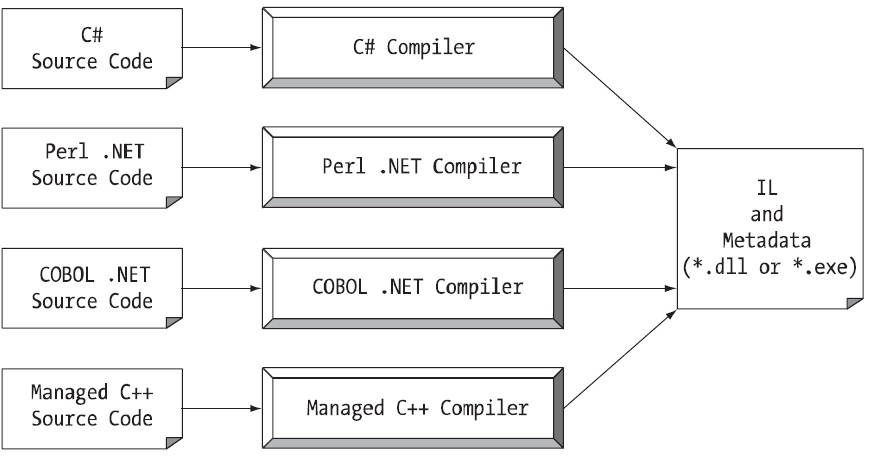
//////////////////////////////////////



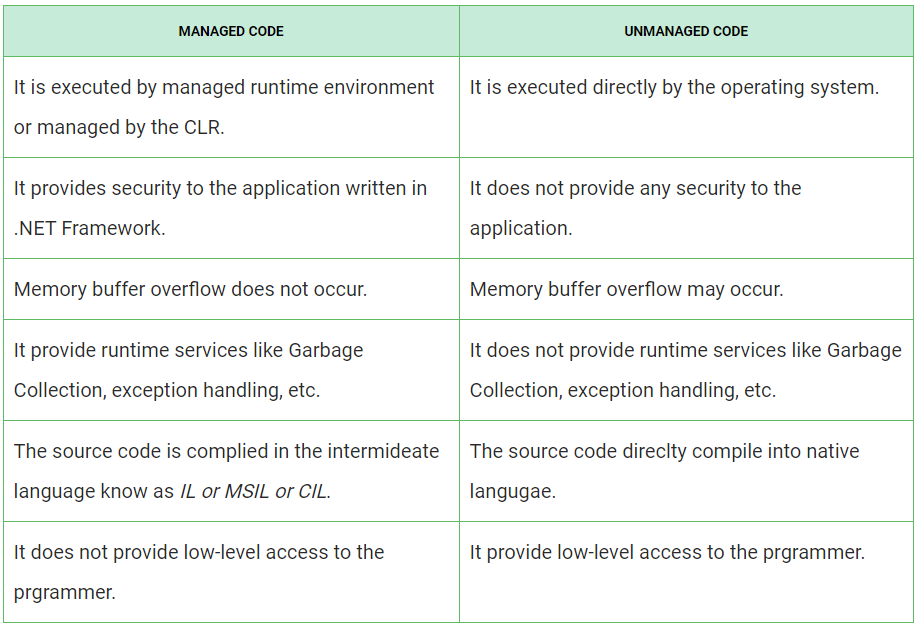
///////////////////////////////////////////////////////////

**YÖNETİLEN (MANAGED) VE YÖNETİLMEYEN (UNMANAGED) KOD**

**Yönetilen kod**, .NET Framework'te CLR (Ortak Dil Çalışma Zamanı - Common Language Runtime) tarafından yönetilen koddur.

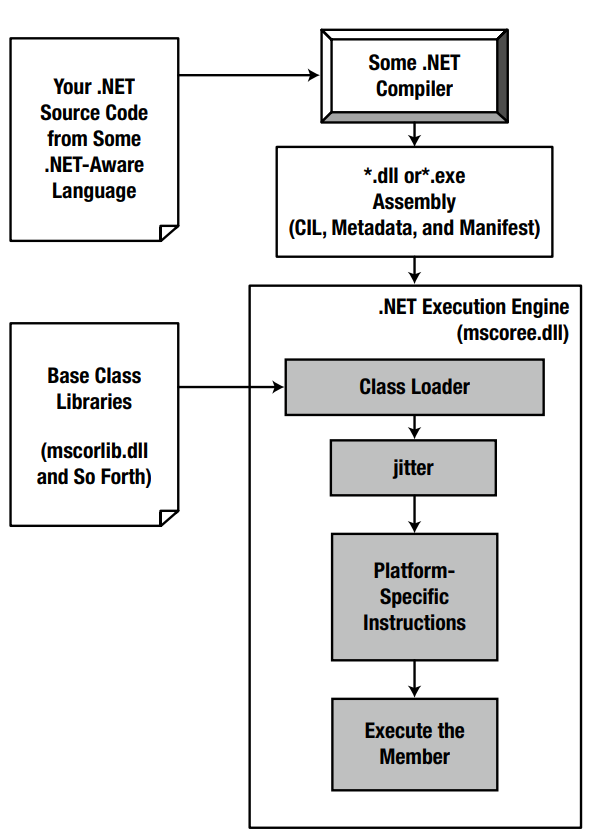
**Yönetilmeyen kod** ise doğrudan işletim sistemi tarafından yürütülen koddur. Yönetilen kod ile Yönetilmeyen kod arasındaki bazı önemli farklar aşağıdadır.

|  |  |
| --- | --- |
| MANAGED CODE | UNMANAGED CODE |
| Yönetilen çalışma zamanı tarafından yürütülür veya CLR tarafından yönetilir. | Doğrudan işletim sistemi tarafından yürütülür. |
| .NET framework’de yazılan uygulamaya güvenlik sağlar. | Uygulamaya herhangi bir güvenlik sağlamaz. |
| Bellek arabelleği taşması oluşmuyor. | Bellek arabelleği taşması oluşabilir. |
| Çöp Toplama, istisna işleme gibi çalışma zamanı hizmetleri sağlar. | Çöp Toplama, özel durum işleme gibi çalışma zamanı hizmetleri sağlamaz. |
| Kaynak kodu, IL veya MSIL veya CIL olarak bilinen intermediate dilinde derlenir. | Kaynak kodu doğrudan ana dilde derlenir. |
| Programlayıcıya düşük seviyeli erişim sağlamaz. | Programlayıcıya düşük seviyeli erişim sağlar. |



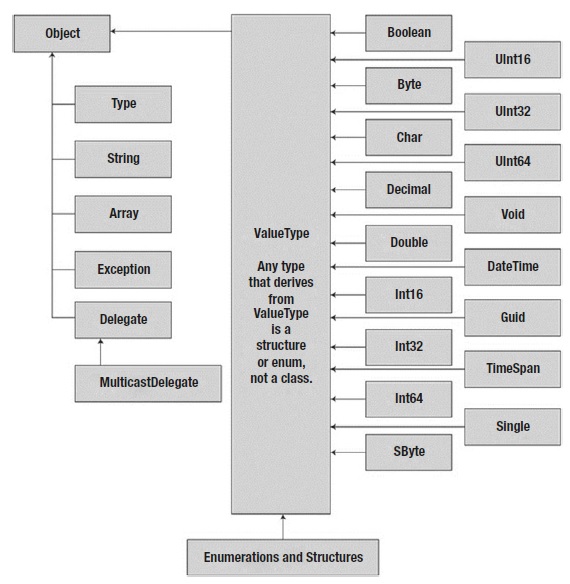
/////////////////////////////////////////////

**MCOREE.DLL ÇALIŞMA MANTIĞI**



///////////////////////////////////////////////

SİSTEM VERİ TİPLERİ



/////////////////////////////////////////////////

**C# Coding Standards and Naming Conventions**

| **Object Name** | **Notation** | **Length** | **Plural** | **Prefix** | **Suffix** | **Abbreviation** | **Char Mask** | **Underscores** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Class name | PascalCase | 128 | No | No | Yes | No | [A-z][0-9] | No |
| Constructor name | PascalCase | 128 | No | No | Yes | No | [A-z][0-9] | No |
| Method name | PascalCase | 128 | Yes | No | No | No | [A-z][0-9] | No |
| **Method arguments** | **camelCase** | 128 | Yes | No | No | Yes | [A-z][0-9] | No |
| **Local variables** | camelCase | 50 | Yes | No | No | Yes | [A-z][0-9] | No |
| Constants name | PascalCase | 50 | No | No | No | No | [A-z][0-9] | No |
| **Field name** | **camelCase** | 50 | Yes | No | No | Yes | [A-z][0-9] | **Yes** |
| Properties name | PascalCase | 50 | Yes | No | No | Yes | [A-z][0-9] | No |
| Delegate name | PascalCase | 128 | No | No | Yes | Yes | [A-z] | No |
| Enum type name | PascalCase | 128 | Yes | No | No | No | [A-z] | No |

**1. Do use PascalCasing for class names and method names:**

public class ClientActivity

{

public void ClearStatistics()

{

//...

}

public void CalculateStatistics()

{

//...

}

}

***Why: consistent with the Microsoft's .NET Framework and easy to read.***

**2. Do use camelCasing for method arguments and local variables:**

public class UserLog

{

public void Add(LogEvent logEvent)

{

int itemCount = logEvent.Items.Count;

// ...

}

}

***Why: consistent with the Microsoft's .NET Framework and easy to read.***

**3. Do not use Hungarian notation or any other type identification in identifiers**

// Correct

int counter;

string name;

// Avoid

int iCounter;

string strName;

***Why: consistent with the Microsoft's .NET Framework and Visual Studio IDE makes determining types very easy (via tooltips). In general you want to avoid type indicators in any identifier.***

**4. Do not use Screaming Caps for constants or readonly variables:**

// Correct

public const string ShippingType = "DropShip";

// Avoid

public const string SHIPPINGTYPE = "DropShip";

***Why: consistent with the Microsoft's .NET Framework. Caps grab too much attention.***

**5. Use meaningful names for variables. The following example uses seattleCustomers for customers who are located in Seattle:**

var seattleCustomers = from customer in customers

where customer.City == "Seattle"

select customer.Name;

***Why: consistent with the Microsoft's .NET Framework and easy to read.***

**6. Avoid using Abbreviations. Exceptions: abbreviations commonly used as names, such as Id, Xml, Ftp, Uri.**

// Correct

UserGroup userGroup;

Assignment employeeAssignment;

// Avoid

UserGroup usrGrp;

Assignment empAssignment;

// Exceptions

CustomerId customerId;

XmlDocument xmlDocument;

FtpHelper ftpHelper;

UriPart uriPart;

***Why: consistent with the Microsoft's .NET Framework and prevents inconsistent abbreviations.***

**7. Do use PascalCasing for abbreviations 3 characters or more (2 chars are both uppercase):**

HtmlHelper htmlHelper;

FtpTransfer ftpTransfer;

UIControl uiControl;

***Why: consistent with the Microsoft's .NET Framework. Caps would grab visually too much attention.***

**8. Do not use Underscores in identifiers. Exception: you can prefix private fields with an underscore:**

// Correct

public DateTime clientAppointment;

public TimeSpan timeLeft;

// Avoid

public DateTime client\_Appointment;

public TimeSpan time\_Left;

