp.Net Webforms to wrap their minds around it.
- Third party control library support is not that strong.
- No ViewState (this is also a strength).

Opportunities

- Allows for Test Driven Development (TDD) – it is built with TDD in mind, so its much easier to write unit test cases, mock objects, and to intercept the program flow.
- Allows for reuse of the same models to present different UIs and Interfaces.

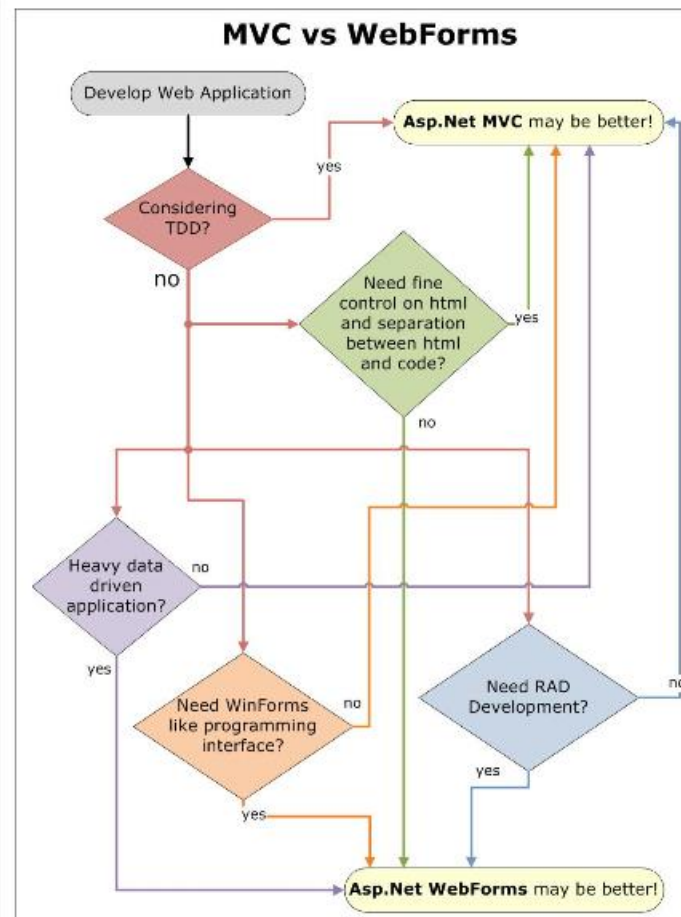
Threats

- Bigger ramp-up and training time required for developers with no or little experience in web application development.

ASP.NET MVC Tenets

Alternative
Testable
Extensible
Routable

Choosing between ASP.NET Webforms and ASP.NET MVC



ASP.NET MVC doesn't have...

- Postbacks
- View state
- Control state
- Server-side form
- Page/Control lifecycle

ASP.NET MVC still has...

- Web designer
- Master pages & User controls
- Membership/Roles/Profile
- Globalization & Caching
- HTTP intrinsics:
HttpContext
HttpRequest
HttpResponse
Etc.

ASP.NET Routing

- Routing provides "clean" URLs
- URL is mapped to a route handler
- Extra level of indirection
- Handlers can be changed without impacting URL
- URL can be changed without impacting handler
- Enables support for multilingual URLs
- URL Example: <http://www.transflower.in/Home/trainings>

Developers adds Routes to a global RouteTable
Mapping creates a RouteData - a bag of key/values

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
    routes.MapRoute( "Default",
        "{controller}/{action}/{id}",
        new { controller = "Home", // Parameter defaults
            action = "Index", action = "Index", id = "" } );
}

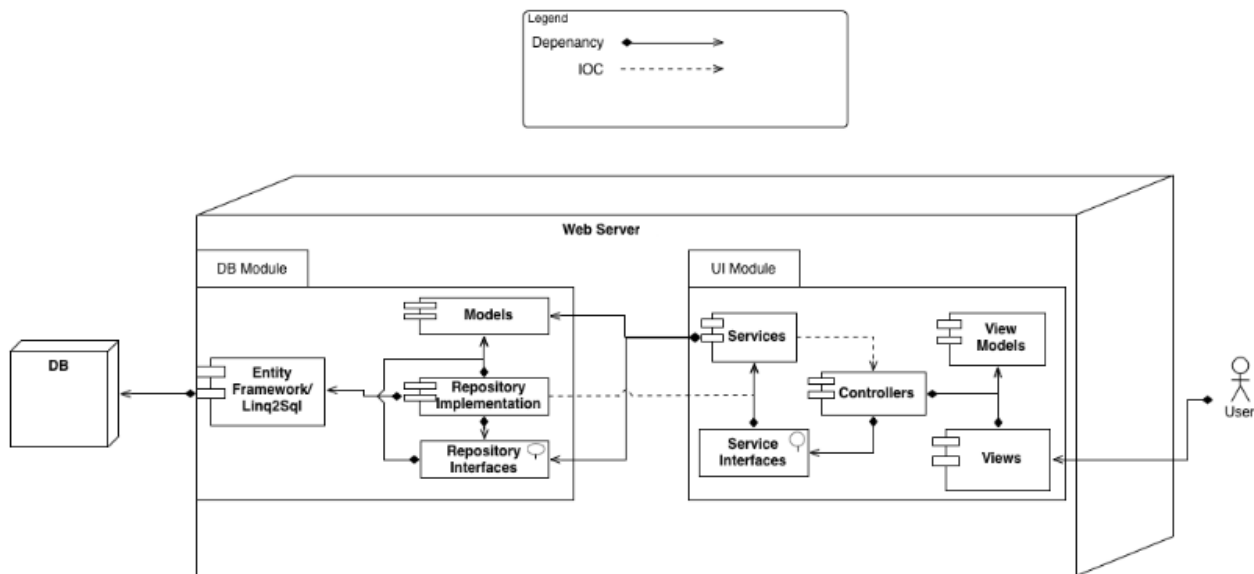
protected void Application_Start()
{
    RegisterRoutes(RouteTable.Routes);
}
```

ASP.NET MVC

Separation of concerns

Model
View
Controller

How does MVC Look?



MVC Controllers

- Scenarios, Goals and Design
 - URLs route to controller "actions",
 - not pages – mark actions in Controller.
 - Controller executes logic, chooses view.
 - All public methods are accessible

```

public void ShowPost(int id) {
    Post p = PostRepository.GetPostById(id);
    if (p != null) {
        RenderView("showpost", p);
    } else {
        RenderView("nosuchpost", id);
    }
}

```

Controller Conventions

Controller

Must...

- Be suffixed with "Controller"
- Implement `Controller` (or inherit from `Controller`)

Action

Must...

- Be Public
- Return `ActionResult` or void

Can't...

- Be generic

Have a `NonActionAttribute`

Have out/ref parameters

Views

- Are for rendering/output.
- Pre-defined and extensible rendering helpers
- Can use `.ASPX`, `.ASCX`, `.MASTER`, etc.
- Can replace with other view technologies
- Template engines (NVelocity, Brail, ...).
- Output formats (images, RSS, JSON, ...).
- Mock out for testing.
- Controller sets data on the View
- Loosely typed or strongly typed data

Clean URLs

```
//Don't settle for...
```

```
/Products.aspx?CategoryID=123
```

```
//When you can easily have...
```

```
/Product/Puppies
```


ASP.NET MVC Features

- Razor View Engine
- Multiple View Engine Support
- Validation Improvements
- Dynamic ViewBag
- Global Filters
- Action Results
- Taskbased Helpers
- Dependency Injection
- Porting MVC Libraries to jQuery
- Granular Validation Input
- Caching Support
- JSON Binding Support

```
protected void Application_Start()
{
    ViewEngines.Engines.Add(new SparkViewFactory());
    ...
}
```

Razor

The Razor View Engine

Razor syntax is clean and concise, requiring a minimum number of keystrokes.

Razor is easy to learn, in part because it's based on existing languages like C# and Visual Basic.

Visual Studio includes IntelliSense and code colorization for Razor syntax.

Razor views can be unit tested without requiring that you run the application or launch a web server.

Syntax Comparison

Using traditional asp.net code

```
<h1>Code Nugget Example with .ASPX</h1>
<h3>Hello <%=name %>, the year is <%= DateTime.Now.Year %></h3>
<p>Checkout <a href="/Products/Details/<%=productId %">this product</a>
</p>
```

Using Razor

```
<h1>Razor Example</h1>
<h3>
    Hello @name, the year is @DateTime.Now.Year
</h3>
<p>
    Checkout <a href="="/Products/Details/@productId">this product</a>
</p>
```

Dynamic View Bag

MVC 2 controllers support a ViewData property that enables you to pass data to a view template using a late-bound dictionary API.

In MVC 3, you can also use somewhat simpler syntax with the ViewBag property to accomplish the same purpose.

For example, instead of writing **ViewData["Message"] = "text"**, you can write **ViewBag.Message="text"**.

You do not need to define any strongly-typed classes to use the ViewBagproperty.

Dynamic property, you can instead just get or set properties and it will resolve them dynamically at run time.