// Exception (Class field)

private DateTime \_registrationDate;

***Why: consistent with the Microsoft's .NET Framework and makes code more natural to read (without 'slur'). Also avoids underline stress (inability to see underline).***

**9. Do use predefined type names (C# aliases) like int, float, string for local, parameter and member declarations. Do use .NET Framework names like Int32, Single, String when accessing the type's static members like Int32.TryParse or String.Join.**

// Correct

string firstName;

int lastIndex;

bool isSaved;

string commaSeparatedNames = String.Join(", ", names);

int index = Int32.Parse(input);

// Avoid

String firstName;

Int32 lastIndex;

Boolean isSaved;

string commaSeparatedNames = string.Join(", ", names);

int index = int.Parse(input);

***Why: consistent with the Microsoft's .NET Framework and makes code more natural to read.***

**10. Do use implicit type var for local variable declarations. Exception: primitive types (int, string, double, etc) use predefined names.**

var stream = File.Create(path);

var customers = new Dictionary();

// Exceptions

int index = 100;

string timeSheet;

bool isCompleted;

***Why: removes clutter, particularly with complex generic types. Type is easily detected with Visual Studio tooltips.***

**11. Do use noun or noun phrases to name a class.**

public class Employee

{

}

public class BusinessLocation

{

}

public class DocumentCollection

{

}

***Why: consistent with the Microsoft's .NET Framework and easy to remember.***

**12. Do prefix interfaces with the letter I. Interface names are noun (phrases) or adjectives.**

public interface IShape

{

}

public interface IShapeCollection

{

}

public interface IGroupable

{

}

***Why: consistent with the Microsoft's .NET Framework.***

**13. Do name source files according to their main classes. Exception: file names with partial classes reflect their source or purpose, e.g. designer, generated, etc.**

// Located in Task.cs

public partial class Task

{

}

// Located in Task.generated.cs

public partial class Task

{

}

***Why: consistent with the Microsoft practices. Files are alphabetically sorted and partial classes remain adjacent.***

**14. Do organize namespaces with a clearly defined structure:**

// Examples

namespace Company.Product.Module.SubModule

{

}

namespace Product.Module.Component

{

}

namespace Product.Layer.Module.Group

{

}

***Why: consistent with the Microsoft's .NET Framework. Maintains good organization of your code base.***

**15. Do vertically align curly brackets:**

// Correct

class Program

{

static void Main(string[] args)

{

//...

}

}

***Why: Microsoft has a different standard, but developers have overwhelmingly preferred vertically aligned brackets.***

**16. Do declare all member variables at the top of a class, with static variables at the very top.**

// Correct

public class Account

{

public static string BankName;

public static decimal Reserves;

public string Number { get; set; }

public DateTime DateOpened { get; set; }

public DateTime DateClosed { get; set; }

public decimal Balance { get; set; }

// Constructor

public Account()

{

// ...

}

}

***Why: generally accepted practice that prevents the need to hunt for variable declarations.***

**17. Do use singular names for enums. Exception: bit field enums.**

// Correct

public enum Color

{

Red,

Green,

Blue,

Yellow,

Magenta,

Cyan

}

// Exception

[Flags]

public enum Dockings

{

None = 0,

Top = 1,

Right = 2,

Bottom = 4,

Left = 8

}

***Why: consistent with the Microsoft's .NET Framework and makes the code more natural to read. Plural flags because enum can hold multiple values (using bitwise 'OR').***

**18. Do not explicitly specify a type of an enum or values of enums (except bit fields):**

// Don't

public enum Direction : long

{

North = 1,

East = 2,

South = 3,

West = 4

}

// Correct

public enum Direction

{

North,

East,

South,

West

}

***Why: can create confusion when relying on actual types and values.***

**19. Do not use an "Enum" suffix in enum type names:**

// Don't

public enum CoinEnum

{

Penny,

Nickel,

Dime,

Quarter,

Dollar

}

// Correct

public enum Coin

{

Penny,

Nickel,

Dime,

Quarter,

Dollar

}

***Why: consistent with the Microsoft's .NET Framework and consistent with prior rule of no type indicators in identifiers.***

**20. Do not use "Flag" or "Flags" suffixes in enum type names:**

// Don't

[Flags]

public enum DockingsFlags

{

None = 0,

Top = 1,

Right = 2,

Bottom = 4,

Left = 8

}

// Correct

[Flags]

public enum Dockings

{

None = 0,

Top = 1,

Right = 2,

Bottom = 4,

Left = 8

}

***Why: consistent with the Microsoft's .NET Framework and consistent with prior rule of no type indicators in identifiers.***

**21. Do use suffix EventArgs at creation of the new classes comprising the information on event:**

// Correct

public class BarcodeReadEventArgs : System.EventArgs

{

}

***Why: consistent with the Microsoft's .NET Framework and easy to read.***

**22. Do name event handlers (delegates used as types of events) with the "EventHandler" suffix, as shown in the following example:**

public delegate void ReadBarcodeEventHandler(object sender, ReadBarcodeEventArgs e);

***Why: consistent with the Microsoft's .NET Framework and easy to read.***

**23. Do not create names of parameters in methods (or constructors) which differ only by the register:**

// Avoid

private void MyFunction(string name, string Name)

{

//...

}

***Why: consistent with the Microsoft's .NET Framework and easy to read, and also excludes possibility of occurrence of conflict situations.***

**24. DO use two parameters named sender and e in event handlers. The sender parameter represents the object that raised the event. The sender parameter is typically of type object, even if it is possible to employ a more specific type.**

public void ReadBarcodeEventHandler(object sender, ReadBarcodeEventArgs e)

{

//...

}

***Why: consistent with the Microsoft's .NET Framework***

***Why: consistent with the Microsoft's .NET Framework and consistent with prior rule of no type indicators in identifiers.***

**25. Do use suffix Exception at creation of the new classes comprising the information on exception:**

// Correct

public class BarcodeReadException : System.Exception

{

}

***Why: consistent with the Microsoft's .NET Framework and easy to read.***

**26. Do use suffix Any, Is, Have or similar keywords for boolean identifier :**

// Correct

public static bool IsNullOrEmpty(string value) {

return (value == null || value.Length == 0);

}

***Why: consistent with the Microsoft's .NET Framework and easy to read.***

**Offical Reference**

1. [MSDN General Naming Conventions](http://msdn.microsoft.com/en-us/library/ms229045(v=vs.110).aspx)
2. [DoFactory C# Coding Standards and Naming Conventions](http://www.dofactory.com/reference/csharp-coding-standards)
3. [MSDN Naming Guidelines](http://msdn.microsoft.com/en-us/library/xzf533w0%28v=vs.71%29.aspx)
4. [MSDN Framework Design Guidelines](http://msdn.microsoft.com/en-us/library/ms229042.aspx)

///////////////////////////////////////////

**CHECKED KEYWORD**

Bir ifade ve ifade bloklarını **checked** anahtar sözcüğü kapsamına alırsak, toplama, çıkarma, çarpma ve bölme gibi matematiksel işlemler yapılırken oluşabilecek aşağı taşma (underflow) veya yukarı taşma (overflow) durumlarında C# derleyicisi ek CIL talimatı (komutu) yayınlar. Bir taşma meydana geldiğinde çalışma zamanı istisnası,hatası,(runtime exception) alırız: Sysytem.OverflowException.

İki bayt değerini toplamak istediğimizde, aşağıdaki örnekteki gibi;

static void ProcessBytes()  
{  
byte b1 = 100;  
byte b2 = 250;  
byte toplam = (byte)Add(b1, b2);  
// toplam değişkeni değeri 350. Fakat, sonuç 94  
Console.WriteLine("Toplam = {0}", toplam);  
}

Yukarıdaki örnekte görüldüğü gibi toplam değişkenin değeri 350 olması gerekir. Fakat **System.Byte** ( byte )maksimum 256 (0 ve 255 aralığında değer alır) değerini tutar. 350-256=94 sonuç. Kodu aşağıdaki gibi düzenlersek taşma durumlarında runtime exception hatası alırız.

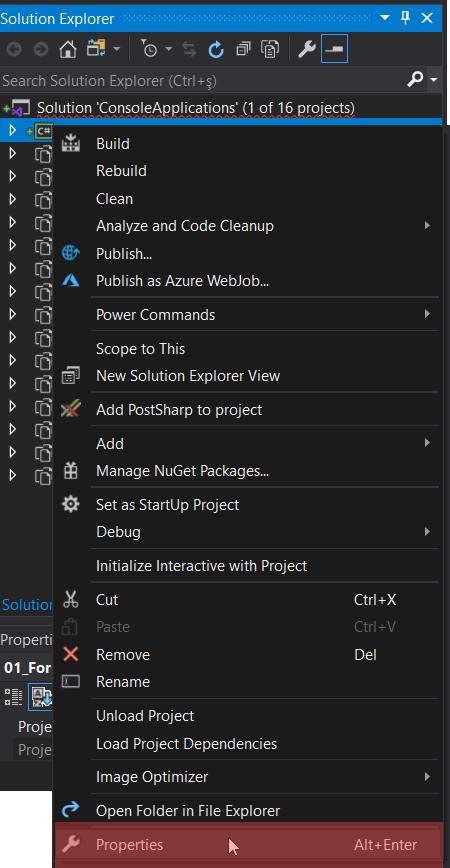
static void ProcessBytes()  
{  
byte b1 = 100;  
byte b2 = 250;  
// This time, tell the compiler to add CIL code  
// to throw an exception if overflow/underflow  
// takes place.  
try  
{  
byte sum = checked((byte)Add(b1, b2));  
Console.WriteLine("sum = {0}", sum);  
}  
catch (OverflowException ex)  
{  
Console.WriteLine(ex.Message);  
}  
}

Eğer checked anahtar kelimesini ifade bloklarına uygulamak istersek;