```
<head>
  <title>@ViewBag.Title<title>
</head>
```

Filters

ASP.NET MVC provides a simple way to inject your piece of code or logic either before or after an action is executed. This is achieved by decorating the controllers or actions with ASP.NET MVC attributes or custom attributes. An attribute or custom attribute implements the ASP.NET MVC filters (filter interface) and can contain your piece of code or logic. You can make your own custom filters or attributes either by implementing ASP.NET MVC filter interface or by inheriting and overriding methods of ASP.NET MVC filter attribute class if available.

Typically, Filters are used to perform the following common functionalities in your ASP.NET MVC application.

1. **Custom Authentication**
2. **Custom Authorization (User based or Role based)**
3. **Error handling or logging**
4. **User Activity Logging**
5. **Data Caching**
6. **Data Compression**

The ASP.NET MVC framework provides five types of filters.

1. **Authentication filters**
2. **Authorization filters**
3. **Action filters**
4. **Result filters**
5. **Exception filters**

Authentication Filters

This filter is introduced with ASP.NET MVC5. The `IAuthenticationFilter` interface is used to create CustomAuthentication filter. The definition of this interface is given below:

```
public interface IAuthenticationFilter
{
    void OnAuthentication(AuthenticationContext filterContext);

    void OnAuthenticationChallenge(AuthenticationChallengeContext filterContext);
}
```

You can create your CustomAuthentication filter attribute by implementing `IAuthenticationFilter` as shown below-

```
public class CustomAuthenticationAttribute : ActionFilterAttribute, IAuthenticationFilter
{
    public void OnAuthentication(AuthenticationContext filterContext)
    {
        //Logic for authenticating a user
    }
    //Runs after the OnAuthentication method
    public void OnAuthenticationChallenge(AuthenticationChallengeContext filterContext)
    {
        //TODO: Additional tasks on the request
    }
}
```

Authorization Filters

The ASP.NET MVC Authorize filter attribute implements the `IAuthorizationFilter` interface. The definition of this interface is given below-

```
public interface IAuthorizationFilter
{
    void OnAuthorization(AuthorizationContext filterContext);
}
```

The `AuthorizeAttribute` class provides the following methods to override in the `CustomAuthorize` attribute class.

```
public class AuthorizeAttribute : FilterAttribute, IAuthorizationFilter
{
    protected virtual bool AuthorizeCore(HttpContextBase httpContext);
    protected virtual void HandleUnauthorizedRequest(AuthorizationContext filterContext);
    public virtual void OnAuthorization(AuthorizationContext filterContext);
    protected virtual HttpValidationStatus OnCacheAuthorization(HttpContextBase httpContext);
}
```

In this way you can make your `CustomAuthorize` filter attribute either by implementing `IAuthorizationFilter` interface or by inheriting and overriding above methods of `AuthorizeAttribute` class.

Action Filters

Action filters are executed before or after an action is executed. The **`IActionFilter`** interface is used to create an Action Filter which provides two methods **`OnActionExecuting`** and **`OnActionExecuted`** which will be executed before or after an action is executed respectively.

```
public interface IActionFilter
{
    void OnActionExecuting(ActionExecutingContext filterContext);
    void OnActionExecuted(ActionExecutedContext filterContext);
}
```

Result Filters

Result filters are executed before or after generating the result for an action. The Action Result type can be `ViewResult`, `PartialViewResult`, `RedirectToRouteResult`, `RedirectResult`, `ContentResult`, `JsonResult`, `FileResult` and `EmptyResult` which derives from the `ActionResult` class. Result filters are called after the Action filters. The `IResultFilter` interface is used to create an Result Filter which provides two methods `OnResultExecuting` and `OnResultExecuted` which will be executed before or after generating the result for an action respectively.

```
public interface IResultFilter
{
    void OnResultExecuted(ResultExecutedContext filterContext);
    void OnResultExecuting(ResultExecutingContext filterContext);
}
```

Exception Filters

Exception filters are executed when exception occurs during the actions execution or filters execution. The `IExceptionHandler` interface is used to create an Exception Filter which provides `OnException` method which will be executed when exception occurs during the actions execution or filters execution.

```
public interface IExceptionHandler
{
    void OnException(ExceptionContext filterContext);
}
```

ASP.NET MVC `HandleErrorAttribute` filter is an Exception filter which implements `IExceptionHandler`. When `HandleErrorAttribute` filter receives the exception it returns an Error view located in the `Views/Shared` folder of your ASP.NET MVC application.

Order of Filter Execution

All ASP.NET MVC filter are executed in an order. The correct order of execution is given below:

1. **Authentication filters**
2. **Authorization filters**
3. **Action filters**
4. **Result filters**

Configuring Filters

You can configure your own custom filter into your application at following three levels:

Global level

By registering your filter into `Application_Start` event of `Global.asax.cs` file with the help of `FilterConfig` class.

```
protected void Application_Start()
{
    FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
}
```

Controller level

By putting your filter on the top of the controller name as shown below-

```
[Authorize(Roles="Admin")]
public class AdminController : Controller
{
    //
}
```

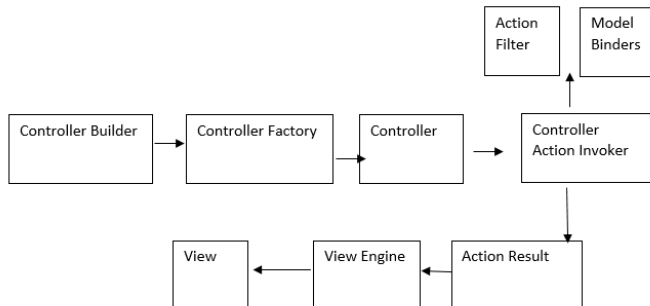
Action level

By putting your filter on the top of the action name as shown below-

```
public class UserController : Controller
{
    [Authorize(Users="User1,User2")]
    public ActionResult LinkLogin(string provider)
    {
        // TODO:
        return View();
    }
}
```

ASP.NET MVC Extensibility

Any of these ASP.NET MVC components can be replaced.



Dependency Injection Improvements

ASP.NET MVC provides better support for applying Dependency Injection (DI) and for integrating with Dependency Injection or Inversion of Control (IOC) containers.

Support for DI has been added in the following areas:

- Controllers (registering and injecting controller factories, injecting controllers).
- Views (registering and injecting view engines, injecting dependencies into view pages).
- Action filters (locating and injecting filters).
- Model binders (registering and injecting).
- Model validation providers (registering and injecting).
- Model metadata providers (registering and injecting).
- Value providers (registering and injecting).

Model Binding

Model binding is the process of creating .NET objects using the data sent by the browser in an HTTP request. Default model binder search in following location and order for named parameters data.

Name	Description
Request.Form	Content HTML form element data
RouteData.Values	Application routes values
Request.QueryString	Data in the query string of the request URL
Request.Files	Files that have been uploaded as part of the request

Bind attribute:

- To include or exclude model properties from the binding process.
- To include first and last name in person object. Person FirstName and LastName property value will be consider other will ignored.

```
public ActionResult Register([Bind(Include = "FirstName, LastName")] Person person)
```

Example: To exclude sex property in person object.
Person sex property value will be ignored.