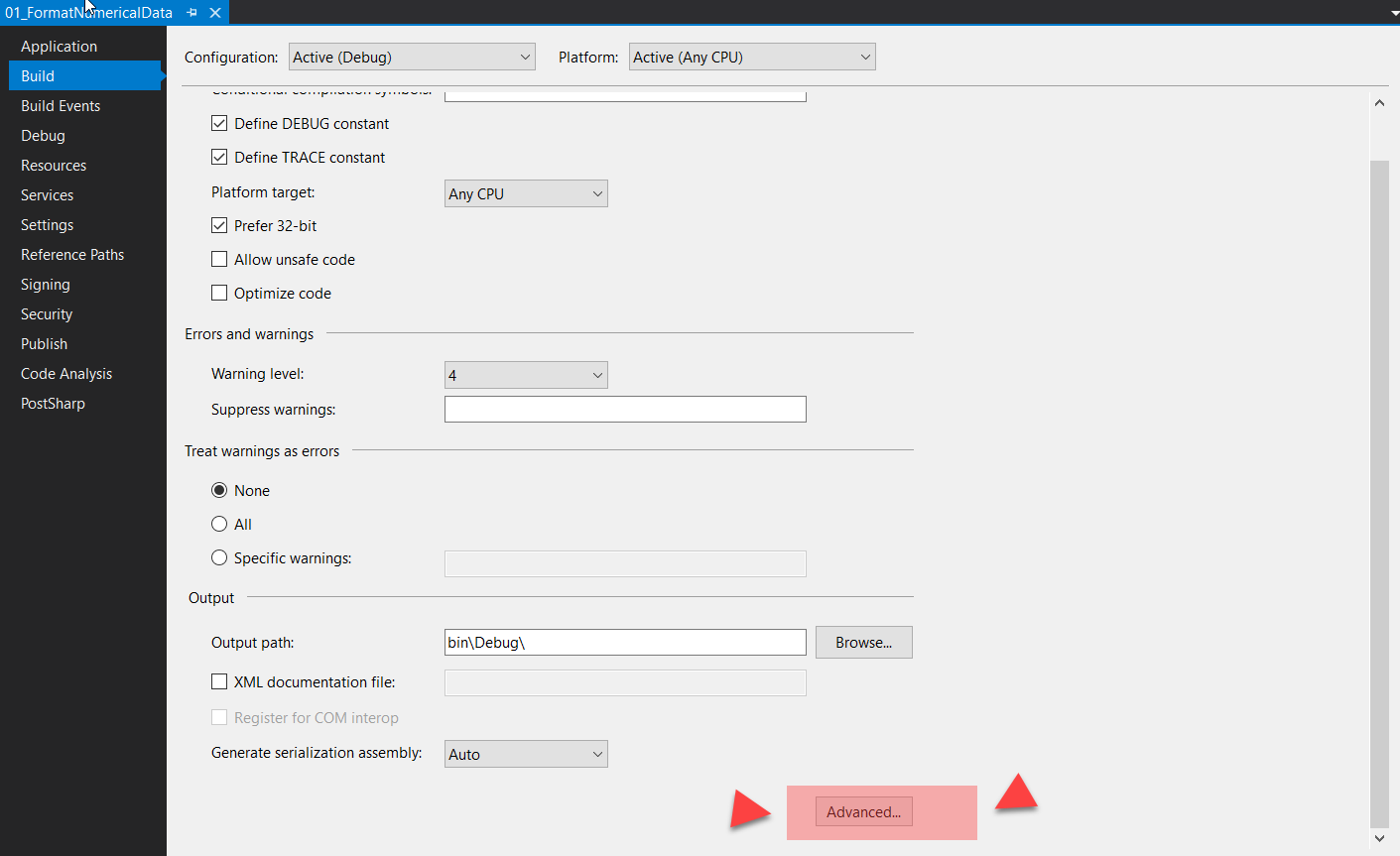
try  
{  
checked  
{  
byte sum = (byte)Add(b1, b2);  
Console.WriteLine("sum = {0}", sum);  
}  
}  
catch (OverflowException ex)  
{  
Console.WriteLine(ex.Message);  
}

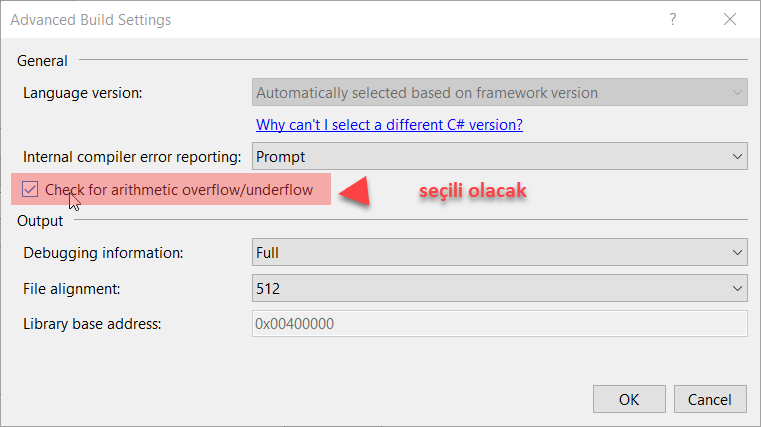
**PROJE BAZINDA OVERFLOW KONTROLÜ**

Proje ismine sağ tıklayıp, **Properties** seçilir.



Açılan pencereden “ **Advanced ”** butonuna tıklanır.





Not: Proje başında bu ayarlamanın yapılması daha iyi olur.

///////////////////////////////////////

**Man() Metodu Çeşitleri**

**// int return type, array of strings as the parameter.**static int Main(string[] args)  
{  
**// Must return a value before exiting!**return 0;  
}  
**// No return type, no parameters.**  
static void Main()  
{}  
**// int return type, no parameters.**  
static int Main()  
{  
**// Must return a value before exiting!**return 0;  
}

///////////////////////////////////////////////////

**BİLGİSAYARIN ÇEVRESEL ÖZELİKLERİNİ BULMA (Enviroment Sınıfı)**

namespace \_24\_EnvironmentProperties

{

class Program

{

static void Main(string[] args)

{

string str;

string nl = Environment.NewLine;

//

Console.ForegroundColor = ConsoleColor.DarkGreen;

Console.WriteLine();

Console.WriteLine("--\*\* Environment Members - Üyeleri \*\*\*\*\*\*--");

Console.ForegroundColor = ConsoleColor.DarkYellow;

//  Invoke this sample with an arbitrary set of command line arguments.

Console.WriteLine("CommandLine: {0}", Environment.CommandLine);

string[] arguments = Environment.GetCommandLineArgs();

Console.WriteLine("GetCommandLineArgs: {0}", String.Join(", ", arguments));

//  <-- Keep this information secure! -->

Console.WriteLine("CurrentDirectory: {0}", Environment.CurrentDirectory);

Console.WriteLine("ExitCode: {0}", Environment.ExitCode);

Console.WriteLine("HasShutdownStarted: {0}", Environment.HasShutdownStarted);

//  <-- Keep this information secure! -->

Console.WriteLine("MachineName: {0}", Environment.MachineName);

Console.WriteLine("NewLine: {0}  first line{0}  second line{0}  third line",

Environment.NewLine);

Console.WriteLine("OSVersion: {0}", Environment.OSVersion.ToString());

Console.WriteLine("StackTrace: '{0}'", Environment.StackTrace);

//  <-- Keep this information secure! -->

Console.WriteLine("SystemDirectory: {0}", Environment.SystemDirectory);

Console.WriteLine("TickCount: {0}", Environment.TickCount);

//  <-- Keep this information secure! -->

Console.WriteLine("UserDomainName: {0}", Environment.UserDomainName);

Console.WriteLine("UserInteractive: {0}", Environment.UserInteractive);

//  <-- Keep this information secure! -->

Console.WriteLine("UserName: {0}", Environment.UserName);

Console.WriteLine("Version: {0}", Environment.Version.ToString());

Console.WriteLine("WorkingSet: {0}", Environment.WorkingSet);

//  No example for Exit(exitCode) because doing so would terminate this example.

//  <-- Keep this information secure! -->

string query = "My system drive is %SystemDrive% and my system root is %SystemRoot%";

str = Environment.ExpandEnvironmentVariables(query);

Console.WriteLine("ExpandEnvironmentVariables: {0}  {1}", nl, str);

Console.WriteLine("GetEnvironmentVariable: {0}  My temporary directory is {1}.", nl,

 Environment.GetEnvironmentVariable("TEMP"));

Console.ForegroundColor = ConsoleColor.DarkBlue;

Console.WriteLine("GetEnvironmentVariables: ");

IDictionary environmentVariables = Environment.GetEnvironmentVariables();

foreach (DictionaryEntry de in environmentVariables)

{

Console.WriteLine("  {0} = {1}", de.Key, de.Value);

}

Console.WriteLine("GetFolderPath: {0}",

 Environment.GetFolderPath(Environment.SpecialFolder.System));

Console.ForegroundColor = ConsoleColor.DarkGray;

string[] drives = Environment.GetLogicalDrives();

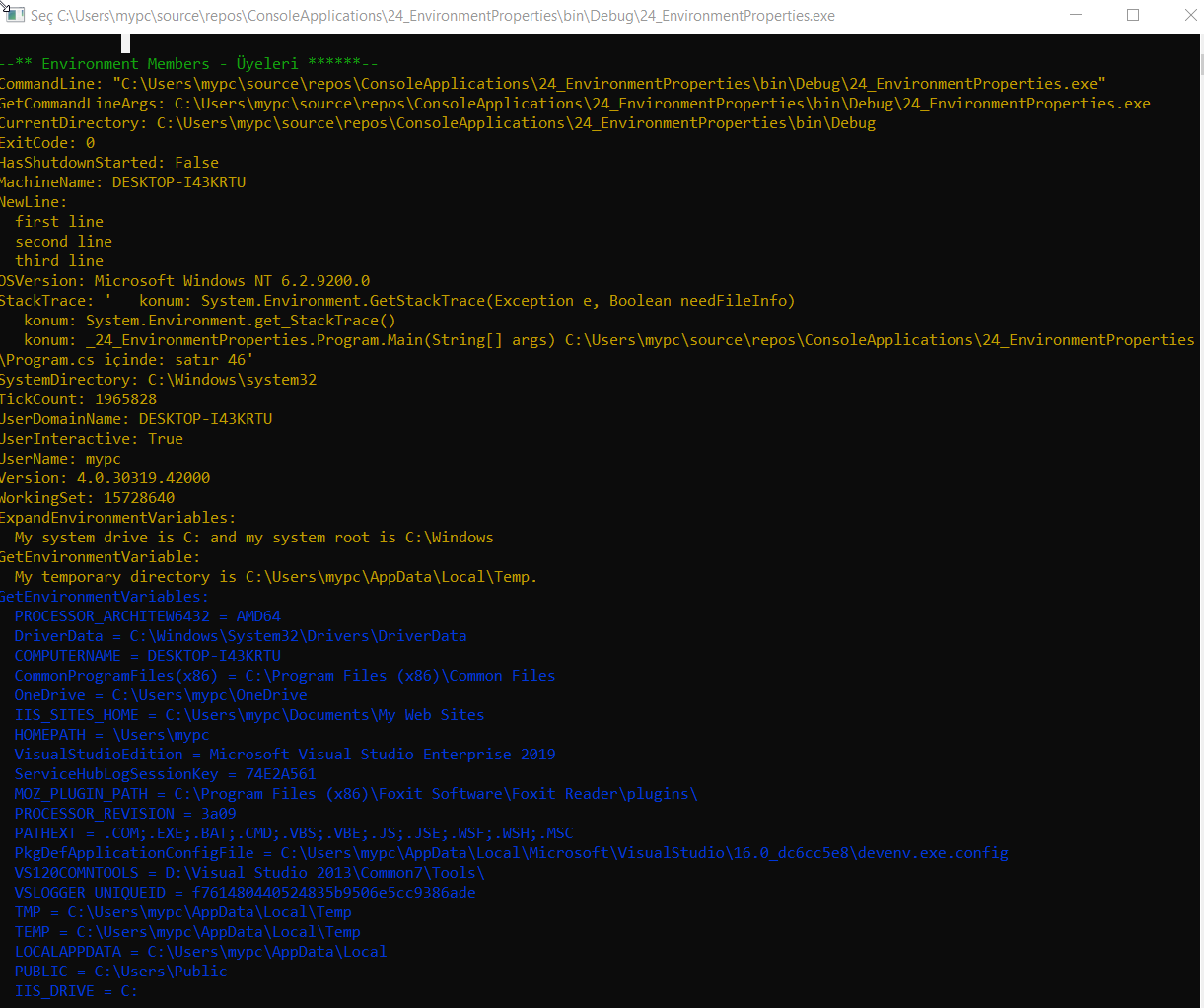
Console.WriteLine("GetLogicalDrives: {0}", String.Join(", ", drives));