```
public ActionResult Register([Bind(Exclude = "Sex")] Person person)
```


MVC Interview Questions/Answers

What are the 3 main components of an ASP.NET MVC application?

1. M - Model
2. V - View
3. C - Controller

In which assembly is the MVC framework defined?

System.Web.Mvc

Is it possible to combine ASP.NET webforms and ASP.MVC and develop a single web application?

Yes, it is possible to combine ASP.NET webforms and ASP.MVC and develop a single web application.

What does Model, View and Controller represent in an MVC application?

Model: Model represents the application data domain. In short the applications business logic is contained with in the model.

View: Views represent the user interface, with which the end users interact. In short the all the user interface logic is contained with in the UI.

Controller: Controller is the component that responds to user actions. Based on the user actions, the respective controller, work with the model, and selects a view to render that displays the user interface. The user input logic is contained with in the controller.

What is the greatest advantage of using asp.net mvc over asp.net webforms?

It is difficult to unit test UI with webforms, where views in mvc can be very easily unit tested.

Which approach provides better support for test driven development - ASP.NET MVC or ASP.NET Webforms?

ASP.NET MVC

What are the advantages of ASP.NET MVC?

1. Extensive support for TDD. With asp.net MVC, views can also be very easily unit tested.
2. Complex applications can be easily managed
3. Separation of concerns. Different aspects of the application can be divided into Model, View and Controller.
4. ASP.NET MVC views are light weight, as they donot use viewstate.

Is it possible to unit test an MVC application without running the controllers in an ASP.NET process?

Yes, all the features in an asp.net MVC application are interface based and hence mocking is much easier. So, we don't have to run the controllers in an ASP.NET process for unit testing.

Is it possible to share a view across multiple controllers?

Yes, put the view into the shared folder. This will automatically make the view available across multiple controllers.

What is the role of a controller in an MVC application?

The controller responds to user interactions, with the application, by selecting the action method to execute and also selecting the view to render.

Where are the routing rules defined in an asp.net MVC application?

In Application_Start event in Global.asax

Name a few different return types of a controller action method?

The following are just a few return types of a controller action method. In general an action method can return an instance of a any class that derives from ActionResult class.

1. ViewResult
2. JavaScriptResult
3. RedirectResult
4. ContentResult
5. JsonResult

What is the significance of NonActionAttribute?

In general, all public methods of a controller class are treated as action methods. If you want prevent this default behaviour, just decorate the public method with NonActionAttribute.

What is the significance of ASP.NET routing?

ASP.NET MVC uses ASP.NET routing, to map incoming browser requests to controller action methods. ASP.NET Routing makes use of route table. Route table is created when your web application first starts. The route table is present in the Global.asax file.

What are the 3 segments of the default route, that is present in an ASP.NET MVC application?

- 1st Segment - Controller Name
- 2nd Segment - Action Method Name
- 3rd Segment - Parameter that is passed to the action method

Example: <http://pragimtech.com/Customer/Details/5>

Controller Name = Customer

Action Method Name = Details

Parameter Id = 5

ASP.NET MVC application, makes use of settings at 2 places for routing to work correctly. What are these 2 places?

1. Web.Config File : ASP.NET routing has to be enabled here.
2. Global.asax File : The Route table is created in the application Start event handler, of the Global.asax file.

What is the advantage of using ASP.NET routing?

In an ASP.NET web application that does not make use of routing, an incoming browser request should map to a physical file. If the file does not exist, we get page not found error.

An ASP.NET web application that does make use of routing, makes use of URLs that do not have to map to specific files in a Web site. Because the URL does not have to map to a file, you can use URLs that are descriptive of the user's action and therefore are more easily understood by users.

What are the 3 things that are needed to specify a route?

1. URL Pattern - You can include placeholders in a URL pattern so that variable data can be passed to the request handler without requiring a query string.
2. Handler - The handler can be a physical file such as an .aspx file or a controller class.
3. Name for the Route - Name is optional.

Is the following route definition a valid route definition?

{controller}/{action}/{id}

No, the above definition is not a valid route definition, because there is no literal value or delimiter between the placeholders. Therefore, routing cannot determine where to separate the value for the controller placeholder from the value for the action placeholder.

What is the use of the following default route?

{resource}.axd/{*pathInfo}

This route definition, prevent requests for the Web resource files such as WebResource.axd or ScriptResource.axd from being passed to a controller.

What is the difference between adding routes, to a webforms application and to an mvc application?

To add routes to a webforms application, we use MapPageRoute() method of the RouteCollection class, where as to add routes to an MVC application we use MapRoute() method.

How do you handle variable number of segments in a route definition?

Use a route with a catch-all parameter. An example is shown below. * is referred to as catch-all parameter.
controller/{action}/{*parametervalues}

What are the 2 ways of adding constraints to a route?

1. Use regular expressions
2. Use an object that implements IRouteConstraint interface

Give 2 examples for scenarios when routing is not applied?

1. A Physical File is Found that Matches the URL Pattern - This default behaviour can be overridden by setting the RouteExistingFiles property of the RouteCollection object to true.
2. Routing Is Explicitly Disabled for a URL Pattern - Use the RouteCollection.Ignore() method to prevent routing from handling certain requests.

What is the use of action filters in an MVC application?

Action Filters allow us to add pre-action and post-action behavior to controller action methods.

If I have multiple filters impleted, what is the order in which these filters get executed?

1. Authorization filters
2. Action filters
3. Response filters
4. Exception filters

What are the different types of filters, in an asp.net mvc application?

1. Authorization filters
2. Action filters
3. Result filters
4. Exception filters

Give an example for Authorization filters in an asp.net mvc application?

1. RequireHttpsAttribute
2. AuthorizeAttribute

Which filter executes first in an asp.net mvc application?

Authorization filter

What are the levels at which filters can be applied in an asp.net mvc application?

1. Action Method
2. Controller
3. Application

Is it possible to create a custom filter?

Yes

What filters are executed in the end?

Exception Filters

Is it possible to cancel filter execution?

Yes

What type of filter does OutputCacheAttribute class represents?

Result Filter

What are the 2 popular asp.net mvc view engines?

1. Razor
2. .aspx

What symbol would you use to denote, the start of a code block in razor views?

@

What symbol would you use to denote, the start of a code block in aspx views?

<%= %>

In razor syntax, what is the escape sequence character for @ symbol?

The escape sequence character for @ symbol, is another @ symbol

When using razor views, do you have to take any special steps to protect your asp.net mvc application from cross site scripting (XSS) attacks?

No, by default content emitted using a @ block is automatically HTML encoded to protect from cross site scripting (XSS) attacks.

When using aspx view engine, to have a consistent look and feel, across all pages of the application, we can make use of asp.net master pages. What is asp.net master pages equivalent, when using razor views?

To have a consistent look and feel when using razor views, we can make use of layout pages. Layout pages, reside in the shared folder, and are named as _Layout.cshtml

What are sections?

Layout pages, can define sections, which can then be overridden by specific views making use of the layout. Defining and overriding sections is optional.

What are the file extensions for razor views?

1. .cshtml - If the programming language is C#
2. .vbhtml - If the programming language is VB

How do you specify comments using razor syntax?

Razor syntax makes use of @* to indicate the beginning of a comment and *@ to indicate the end. An example is shown below.

@* This is a Comment *@

LINQ Language Integrated Query

The C# 3.0 language enhancements build on C# 2.0 to increase developer productivity: they make written code more concise and make working with data as easy as working with objects. These features provide the foundation for the LINQ project, a general purpose declarative query facility that can be applied to in-memory collections and data stored in external sources such as XML files and relational databases.

The C# 3.0 language enhancements consist of:

Auto-implemented properties	automate the process of creating properties with trivial implementations
Implicitly typed local variables	permit the type of local variables to be inferred from the expressions used to initialize them
Implicitly typed arrays	a form of array creation and initialization that infers the element type of the array from an array initializer
Extension methods	which make it possible to extend existing types and constructed types with additional methods
Lambda expressions	an evolution of anonymous methods that concisely improves type inference and conversions to both delegate types and expression trees
Expression trees	permit lambda expressions to be represented as data (expression trees) instead of as code (delegates)
Object and collection initializers	you can use to conveniently specify values for one or more fields or properties for a newly created object, combining creation and initialization into a single step
Query expressions	provide a language-integrated syntax for queries that is similar to relational and hierarchical query languages such as SQL and XQuery
Anonymous types	are tuple types automatically inferred and created from object initializers

Use of Automatically Implemented Properties

Easy Initialization with Object and Collection Initializers

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

namespace NewLanguageFeatures
{
    public class Customer
    {
        public int CustomerID { get; private set; }
        public string Name { get; set; }
        public string City { get; set; }

        public Customer(int ID)
        {
            CustomerID = ID;
        }

        public override string ToString()
        {
            return Name + "\t" + City + "\t" + CustomerID;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Customer c = new Customer(1);
            c.Name = "Maria Anders";
            c.City = "Berlin";

            Console.WriteLine(c);
        }
    }
}
```