Console.ReadKey();

}

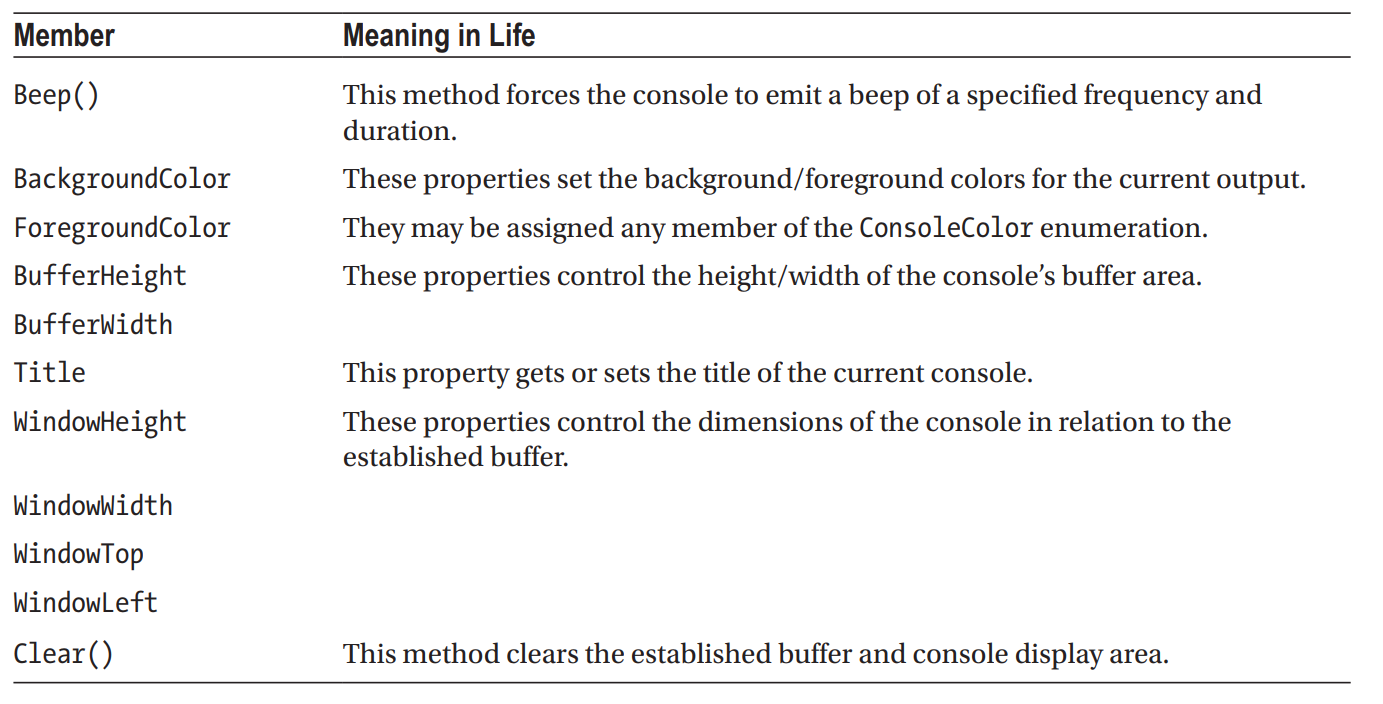
}

}



////////////////////////////////////////////////////////////////////////

**SYSTEM.COSOLE SINIFININ ÜYELERİ (** *Members of System.Console* **)**



//////////////////////////////////////////////////////////////

**C# params Tanımlayıcısı (params modifier)**

C# programlama dilinde yer alan ve metotlara değişken sayıda parametre göndermeye imkan veren **params** anahtar kelimesi nedir ve kullanımı ile ilgili bilgiler yer alıyor.

**C# params nedir?**

Metotların değişken sayıda parametre almasına imkan veren bir anahtar kelimedir.

Metotlar oluşturulurken kullanılacak parametre sayısı önceden belirlenir.

Örneğin; İki adet sayı toplanmak istediğinde aşağıdaki gibi bir metot yazılabilir.

Kopyalaint Topla(int sayi1, int sayi2);

Toplanacak sayı adedi üç adet olduğunda aşağıdaki gibi bir metot yazılabilir.

Kopyalaint Topla(int sayi1, int sayi2, int sayi3);

Bazen parametre sayısı tam belli olmayabilir.

Bu durumda her eklenen parametre için yeni bir metot tanımlanabilir.

Ancak bu etkili bir çözüm olmayacaktır.

Parametre sayısı belli olmayan metot tanımlamak için **params** özelliği kullanılabilir.

Kopyalaint Topla(params int[] sayilar);

Artık **Topla** metodu **int** türünden değişken sayıda parametre alabilir.

Değişken türü belli olmayan durumlarda C# içerisindeki her şeyin **object** türünden türediği özelliği kullanılabilir.

void Yazdir(params object[] gelenDegerler);

Gönderilen parametre tür bilgisi **object** sınıfında yer alan **GetType** metodu ile alınabilir.

using System;

class Program {

static void Main() {

DegerAl(10);

DegerAl(10, 30f);

DegerAl(10, 30f, 40.3);

DegerAl(10, 30f, 40.3, "Yusuf");

DegerAl(10, 30f, 40.3, "Yusuf", 3M);

}

static void DegerAl(params object[] gelenDegerler)

{

foreach (object eleman in gelenDegerler)

{

Console.WriteLine("Değer: {0} - Tipi: {1}", eleman, eleman.GetType().Name);

}

Console.WriteLine();

} }

Aşağıda **params** özelliği ile gönderilen sayılardan en büyüğünü bulan örnek metot yer almaktadır.

using System;

class Program {

static void Main() {

int sonuc = EnBuyukSayiyiBul(10, 5, 7, 99, 1531, 22, 331313); Console.WriteLine(sonuc);

}

static int EnBuyukSayiyiBul(params int[] sayilar) {

int enBuyukSayi = -1; if(sayilar.Length > 0)

{ enBuyukSayi = sayilar[0];

}

foreach (int sayi in sayilar)

{

if(enBuyukSayi < sayi) { enBuyukSayi = sayi; }

}

return enBuyukSayi;

} }

C# aşırı yüklenmiş metotları seçerken önceliği normal metotlara verir.

using System;

class Program {

static void Main() {

Topla(1, 2);

}

static void Topla(int sayi1, int sayi2) {

Console.WriteLine("Ben normal metodum.");

}

static void Topla(params int[] gelenSayilar) {

Console.WriteLine("Ben params kullanılmış metodum.");

} }

Bu özellik C# içerisinde yazı formatlama için kullanılan **String.Format** ve **Console.WriteLine** metotlarında da kullanılmaktadır.

using System;

class Program {

static void Main() {

string Ad = "Yusuf";

string Soyad = "SEZER";

Console.WriteLine("Adınız: {0}, Soyadınız: {1}", Ad, Soyad);

string sonuc = String.Format("Adınız: {0}, Soyadınız: {1}", Ad, Soyad);

} }

namespace \_23\_ParamsParameter\_2

{

class Program

{

static double CalculateAverage(params int[] values)

{

ForegroundColor = DarkBlue;

Console.WriteLine("You sent me {0} integers.", values.Length);

double sum = 0;

if (values.Length == 0)

return sum;

for (int i = 0; i < values.Length; i++)

sum += values[i];

return (sum / values.Length);

}

static void Main(string[] args)

{

int[] myInts = new int[20];

Random random = new Random();

for (int i = 0; i < 20; i++)

{

myInts[i] = random.Next(100,1000);

}

WriteLine();

ForegroundColor = DarkGreen;

double average = CalculateAverage(myInts);

WriteLine(" Average of data is: {0}", average);

ReadKey();

}

}

}

namespace \_22\_ParamsParameter

{

class Program

{

// Return average of "some number" of doubles.

static double CalculateAverage(params double[] values)

{

Console.WriteLine("You sent me {0} doubles.", values.Length);

double sum = 0;

if (values.Length == 0)

return sum;

for (int i = 0; i < values.Length; i++)

sum += values[i];

return (sum / values.Length);

}

static void Main(string[] args)

{

Console.ForegroundColor = ConsoleColor.DarkGreen;

double average;

average = CalculateAverage(4.0, 3.2, 5.7, 64.22, 87.2);

Console.WriteLine("Average of data is: {0}", average);

Console.ReadKey();

}

}

}

/////////////////////////////////////////////////////////

YEREL FONKSİYONLAR (Local Functions)

Aslında metod içerisinde tanımlanan metotlardır. Yani bir metod sadece tanımlandığı metot içerisinde çağrılabilir demektir. Özel (custom) tanımlı öz yinelemeli metodlar (iterator method) ve asenktron metodlarda (asynchronous methods) için bir özelliktir. Aşağıdaki örnekte Add() metodu sadece AddWrapper() metodu içerisinde çağrılabilmektedir.

static int AddWrapper(int x, int y)  
{  
//Do some validation here  
return Add();  
int Add()  
{  
return x + y;  
}  
}

///////////////////////////////////////////////////////////////

STRUCT

Struct türleri, uygulamanızdaki matematiksel, geometrik ve diğer “atomik” varlıkları modellemek için çok uygundur. Bir struct (enum gibi) kullanıcı tanımlı bir türdür; ancak, yapılar sadece isim-değer çiftlerinin bir toplamı değildir. Bunun yerine, yapılar herhangi bir sayıda veri alanı içerebilen türlerdir ve bu alanlarda çalışan üyelerdir. Sınıflardan farkı kalıtım özellikleri yoktur. Sınıflarla benzer özellikleri ise; alanlara (field) sahiptirler, metodları ve yapıcı (kurucu=constructor) metodları vardır.

struct Point  
{  
**// Fields of the structure.**  
public int X;  
public int Y;  
**// Add 1 to the (X, Y) position.**  
public void Increment()  
{  
X++; Y++;  
}  
**// Subtract 1 from the (X, Y) position.**public void **Decrement()**  
{  
X--; Y--;  
}  
**// Display the current position.**  
public void Display()  
{  
Console.WriteLine("X = {0}, Y = {1}", X, Y);  
}  
}

**//Program.cs**

static void Main(string[] args)  
{  
Console.WriteLine("\*\*\*\*\* A First Look at Structures \*\*\*\*\*\n");  
**// Create an initial Point.**  
Point myPoint;  
myPoint.X = 349;  
myPoint.Y = 76;  
myPoint.**Display**();

**// Adjust the X and Y values.**myPoint.**Increment**();  
myPoint.**Display**();  
Console.ReadLine();  
}

Özel kurucu metodu aşağıdaki gibi tanımlanabilir.

struct Point  
{  
**// Fields of the structure.**  
public int X;  
public int Y;  
**// A custom constructor.**  
public Point(int XPos, int YPos)  
{  
X = XPos;  
Y = YPos;  
}  
...  
}

Yeni bir Point değişkeni oluşturma ise aşağıdaki gibi olur.