Implicitly Typed Local Variables and Implicitly Typed Arrays

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

namespace NewLanguageFeatures
{
    public class Customer
    {
        public int CustomerID { get; private set; }
        public string Name { get; set; }
        public string City { get; set; }

        public Customer(int ID)    { CustomerID = ID; }

        public override string ToString()
        {
            return Name + "\t" + City + "\t" + CustomerID;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            List<Customer> customers = CreateCustomers();

            Console.WriteLine("Customers:\n");
            foreach (Customer c in customers)
                Console.WriteLine(c);
        }

        static List<Customer> CreateCustomers()
        {
            return new List<Customer>
            {
                new Customer(1) { Name = "Maria Anders",      City = "Berlin"    },
                new Customer(2) { Name = "Laurence Lebihan",   City = "Marseille" },
                new Customer(3) { Name = "Elizabeth Brown",    City = "London"    },
                new Customer(4) { Name = "Ann Devon",          City = "London"    },
                new Customer(5) { Name = "Paolo Accorti",       City = "Torino"    },
                new Customer(6) { Name = "Fran Wilson",       City = "Portland"  },
                new Customer(7) { Name = "Simon Crowther",     City = "London"    },
                new Customer(8) { Name = "Liz Nixon",           City = "Portland"  }
            };
        }
    }
}
```

Extending Types with Extension Methods

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

namespace NewLanguageFeatures
{
    public class Customer
    {
        public int CustomerID { get; private set; }
        public string Name { get; set; }
        public string City { get; set; }

        public Customer(int ID)
        {
            CustomerID = ID;
        }

        public override string ToString()
        {
            return Name + "\t" + City + "\t" + CustomerID;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            List<Customer> customers = CreateCustomers();

            Console.WriteLine("Customers:\n");
            foreach (Customer c in customers)
                Console.WriteLine(c);
        }

        static List<Customer> CreateCustomers()
        {
            return new List<Customer>
            {
                new Customer(1) { Name = "Maria Anders",           City = "Berlin" },
                new Customer(2) { Name = "Laurence Lebihan",       City = "Marseille" },
                new Customer(3) { Name = "Elizabeth Brown",        City = "London" },
                new Customer(4) { Name = "Ann Devon",              City = "London" },
                new Customer(5) { Name = "Paolo Accorti",           City = "Torino" },
                new Customer(6) { Name = "Fran Wilson",           City = "Portland" },
                new Customer(7) { Name = "Simon Crowther",         City = "London" },
                new Customer(8) { Name = "Liz Nixon",               City = "Portland" }
            };
        }
    }
}
```

Working with Lambda Expressions

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

namespace NewLanguageFeatures
{
    public static class Extensions
    {
        public static List<T> Append<T>(this List<T> a, List<T> b)
        {
            var newList = new List<T>(a);
            newList.AddRange(b);
            return newList;
        }

        public static bool Compare(this Customer customer1, Customer customer2)
        {
            if (customer1.CustomerID == customer2.CustomerID &&
                customer1.Name == customer2.Name &&
                customer1.City == customer2.City)
            {
                return true;
            }
            return false;
        }
    }

    public class Customer
    {
        public int CustomerID { get; private set; }
        public string Name { get; set; }
        public string City { get; set; }

        public Customer(int ID)
        {
            CustomerID = ID;
        }

        public override string ToString()
        {
            return Name + "\t" + City + "\t" + CustomerID;
        }
    }
}
```



```
class Program
{
    static void Main(string[] args)
    {
        var customers = CreateCustomers();

        var addedCustomers = new List<Customer>
        {
            new Customer(9) { Name = "Paolo Accorti", City = "Torino" },
            new Customer(10) { Name = "Diego Roel", City = "Madrid" }
        };

        var updatedCustomers = customers.Append(addedCustomers);

        var newCustomer = new Customer(10)
        {
            Name = "Diego Roel",
            City = "Madrid"
        };

        foreach (var c in updatedCustomers)
        {
            if (newCustomer.Compare(c))
            {
                Console.WriteLine("The new customer was already in the list");
                return;
            }
        }
        Console.WriteLine("The new customer was not in the list");
    }

    static List<Customer> CreateCustomers()
    {
        return new List<Customer>
        {
            new Customer(1) { Name = "Maria Anders",           City = "Berlin" },
            new Customer(2) { Name = "Laurence Lebihan",        City = "Marseille" },
            new Customer(3) { Name = "Elizabeth Brown",          City = "London" },
            new Customer(4) { Name = "Ann Devon",                City = "London" },
            new Customer(5) { Name = "Paolo Accorti",             City = "Torino" },
            new Customer(6) { Name = "Fran Wilson",             City = "Portland" },
            new Customer(7) { Name = "Simon Crowther",           City = "London" },
            new Customer(8) { Name = "Liz Nixon",                 City = "Portland" }
        };
    }
}
```

Using Lambda Expressions to Create Expression Trees

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

namespace NewLanguageFeatures
{
    public static class Extensions
    {
        public static List<T> Append<T>(this List<T> a, List<T> b)
        {
            var newList = new List<T>(a);
            newList.AddRange(b);
            return newList;
        }

        public static bool Compare(this Customer customer1, Customer customer2)
        {
            if (customer1.CustomerID == customer2.CustomerID &&
                customer1.Name == customer2.Name &&
                customer1.City == customer2.City)
            {
                return true;
            }
            return false;
        }
    }

    public class Customer
    {
        public int CustomerID { get; private set; }
        public string Name { get; set; }
        public string City { get; set; }

        public Customer(int ID)
        {
            CustomerID = ID;
        }

        public override string ToString()
        {
            return Name + "\t" + City + "\t" + CustomerID;
        }
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        var customers = CreateCustomers();

        foreach (var c in FindCustomersByCity(customers, "London"))
            Console.WriteLine(c);
    }

    public static List<Customer> FindCustomersByCity(
        List<Customer> customers,
        string city)
    {
        return customers.FindAll(c => c.City == city);
    }

    static List<Customer> CreateCustomers()
    {
        return new List<Customer>
        {
            new Customer(1) { Name = "Maria Anders",           City = "Berlin" },
            new Customer(2) { Name = "Laurence Lebihan",        City = "Marseille" },
            new Customer(3) { Name = "Elizabeth Brown",         City = "London" },
            new Customer(4) { Name = "Ann Devon",               City = "London" },
            new Customer(5) { Name = "Paolo Accorti",           City = "Torino" },
            new Customer(6) { Name = "Fran Wilson",            City = "Portland" },
            new Customer(7) { Name = "Simon Crowther",          City = "London" },
            new Customer(8) { Name = "Liz Nixon",               City = "Portland" }
        };
    }
}
```