**// Call custom constructor.**Point p2 = new Point(50, 60);  
**// Prints X=50,Y=60.**  
p2.Display();

///////////////////////////////////////////////////////////////

**OTOMATİK ÖZELLİK (AUTOMATIC PROPERTY)**

class Car  
{  
**// Automatic properties!**  
public string PetName { get; set; }  
public int Speed { get; set; }  
public string Color { get; set; }  
public void DisplayStats()  
{  
Console.WriteLine("Car Name: {0}", PetName);  
Console.WriteLine("Speed: {0}", Speed);  
Console.WriteLine("Color: {0}", Color);  
}  
}

static void Main(string[] args)  
{  
Console.WriteLine("\*\*\*\*\* Fun with Automatic Properties \*\*\*\*\*\n");  
Car c = new Car();  
c.PetName = "Frank";  
c.Speed = 55;  
c.Color = "Red";  
Console.WriteLine("Your car is named {0}? That's odd...",  
c.PetName);  
c.DisplayStats();  
Console.ReadLine();  
}

class Point  
{  
public int X { get; set; }  
public int Y { get; set; }

public Point(int xVal, int yVal)  
{  
X = xVal;  
Y = yVal;  
}  
public Point() { }  
public void DisplayStats()  
{  
Console.WriteLine("[{0}, {1}]", X, Y);  
}  
}

**BAŞLATMA SÖZDİZİMİ İLE ÖZEL OLUŞTURUCULARI ÇAĞIRMA (Calling Custom Constructors with Initialization Syntax)**

// Here, the default constructor is called implicitly.  
Point finalPoint = new Point { X = 30, Y = 30 };

// Here, the default constructor is called explicitly.  
Point finalPoint = new Point**()** { X = 30, Y = 30 };

// Calling a custom constructor.  
Point pt = new Point(10, 16) { X = 100, Y = 100 };

**enum** **PointColor**  
{ LightBlue,

BloodRed,

Gold

}  
class **Point**  
{  
public int X { get; set; }  
public int Y { get; set; }  
public **PointColor** Color{ get; set; }  
public **Point**(**int** xVal, int yVal)  
{  
X = xVal;  
Y = yVal;  
Color = PointColor.Gold;  
}  
public Point(PointColor ptColor)  
{  
Color = ptColor;  
}  
public Point()  
: this(PointColor.BloodRed){ }  
public void DisplayStats()  
{  
Console.WriteLine("[{0}, {1}]", X, Y);  
Console.WriteLine("Point is {0}", Color);  
}  
}

///////////////////////////////////////////////////

**C# CONSTRUCTORS**

A constructor is a special method of the class which gets automatically invoked whenever an instance of the class is created. Like methods, a constructor also contains the collection of instructions that are executed at the time of Object creation. It is used to assign initial values to the **data members** of the same class.  
**Example :**

class Geek

{

.......

// Constructor

public Geek() {}

.......

}

// an object is created of Geek class,

// So above constructor is called

Geek obj = new Geek();

**Important points to Remember About Constructors**

* Constructor of a class must have the same name as the class name in which it resides.
* A constructor can not be abstract, final, static and Synchronized.
* Within a class, you can create only one static constructor.
* A constructor doesn’t have any return type, not even void.
* A static constructor cannot be a parameterized constructor.
* A class can have any number of constructors.
* Access modifiers can be used in constructor declaration to control its access i.e which other class can call the constructor.

**Types of Constructor**

1. **Default Constructor**
2. **Parametrized Constructor**
3. **Copy Constructor**
4. **Private Constructor**
5. **Static Constructor**

**Default Constructor**

A constructor with no parameters is called a default constructor. A default constructor has every instance of the class to be initialized to the same values. The default constructor initializes all numeric fields to zero and all string and object fields to null inside a class.

**Example :**

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // C# Program to illustrate calling  // a Default constructor  using System;  namespace DefaultConstructorExample {    class Geek {        int num;      string name;        // this would be invoked while the      // object of that class created.      Geek()      {          Console.WriteLine("Constructor Called");      }        // Main Method      public static void Main()      {            // this would invoke default          // constructor.          Geek geek1 = new Geek();            // Default constructor provides          // the default values to the          // int and object as 0, null          // Note:          // It Give Warning because          // Fields are not assign          Console.WriteLine(geek1.name);          Console.WriteLine(geek1.num);      }  }  } |

**Output :**

**Constructor Called**

**0**

**Note : This will also show some warnings as follows:**

prog.cs(8, 6): warning CS0649: Field `DefaultConstructorExample.Geek.num' is never assigned to, and will always have its default value `0'

prog.cs(9, 9): warning CS0649: Field `DefaultConstructorExample.Geek.name' is never assigned to, and will always have its default value `null'

**Parameterized Constructor**

A constructor having at least one parameter is called as parameterized constructor. It can initialize each instance of the class to different values.

**Example :**

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // C# Program to illustrate calling of  // parameterized constructor.  using System;  namespace ParameterizedConstructorExample {    class Geek {        // data members of the class.      String name;      int id;        // parameterized constructor would      // initialized data members with      // the values of passed arguments      // while object of that class created.      Geek(String name, int id)      {          this.name = name;          this.id = id;      }        // Main Method      public static void Main()      {            // This will invoke parameterized          // constructor.          Geek geek1 = new Geek("GFG", 1);          Console.WriteLine("GeekName = " + geek1.name +                           " and GeekId = " + geek1.id);      }  }  } |

**Output :**

GeekName = GFG and GeekId = 1

**Copy Constructor**

This constructor creates an object by copying variables from another object. Its main use is to initialize a new instance to the values of an existing instance.  
**Example :**

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // C# Program to illustrate calling  // a Copy constructor  using System;  namespace copyConstructorExample {    class Geeks {        private string month;      private int year;        // declaring Copy constructor      public Geeks(Geeks s)      {          month = s.month;          year = s.year;      }        // Instance constructor      public Geeks(string month, int year)      {          this.month = month;          this.year = year;      }        // Get details of Geeks      public string Details      {          get          {              return "Month: " + month.ToString() +                       "\nYear: " + year.ToString();          }      }        // Main Method      public static void Main()      {            // Create a new Geeks object.          Geeks g1 = new Geeks("June", 2018);            // here is g1 details is copied to g2.          Geeks g2 = new Geeks(g1);            Console.WriteLine(g2.Details);      }  }  } |

**Output :**

Month: June

Year: 2018

**Private Constructor**

If a constructor is created with private specifier is known as Private Constructor. It is not possible for other classes to derive from this class and also it’s not possible to create an instance of this class.  
**Points To Remember :**

* It is the implementation of a singleton class pattern.
* use private constructor when we have only static members.
* Using private constructor, prevents the creation of the instances of that class.

**Example :**

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // C# Program to illustrate calling  // a Private constructor  using System;  namespace privateConstructorExample {    public class Geeks {        // declare private Constructor      private Geeks()      {      }        // declare static variable field      public static int count\_geeks;        // declare static method      public static int geeks\_Count()      {          return ++count\_geeks;      }        // Main Method      public static void Main()      {            // If you uncomment the following          // statement, it will generate          // an error because the constructor          // is unaccessible:          // Geeks s = new Geeks(); // Error            Geeks.count\_geeks = 99;            // Accessing without any          // instance of the class          Geeks.geeks\_Count();            Console.WriteLine(Geeks.count\_geeks);            // Accessing without any          // instance of the class          Geeks.geeks\_Count();            Console.WriteLine(Geeks.count\_geeks);      }  }  } |

**Output :**

100

101

**Static Constructor**

Static Constructor has to be invoked only once in the class and it has been invoked during the creation of the first reference to a static member in the class. A static constructor is initialized static fields or data of the class and to be executed only once.  
**Points To Remember :**

* It can’t be called directly.
* When it is executing then the user has no control.
* It does not take access modifiers or any parameters.
* It is called automatically to initialize the class before the first instance created.

**Example :**

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // C# Program to illustrate calling  // a Static constructor  using System;  namespace staticConstructorExample {    class geeks {        // It is invoked before the first      // instance constructor is run.      static geeks()      {            // The following statement produces          // the first line of output,          // and the line occurs only once.          Console.WriteLine("Static Constructor");      }        // Instance constructor.      public geeks(int i)      {          Console.WriteLine("Instance Constructor " + i);      }        // Instance method.      public string geeks\_detail(string name, int id)      {          return "Name:" + name + " id:" + id;      }        // Main Method      public static void Main()      {            // Here Both Static and instance          // constructors are invoked for          // first instance          geeks obj = new geeks(1);            Console.WriteLine(obj.geeks\_detail("GFG", 1));            // Here only instance constructor          // will be invoked          geeks obj1 = new geeks(2);            Console.WriteLine(obj1.geeks\_detail("GeeksforGeeks", 2));      }  }  } |

**Base constructor.** If we have to add a constructor to a derived class, the base keyword is often useful. This will have the derived constructor invoke the base class constructor.[**Base**](https://www.dotnetperls.com/base)

**Here:** We have a Parrot class that derives from the Bird class (which derives from object).

**Parrot:** The Parrot constructor has a "base" constructor call (which means the Bird constructor is called when a Parrot is created).

**C# program that uses base constructor**

using System;

public **class** Bird

{

public Bird(int value)

{

Console.WriteLine($"Bird() called with {value}");

}

}

public **class** Parrot : Bird

{

public Parrot(int value) : **base**(value)

{

Console.WriteLine($"Parrot called with {value}");

}

}

class Program

{

static void **Main**()

{

Parrot parrot = new Parrot(450);

Console.WriteLine(":::DONE:::");

}

}

**Output**

Bird() called with 450

Parrot called with 450

:::DONE:::

**This constructor.** Sometimes in a class we have many constructors. We can use "this" to have one constructor invocation call another constructor method. This reduces code bloat.

**Tip:** Making code easy to read and understand should be a design goal for many classes.

**Tip 2:** The this-keyword allows code to be shared between the constructors. Constructor initializers are useful in nontrivial classes.

**Example:** The Mouse class has 2 constructors. The first constructor has no parameters. It calls into the second constructor with "this."

**Keyword:** The this-keyword here instructs the compiler to insert a call to the specified constructor at the top of the first constructor.

**C# program that uses this, constructor**

using System;

class Mouse {

public Mouse() : **this**(-1, "")

{

// Uses constructor initializer.

}

public Mouse(int weight, string name)

{

// Constructor implementation.

Console.WriteLine("Constructor weight = {0}, name = {1}", weight, name);

}

}

class Program {

static void Main() {

// Test the 2 constructors for Mouse type.

Mouse mouse1 = new Mouse();

Mouse mouse2 = new Mouse(10, "Sam");

}

}

**Output**

Constructor weight = -1, name =

Constructor weight = 10, name = Sam

////////////////////////////////////////////////

**Kısmi Sınıflar ve Yöntemler (C# Programlama Kılavuzu)**

Bir [sınıfın](https://docs.microsoft.com/tr-tr/dotnet/csharp/language-reference/keywords/class) tanımını, bir [yapıyı,](https://docs.microsoft.com/tr-tr/dotnet/csharp/language-reference/builtin-types/struct) [arabirimi](https://docs.microsoft.com/tr-tr/dotnet/csharp/language-reference/keywords/interface) veya yöntemi iki veya daha fazla kaynak dosya üzerinde bölmek mümkündür. Her kaynak dosya tür veya yöntem tanımının bir bölümünü içerir ve uygulama derlendiğinde tüm parçalar birleştirilir.