Understanding Queries and Query Expressions

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Linq.Expressions;

namespace NewLanguageFeatures
{
    public static class Extensions
    {
        public static List<T> Append<T>(this List<T> a, List<T> b)
        {
            var newList = new List<T>(a);
            newList.AddRange(b);
            return newList;
        }

        public static bool Compare(this Customer customer1, Customer customer2)
        {
            if (customer1.CustomerID == customer2.CustomerID &&
                customer1.Name == customer2.Name &&
                customer1.City == customer2.City)
            {
                return true;
            }
            return false;
        }
    }

    public class Customer
    {
        public int CustomerID { get; private set; }
        public string Name { get; set; }
        public string City { get; set; }

        public Customer(int ID)
        {
            CustomerID = ID;
        }

        public override string ToString()
        {
            return Name + "\t" + City + "\t" + CustomerID;
        }
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        Func<int, int> addOne = n => n + 1;
        Console.WriteLine("Result: {0}", addOne(5));

        Expression<Func<int, int>> addOneExpression = n => n + 1;

        var addOneFunc = addOneExpression.Compile();
        Console.WriteLine("Result: {0}", addOneFunc(5));
    }

    public static List<Customer> FindCustomersByCity(
        List<Customer> customers,
        string city)
    {
        return customers.FindAll(c => c.City == city);
    }

    static List<Customer> CreateCustomers()
    {
        return new List<Customer>
        {
            new Customer(1) { Name = "Maria Anders", City = "Berlin"},
            new Customer(2) { Name = "Laurence Lebihan", City = "Marseille" },
            new Customer(3) { Name = "Elizabeth Brown", City = "London" },
            new Customer(4) { Name = "Ann Devon", City = "London" },
            new Customer(5) { Name = "Paolo Accorti", City = "Torino" },
            new Customer(6) { Name = "Fran Wilson", City = "Portland" },
            new Customer(7) { Name = "Simon Crowther", City = "London" },
            new Customer(8) { Name = "Liz Nixon", City = "Portland" }
        };
    }
}
```

Anonymous Types and Advanced Query Creation

```
using System;
using System.Collections.Generic;
using System.Linq; using System.Text;
using System.Linq.Expressions;

namespace NewLanguageFeatures
{
    public static class Extensions
    {
        public static List<T> Append<T>(this List<T> a, List<T> b)
        {
            var newList = new List<T>(a);
            newList.AddRange(b);
            return newList;
        }

        public static bool Compare(this Customer customer1, Customer customer2)
        {
            if (customer1.CustomerID == customer2.CustomerID &&
                customer1.Name == customer2.Name &&
                customer1.City == customer2.City)
            { return true; }
            return false;
        }
    }

    public class Store
    {
        public string Name { get; set; }
        public string City { get; set; }
        public override string ToString()
        {
            return Name + "\t" + City;
        }
    }

    public class Customer
    {
        public int CustomerID { get; private set; }
        public string Name { get; set; }
        public string City { get; set; }

        public Customer(int ID) { CustomerID = ID; }

        public override string ToString()
        { return Name + "\t" + City + "\t" + CustomerID; }
    }
}
```

```
class Program
{
    static void Query()
    {
        var stores = CreateStores();
        var numLondon = stores.Count(s => s.City == "London");
        Console.WriteLine("There are {0} stores in London. ", numLondon);
    }

    static void Main(string[] args)
    {
        Query();
    }

    public static List<Customer> FindCustomersByCity(
        List<Customer> customers,
        string city)
    {
        return customers.FindAll(c => c.City == city);
    }

    static List<Store> CreateStores()
    {
        return new List<Store>
        {
            new Store { Name = "Jim's Hardware",    City = "Berlin"},
            new Store { Name = "John's Books",      City = "London"},
            new Store { Name = "Lisa's Flowers",    City = "Torino"},
            new Store { Name = "Dana's Hardware",    City = "London"},
            new Store { Name = "Tim's Pets",        City = "Portland"},
            new Store { Name = "Scott's Books",      City = "London"},
            new Store { Name = "Paula's Cafe",      City = "Marseille"}
        };
    }

    static List<Customer> CreateCustomers()
    {
        return new List<Customer>
        {
            new Customer(1) { Name = "Maria Anders", City = "Berlin"},
            new Customer(2) { Name = "Laurence Lebihan", City = "Marseille"},
            new Customer(3) { Name = "Elizabeth Brown", City = "London"},
            new Customer(4) { Name = "Ann Devon", City = "London"},
            new Customer(5) { Name = "Paolo Accorti", City = "Torino"},
            new Customer(6) { Name = "Fran Wilson", City = "Portland"},
            new Customer(7) { Name = "Simon Crowther", City = "London"},
            new Customer(8) { Name = "Liz Nixon", City = "Portland" }
        };
    }
}
```