**Kısmi Sınıflar (Partial Class)**

Sınıf tanımının bölünmesi nin arzu edilen birkaç durum vardır:

* Büyük projeler üzerinde çalışırken, bir sınıfı ayrı dosyalara yaymak, birden çok programcının aynı anda üzerinde çalışmasını sağlar.
* Otomatik olarak oluşturulan kaynakla çalışırken, kod kaynak dosyasını yeniden oluşturmak zorunda kalmadan sınıfa eklenebilir. Visual Studio, Windows Formları, Web hizmeti paketleyici kodu ve benzeri oluştururken bu yaklaşımı kullanır. Visual Studio tarafından oluşturulan dosyayı değiştirmek zorunda kalmadan bu sınıfları kullanan kod oluşturabilirsiniz.
* Bir sınıf tanımını bölmek için, burada gösterildiği gibi [kısmi](https://docs.microsoft.com/tr-tr/dotnet/csharp/language-reference/keywords/partial-type) anahtar kelime değiştiricisini kullanın:

public partial class Employee

{

public void DoWork()

{

}

}

public partial class Employee

{

public void GoToLunch()

{

}

}

Anahtar partial kelime, sınıfın diğer bölümlerinin, yapının veya arabirimin ad alanında tanımlanabileceğini belirtir. Tüm parçalar anahtar partial kelimeyi kullanmalıdır. Tüm parçalar, son türü oluşturmak için derleme zamanında kullanılabilir olmalıdır. Tüm parçalar , , public privateve benzeri gibi aynı erişilebilirlik olmalıdır.

**Örnek 1**

**Açıklama**

Aşağıdaki örnekte, alanların ve sınıfın oluşturucusu, Coordsbir kısmi sınıf tanımında beyan PrintCoords edilir ve üye, başka bir kısmi sınıf tanımında bildirilir.

**Kod**

public partial class Coords

{

private int x;

private int y;

public Coords(int x, int y)

{

this.x = x;

this.y = y;

}

}

public partial class Coords

{

public void PrintCoords()

{

Console.WriteLine("Coords: {0},{1}", x, y);

}

}

class TestCoords

{

static void Main()

{

Coords myCoords = new Coords(10, 15);

myCoords.PrintCoords();

// Keep the console window open in debug mode.

Console.WriteLine("Press any key to exit.");

Console.ReadKey();

}

}

// Output: Coords: 10,15

**Örnek 2**

**Açıklama**

Aşağıdaki örnek, kısmi yapı ve arabirimler de geliştirebileceğinizi gösterir.

**Kod**

partial interface ITest

{

void Interface\_Test();

}

partial interface ITest

{

void Interface\_Test2();

}

partial struct S1

{

void Struct\_Test() { }

}

partial struct S1

{

void Struct\_Test2() { }

}

Büyük projelerde oluşturduğumuz class' lar zamanla okunması zor hale gelebilecek uzunlukta kod satırları ile dolabilmektedir. Partial class bize bir class' ı birden fazla class olarak bölmemize, constructor, değişken, property, metodları vs. düzenli bir şekilde ayrı ayrı oluşturmamızı sağlamaktadır. Fiziksel olarak birden fazla parça ile oluşan partial class' lar, çalışma zamanında tek bir class olarak bütün elemanları içerisinde barındırmaktadır.

Partial class ile fiziksel olarak parça class'ların birleşmesi için class isimlerinin aynı olması gerekmektedir.

Örneğimizde constructor, değişken, property ve metodlarımızı ayrı ayrı classlar içerisinde oluşturalım ve sonra çalışma anındaki birleşmiş halini inceleyelim.

Constructor class' ımızı tanımlayalım.

1. **public** partial **class** Personeller
2. {
3. **public** Personeller()
4. {
6. }
8. **public** Personeller(Personeller personel)
9. {
10. PersonelEkle(personel);
11. }
12. }

Değişkenlerimizi tanımlayalım.

1. **public** partial **class** Personeller
2. {
3. **private** **string** adi;
4. **private** **string** soyadi;
5. **private** **int** yas;
6. **private** List<Personeller> personel = **new** List<Personeller>();
7. }

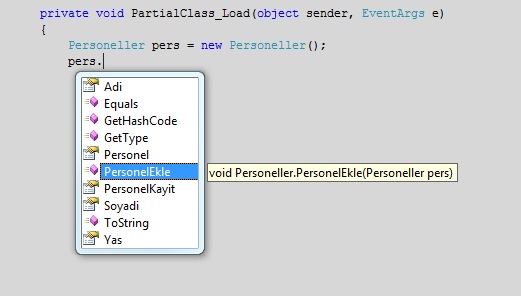
Propertylerimizi tanımlayalım.

1. **public** partial **class** Personeller
2. {
3. **public** **string** Adi
4. {
5. **get** { **return** adi; }
6. **set** { adi = value; }
7. }
9. **public** **string** Soyadi
10. {
11. **get** { **return** soyadi; }
12. **set** { soyadi = value; }
13. }
15. **public** **int** Yas
16. {
17. **get** { **return** yas; }
18. **set** { yas = value; }
19. }
21. **public** List<Personeller> Personel
22. {
23. **get**
24. {
25. **return** personel;
26. }
27. }
29. **public** Personeller PersonelKayit
30. {
31. **set**
32. {
33. personel.Add(value);
34. }
35. }
36. }

Son olarak bir tane metod tanımı yapalım.

1. **public** partial **class** Personeller
2. {
3. **public** **void** PersonelEkle(Personeller pers)
4. {
5. PersonelKayit = pers;
6. }
7. }

Gördüğünüz gibi fiziksel olarak 4 parçadan oluşan class' ımız, çalışma anında aşağıdaki şekilde görüldüğü gibi tek bir parça olarak görüntülenecektir.



Karmaşık class yapılarında oldukça kullanışlı bir yapıya sahiptir. Bu tarz yapılarda partial class ile karmaşık bir şekilde iç içe bulunan yapıları, ayrı birer class olarak tanımlamak koda daha iyi bir şekilde hakim olmanızada yardımcı olacaktır.

/////////////////////////////////////////

EXCEPTION HANDLING

try..catch..finally

C# provides three keywords try, catch and finally to implement exception handling. The try encloses the statements that might throw an exception whereas catch handles an exception if one exists. The finally can be used for any cleanup work that needs to be done.

Try..catch..finally block example:

1. **try**
2. {
3. // Statement which can cause an exception.
4. }
5. **catch**(Type x)
6. {
7. // Statements for handling the exception
8. }
9. **finally**
10. {
11. //Any cleanup code
12. }

If any exception occurs inside the try block, the control transfers to the appropriate catch block and later to the finally block.

But in C#, both catch and finally blocks are optional. The try block can exist either with one or more catch blocks or a finally block or with both catch and finally blocks.

If there is no exception occurred inside the try block, the control directly transfers to finally block. We can say that the statements inside the finally block is executed always. Note that it is an error to transfer control out of a finally block by using break, continue, return or goto.

In C#, exceptions are nothing but objects of the type Exception. The Exception is the ultimate base class for any exceptions in C#. The C# itself provides couple of standard exceptions. Or even the user can create their own exception classes, provided that this should inherit from either Exception class or one of the standard derived classes of Exception class like DivideByZeroExcpetion to ArgumentException etc.

Uncaught Exceptions

The following program will compile but will show an error during execution. The division by zero is a runtime anomaly and program terminates with an error message. Any uncaught exceptions in the current context propagate to a higher context and looks for an appropriate catch block to handle it. If it can't find any suitable catch blocks, the default mechanism of the .NET runtime will terminate the execution of the entire program.

1. //C#: Exception Handling
2. //Author: rajeshvs@msn.com
3. **using** System;
4. **class** MyClient
5. {
6. **public** **static** **void** Main()
7. {
8. **int** x = 0;
9. **int** div = 100/x;
10. Console.WriteLine(div);
11. }
12. }

The modified form of the above program with exception handling mechanism is as follows. Here we are using the object of the standard exception class DivideByZeroException to handle the exception caused by division by zero.

1. //C#: Exception Handling
2. **using** System;
3. **class** MyClient
4. {
5. **public** **static** **void** Main()
6. {
7. **int** x = 0;
8. **int** div = 0;
9. **try**
10. {
11. div = 100 / x;
12. Console.WriteLine("This linein not executed");
13. }
14. **catch** (DivideByZeroException)
15. {
16. Console.WriteLine("Exception occured");
17. }
18. Console.WriteLine($"Result is {div}");
19. }
20. }

Result from above code is show below:



In the above case, the program do not terminate unexpectedly. Instead, the program control passes from the point where exception occurred inside the try block to the catch blocks. If it finds any suitable catch block, executes the statements inside that catch and continues with the normal execution of the program statements.

If a finally block is present, the code inside the finally block will get also be executed.

1. //C#: Exception Handling
2. **using** System;
3. **class** MyClient
4. {
5. **public** **static** **void** Main()
6. {
7. **int** x = 0;
8. **int** div = 0;
9. **try**
10. {
11. div = 100/x;
12. Console.WriteLine("Not executed line");
13. }
14. **catch**(DivideByZeroException)
15. {
16. Console.WriteLine("Exception occured");
17. }
18. **finally**
19. {
20. Console.WriteLine("Finally Block");
21. }
22. Console.WriteLine($"Result is {div}");
23. }
24. }

Remember that in C#, the catch block is optional. The following program is perfectly legal in C#.

1. //C#: Exception Handling
2. **using** System;
3. **class** MyClient
4. {
5. **public** **static** **void** Main()
6. {
7. **int** x = 0;
8. **int** div = 0;
9. **try**
10. {
11. div = 100/x;
12. Console.WriteLine("Not executed line");
13. }
14. **finally**
15. {
16. Console.WriteLine("Finally Block");
17. }
18. Console.WriteLine($"Result is {div}");
19. }
20. }

But in this case, since there is no exception handling catch block, the execution will get terminated. But before the termination of the program statements inside the finally block will get executed. In C#, a try block must be followed by either a catch or finally block.

Multiple Catch Blocks

A try block can throw multiple exceptions, which can handle by using multiple catch blocks. Remember that more specialized catch block should come before a generalized one. Otherwise the compiler will show a compilation error.

1. //C#: Exception Handling: Multiple catch
2. **using** System;
3. **class** MyClient
4. {
5. **public** **static** **void** Main()
6. {
7. **int** x = 0;
8. **int** div = 0;
9. **try**
10. {
11. div = 100 / x;
12. Console.WriteLine("Not executed line");
13. }
14. **catch** (DivideByZeroException de)
15. {
16. Console.WriteLine("DivideByZeroException");
17. }
18. **catch** (Exception)
19. {
20. Console.WriteLine("Exception");
21. }
22. **finally**
23. {
24. Console.WriteLine("Finally Block");
25. }
26. Console.WriteLine($"Result is {div}");
27. }
28. }

Catching all Exceptions

By providing a catch block without brackets or arguments, we can catch all exceptions occurred inside a try block. Even we can use a catch block with an Exception type parameter to catch all exceptions happened inside the try block since in C#, all exceptions are directly or indirectly inherited from the Exception class.

1. //C#: Exception Handling: Handling all exceptions
2. **using** System;
3. **class** MyClient
4. {
5. **public** **static** **void** Main()
6. {
7. **int** x = 0;
8. **int** div = 0;
9. **try**
10. {
11. div = 100 / x;
12. Console.WriteLine("Not executed line");
13. }
14. **catch**
15. {
16. Console.WriteLine("oException");
17. }
18. Console.WriteLine($"Result is {div}");
19. }
20. }

The following program handles all exception with Exception object.

1. //C#: Exception Handling: Handling all exceptions
2. **using** System;
3. **class** MyClient
4. {
5. **public** **static** **void** Main()
6. {
7. **int** x = 0;
8. **int** div = 0;
9. **try**
10. {
11. div = 100 / x;
12. Console.WriteLine("Not executed line");
13. }
14. **catch** (Exception)
15. {
16. Console.WriteLine("oException");
17. }
18. Console.WriteLine($"Result is {div}");
19. }
20. }

Throwing an Exception

In C#, it is possible to throw an exception programmatically. The 'throw' keyword is used for this purpose. The general form of throwing an exception is as follows.

1. **throw** exception\_obj;

For example, the following statement throws an ArgumentException explicitly.

1. **throw** **new** ArgumentException("Exception");
3. //C#: Exception Handling:
4. **using** System;
5. **class** MyClient
6. {
7. **public** **static** **void** Main()
8. {
9. **try**
10. {
11. **throw** **new** DivideByZeroException("Invalid Division");
12. }
13. **catch** (DivideByZeroException)
14. {
15. Console.WriteLine("Exception");
16. }
17. Console.WriteLine("LAST STATEMENT");
18. }
19. }

Re-throwing an Exception

The exceptions, which we caught inside a catch block, can re-throw to a higher context by using the keyword throw inside the catch block. The following program shows how to do this.

1. //C#: Exception Handling: Handling all exceptions
2. **using** System;
3. **class** MyClass
4. {
5. **public** **void** Method()
6. {
7. **try**
8. {
9. **int** x = 0;
10. **int** sum = 100 / x;
11. }
12. **catch** (DivideByZeroException)
13. {
14. **throw**;
15. }
16. }
17. }
18. **class** MyClient
19. {
20. **public** **static** **void** Main()
21. {
22. MyClass mc = **new** MyClass();
23. **try**
24. {
25. mc.Method();
26. }
27. **catch** (Exception)
28. {
29. Console.WriteLine("Exception caught here");
30. }
31. Console.WriteLine("LAST STATEMENT");
32. }
33. }

Standard Exceptions

There are two types of exceptions: exceptions generated by an executing program and exceptions generated by the common language runtime. System.Exception is the base class for all exceptions in C#. Several exception classes inherit from this class including ApplicationException and SystemException. These two classes form the basis for most other runtime exceptions. Other exceptions that derive directly from System.Exception include IOException, WebException etc.

The common language runtime throws SystemException. The ApplicationException is thrown by a user program rather than the runtime. The SystemException includes the ExecutionEngineException, StaclOverFlowException etc. It is not recommended that we catch SystemExceptions nor is it good programming practice to throw SystemExceptions in our applications.

* System.OutOfMemoryException
* System.NullReferenceException
* Syste.InvalidCastException
* Syste.ArrayTypeMismatchException
* System.IndexOutOfRangeException
* System.ArithmeticException
* System.DevideByZeroException
* System.OverFlowException

User-defined Exceptions

In C#, it is possible to create our own exception class. But Exception must be the ultimate base class for all exceptions in C#. So the user-defined exception classes must inherit from either Exception class or one of its standard derived classes.

1. //C#: Exception Handling: User defined exceptions
2. **using** System;
3. **class** MyException : Exception
4. {
5. **public** MyException(**string** str)
6. {
7. Console.WriteLine("User defined exception");
8. }
9. }
10. **class** MyClient
11. {
12. **public** **static** **void** Main()
13. {
14. **try**
15. {
16. **throw** **new** MyException("RAJESH");
17. }
18. **catch** (Exception)
19. {
20. Console.WriteLine("Exception caught here" + e.ToString());
21. }
22. Console.WriteLine("LAST STATEMENT");
23. }
24. }

Design Guidelines

Exceptions should be used to communicate exceptional conditions. Don't use them to communicate events that are expected, such as reaching the end of a file. If there's a good predefined exception in the System namespace that describes the exception condition-one that will make sense to the users of the class-use that one rather than defining a new exception class and put specific information in the message. Finally, if code catches an exception that it isn't going to handle, consider whether it should wrap that exception with additional information before re-throwing it.

//////////////////////////////////////////////////////////////

**STACKTRACE**

**System.Exception.StackTrace** özelliği, kural dışı durumla sonuçlanan çağrı dizisini tanımlamanızı sağlar. İstisna (exception) oluşturulurken otomatik olarak kurulduğundan, StackTrace'in değerini asla ayarlamadığınızı unutmayın.

catch(Exception e)  
{  
...  
Console.WriteLine("Stack: {0}", e.StackTrace);  
}

Yukarıdaki kod çalışırılığında aşağıdakine benzer bir hata çıktısı olacaktır.

Stack: at SimpleException.Car.Accelerate(Int32 delta)  
in c:\MyApps\SimpleException\car.cs:line 65 at SimpleException.Program.Main()  
in c:\MyApps\SimpleException\Program.cs:line 21

//////////////////////////////////////////////////////////

**ICOMAPARABLE INTERFACE**

Bir değer türünün veya sınıfının örneklerini sıralamak veya sıralamak için uyguladığı Genelleştirilmiş türe özgü bir karşılaştırma yöntemi tanımlar.

Arabirim, bir başvuru türünü parametre olarak alan ve geçerli örneğin diğer nesneyle (MSDN) sıralama düzeninde öncekine, izlemeye veya aynı konumda gerçekleşmesine bağlı olarak bir tamsayı döndüren CompareTo yöntemine sahiptir.

// The iteration of the Car can be ordered  
// based on the CarID.  
public class Car : IComparable  
{  
...

// IComparable implementation.  
int IComparable.CompareTo(object obj)  
{  
Car temp = obj as Car;  
if (temp != null)  
{  
if (this.CarID > temp.CarID)  
return 1;  
if (this.CarID < temp.CarID)  
return -1;  
else  
return 0;  
}  
else  
throw new ArgumentException("Parameter is not a Car!");  
}  
}

**Örnekler**

Aşağıdaki örnekte [IComparable](https://docs.microsoft.com/tr-tr/dotnet/api/system.icomparable?view=netframework-4.8) ve önkoşul [CompareTo](https://docs.microsoft.com/tr-tr/dotnet/api/system.icomparable.compareto?view=netframework-4.8) yönteminin uygulanması gösterilmektedir.

C#Kopyala

using System;

using System.Collections;

public class Temperature : IComparable

{

// The temperature value

protected double temperatureF;

public int CompareTo(object obj) {

if (obj == null) return 1;

Temperature otherTemperature = obj as Temperature;

if (otherTemperature != null)

return this.temperatureF.CompareTo(otherTemperature.temperatureF);

else

throw new ArgumentException("Object is not a Temperature");

}

public double Fahrenheit

{

get

{

return this.temperatureF;

}

set {

this.temperatureF = value;

}

}

public double Celsius

{

get

{

return (this.temperatureF - 32) \* (5.0/9);

}

set

{

this.temperatureF = (value \* 9.0/5) + 32;

}

}

}

public class CompareTemperatures

{

public static void Main()

{

ArrayList temperatures = new ArrayList();

// Initialize random number generator.

Random rnd = new Random();

// Generate 10 temperatures between 0 and 100 randomly.

for (int ctr = 1; ctr <= 10; ctr++)

{

int degrees = rnd.Next(0, 100);

Temperature temp = new Temperature();

temp.Fahrenheit = degrees;

temperatures.Add(temp);

}

// Sort ArrayList.

temperatures.Sort();

foreach (Temperature temp in temperatures)

Console.WriteLine(temp.Fahrenheit);

}

}

// The example displays the following output to the console (individual

// values may vary because they are randomly generated):

// 2

// 7

// 16

// 17

// 31

// 37

// 58

// 66

// 72

// 95

**Açıklamalar**

Bu arabirim, değerleri sıralanabilir veya sıralanabilen türler tarafından uygulanır. Bu, sıralama düzeninde geçerli örneğin konumunun, daha önce, sonra veya aynı türdeki ikinci bir nesneyle aynı olup olmadığını belirten tek bir yöntem [CompareTo(Object)](https://docs.microsoft.com/tr-tr/dotnet/api/system.icomparable.compareto?view=netframework-4.8#System_IComparable_CompareTo_System_Object_)tanımlar. Örneğin [IComparable](https://docs.microsoft.com/tr-tr/dotnet/api/system.icomparable?view=netframework-4.8) uygulama [Array.Sort](https://docs.microsoft.com/tr-tr/dotnet/api/system.array.sort?view=netframework-4.8) ve [ArrayList.Sort](https://docs.microsoft.com/tr-tr/dotnet/api/system.collections.arraylist.sort?view=netframework-4.8)gibi yöntemler tarafından otomatik olarak çağrılır.

[CompareTo(Object)](https://docs.microsoft.com/tr-tr/dotnet/api/system.icomparable.compareto?view=netframework-4.8#System_IComparable_CompareTo_System_Object_) yönteminin uygulanması, aşağıdaki tabloda gösterildiği gibi üç değerden birine sahip bir [Int32](https://docs.microsoft.com/tr-tr/dotnet/api/system.int32?view=netframework-4.8) döndürmelidir.

| **1 TABLOSU** | |
| --- | --- |
| **Değer** | **Anlamı** |
| Sıfırdan küçük | Geçerli örnek, sıralama düzeninde [CompareTo](https://docs.microsoft.com/tr-tr/dotnet/api/system.icomparable.compareto?view=netframework-4.8) yöntemi tarafından belirtilen nesneden önce gelir. |
| Sıfırlama | Bu geçerli örnek, sıralama düzeninde [CompareTo](https://docs.microsoft.com/tr-tr/dotnet/api/system.icomparable.compareto?view=netframework-4.8) yöntemi tarafından belirtilen nesneyle aynı konumda meydana gelir. |
| Sıfırdan büyük | Bu geçerli örnek sıralama düzeninde [CompareTo](https://docs.microsoft.com/tr-tr/dotnet/api/system.icomparable.compareto?view=netframework-4.8) yöntemi tarafından belirtilen nesneyi izler. |

Tüm sayısal türler ([Int32](https://docs.microsoft.com/tr-tr/dotnet/api/system.int32?view=netframework-4.8) ve [Double](https://docs.microsoft.com/tr-tr/dotnet/api/system.double?view=netframework-4.8)gibi), [String](https://docs.microsoft.com/tr-tr/dotnet/api/system.string?view=netframework-4.8), [Char](https://docs.microsoft.com/tr-tr/dotnet/api/system.char?view=netframework-4.8)ve [DateTime](https://docs.microsoft.com/tr-tr/dotnet/api/system.datetime?view=netframework-4.8)gibi [IComparable](https://docs.microsoft.com/tr-tr/dotnet/api/system.icomparable?view=netframework-4.8)uygular. Özel türler ayrıca nesne örneklerinin sıralanmasını veya sıralanmasını sağlamak için kendi [IComparable](https://docs.microsoft.com/tr-tr/dotnet/api/system.icomparable?view=netframework-4.8) uygulanmasını sağlamalıdır.

**Yöntemler**

|  |  |
| --- | --- |
| **2 TABLOSU** | |
| [CompareTo(Object)](https://docs.microsoft.com/tr-tr/dotnet/api/system.icomparable.compareto?view=netframework-4.8#System_IComparable_CompareTo_System_Object_) | Mevcut örneği aynı türdeki başka bir nesne ile karşılaştırır ve geçerli örneğin, diğer nesneyle aynı sıralama düzeni konumunda mı olduğunu, önünde mi olduğunu, yoksa arkasında mı olduğunu belirten bir tamsayı döndürür. |

// This helper class is used to sort an array of Cars by pet name.  
public class PetNameComparer : IComparer  
{  
// Test the pet name of each object.  
int IComparer.Compare(object o1, object o2)  
{  
Car t1 = o1 as Car;  
Car t2 = o2 as Car;  
if(t1 != null && t2 != null)  
return String.Compare(t1.PetName, t2.PetName);  
else  
throw new ArgumentException("Parameter is not a Car!");  
}  
}

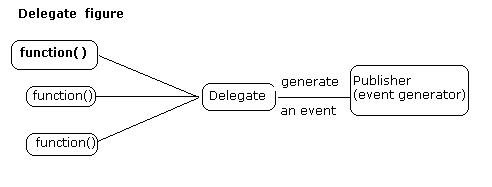
static void Main(string[] args)  
{  
...  
// Now sort by pet name.  
Array.Sort(myAutos, new PetNameComparer());  
// Dump sorted array.  
Console.WriteLine("Ordering by pet name:");  
foreach(Car c in myAutos)  
Console.WriteLine("{0} {1}", c.CarID, c.PetName);  
...  
}

///////////////////////////////////////////////////////////////////////////////////

**DELEGATE**

1. What is a Delegate in C#?

 Delegate is one of the base types in .NET. Delegate is a class, which is used to create and invoke delegates at runtime.



A delegate in C# is similar to a function pointer in C or C++. It's a new type of object in C#. Delegate is very special type of object as earlier the entire the object we used to defined contained data but delegate just contains the details of a method.

**2. Why do we need delegates in C#?**

Programmers often needs to pass a method as a parameter of other methods. For this purpose we create and use delegates.

A delegate is a class that encapsulates a method signature. Although it can be used in any context, it often serves as the basis for the event-handling model in C# and .NET.

One good way of understanding delegates is by thinking of a delegate as something that gives a name to a method signature.

Example:

1. **public** **delegate** **int** DelegateMethod(**int** x, **int** y);

Any method that matches the delegate's signature, which consists of the return type and parameters, can be assigned to the delegate.

This makes is possible to programmatically change method calls, and also plug new code into existing classes. As long as you know the delegate's signature, you can assign your own-delegated method.

This ability to refer to a method as a parameter makes delegates ideal for defining callback methods.

**Delegate magic**

In class we create its object, which is instance, but in delegate when we create instance that is also referred as delegate (means whatever you do you will get delegate).

Delegate does not know or care about the class of the object that it references. Any object will do; all that matters is that the method's argument types and return type match the delegate's. This makes delegates perfectly suited for "anonymous" invocation.

**3. What Are The Benefits Of Delegates?**

In simple words delegates are object oriented and type-safe and very secure as they ensure that the signature of the method being called is correct. Delegates makes event handling simple and easy.

4. What are types of delegates in C#?

There are two types of delegates, singlecast delegates, and multiplecast delegates.

**Singlecast delegate**

Singlecast delegate point to single method at a time. In this the delegate is assigned to a single method at a time. They are derived from System.Delegate class.

**Multicast Delegate**

When a delegate is wrapped with more than one method that is known as a multicast delegate.

In C#, delegates are multicast, which means that they can point to more than one function at a time. They are derived from System.MulticastDelegate class.

5. How to define a delegates in C#?

There are three steps in defining and using delegates:

*1. Declaration*

To create a delegate, you use the delegate keyword.

1. [attributes] [modifiers] **delegate** ReturnType Name ([formal-parameters]);

* The attributes factor can be a normal C# attribute.
* The modifier can be one or an appropriate combination of the following keywords: new, public, private, protected, or internal.
* The ReturnType can be any of the data types we have used so far. It can also be a type void or the name of a class.
* The Name must be a valid C# name.

Because a delegate is a definituon for a method, you must use parentheses, required for every method. If this method will not take any argument, leave the parentheses empty.

Example:

1. public delegate void DelegateExample();

The above code is how a delegate with no papameters is defined.

*2. Instantiation*

1. DelegateExample d1 = new DelegateExample(Display);

The above code shows how a delegate is initiated.

*3. Invocation*

1. d1();

The above code piece invokes a delegate d1().

6. What is a Singlecast delegate in C#?

Here is a sample code that demonstrates how to create and use a singlecast delegate.

1. using System;
2. namespace ConsoleApplication5
3. {
4. class Program
5. {
6. public delegate void delmethod();
8. **public** **class** P
9. {
11. **public** **static** **void** display()
12. {
13. Console.WriteLine("Hello!");
14. }
16. **public** **static** **void** show()
17. {
18. Console.WriteLine("Hi!");
19. }
21. **public** **void** print()
22. {
23. Console.WriteLine("Print");
24. }
25. }
27. **static** **void** Main(**string**[] args)
28. {
29. // here we have assigned static method show() of class P to delegate delmethod()
30. delmethod del1 = P.show;
32. // here we have assigned static method display() of class P to delegate delmethod() using new operator
33. // you can use both ways to assign the delagate
34. delmethod del2 = **new** delmethod(P.display);
35. P obj = **new** P();
37. // here first we have create instance of class P and assigned the method print() to the delegate i.e. delegate with class
38. delmethod del3 = obj.print;
40. del1();
41. del2();
42. del3();
43. Console.ReadLine();
44. }
45. }
46. }

7. What is a Multicast delegate in C#?

Here is sample code that demonstrates how to create and use a multicast delegate.

1. **using** System;
2. **namespace** delegate\_Example4
3. {
5. **class** Program
6. {
7. **public** **delegate** **void** delmethod(**int** x, **int** y);
9. **public** **class** TestMultipleDelegate
10. {
11. **public** **void** plus\_Method1(**int** x, **int** y)
12. {
13. Console.Write("You are in plus\_Method");
14. Console.WriteLine(x + y);
15. }
17. **public** **void** subtract\_Method2(**int** x, **int** y)
18. {
19. Console.Write("You are in subtract\_Method");
20. Console.WriteLine(x - y);
21. }
22. }
24. **static** **void** Main(**string**[] args)
25. {
27. TestMultipleDelegate obj = **new** TestMultipleDelegate();
28. delmethod del = **new** delmethod(obj.plus\_Method1);
30. // Here we have multicast
31. del += **new** delmethod(obj.subtract\_Method2);
32. // plus\_Method1 and subtract\_Method2 are called
33. del(50, 10);
34. Console.WriteLine();
35. //Here again we have multicast
36. del -= **new** delmethod(obj.plus\_Method1);
37. //Only subtract\_Method2 is called
38. del(20, 10);
39. Console.ReadLine();
40. }
41. }
42. }

**Point to remember about Delegates:**

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

The most common example of using delegates is in events.

* They are extensively used in threading
* Delegates are also used for generic class libraries, which have generic functionality, defined.

**8. What are Anonymous Delegates in C#?**

You can create a delegate, but there is no need to declare the method associated with it. You do not have to explicitly define a method prior to using the delegate. Such a method is referred to as anonymous. In other words, if a delegate itself contains its method definition it is known as anonymous method.

The code is an example of using an anonymous delegate.

1. **using** System;
3. **public** **delegate** **void** Test();
5. **public** **class** Program
6. {
7. **static** **int** Main()
8. {
9. Test Display = **delegate**()
10. {
11. Console.WriteLine("Anonymous Delegate method");
12. };
14. Display();
15. **return** 0;
16. }
17. }

Note: You can also handle event in anonymous method.

**9. How Delegates are Related to Events in C#?**

Events and delegate work together. An event is a reference to a delegate i.e. when an event is raised, a delegate is called. In C# terms, events are a special form of delegates.

Events play an important part in user interfaces and programming notifications. Events and delegates work hand-in-hand to provide a communication between code from one class to other class. When something happens in one class or one part of the code and other part of the code needs a notification, events are used.

A C# event is a class member that is activated whenever the event it was designed for occurs. It starts with a class that declares an event. Any class, including the same class that the event is declared in, may register one of its methods for the event. This occurs through a delegate, which specifies the signature of the method that is registered for the event. The event keyword is a delegate modifier. It must always be used in connection with a delegate.

The delegate may be one of the pre-defined .NET delegates or one you declare yourself. Whichever is appropriate, you assign the delegate to the event, which effectively registers the method that will be called when the event fires.

**10. How to Use Events and Delegates in C#?**

Once an event is declared, it must be associated with one or more event handlers before it can be raised. An event handler is nothing but a method that is called using a delegate. Use the += operator to associate an event with an instance of a delegate that already exists.

Example:

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

An event has the value null if it has no registered listeners.

Although events are mostly used in Windows controls programming, they can also be implemented in console, web and other applications.

**Program for creating a custom Singlecast delegate and event**

1. **using** System;
2. **namespace** delegate\_custom
3. {
4. **class** Program
5. {
6. **public** **delegate** **void** MyDelegate(**int** a);
8. **public** **class** XX
9. {
10. **public** **event** MyDelegate MyEvent;
12. **public** **void** RaiseEvent()
13. {
14. MyEvent(20);
15. Console.WriteLine("Event Raised");
16. }
18. **public** **void** Display(**int** x)
19. {
20. Console.WriteLine("Display Method {0}", x);
21. }
22. }
24. **static** **void** Main(**string**[] args)
25. {
27. XX obj = **new** XX();
28. obj.MyEvent += **new** MyDelegate(obj.Display);
30. obj.RaiseEvent();
31. Console.ReadLine();
32. }
33. }
34. }

**Program for creating custom a multiplecast delegate and event**

1. **using System;**
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** System.Text;
6. **namespace** delegate\_custom\_multicast
7. {
8. **class** Program
9. {
10. **public** **delegate** **void** MyDelegate(**int** a, **int** b);
12. **public** **class** XX
13. {
14. **public** **event** MyDelegate MyEvent;
16. **public** **void** RaiseEvent(**int** a, **int** b)
17. {
18. MyEvent(a, b);
19. Console.WriteLine("Event Raised");
20. }
22. **public** **void** Add(**int** x, **int** y)
23. {
24. Console.WriteLine("Add Method {0}", x + y);
25. }
27. **public** **void** Subtract(**int** x, **int** y)
28. {
29. Console.WriteLine("Subtract Method {0}", x - y);
30. }
31. }
33. **static** **void** Main(**string**[] args)
34. {
36. XX obj = **new** XX();
37. obj.MyEvent += **new** MyDelegate(obj.Add);
38. obj.MyEvent += **new** MyDelegate(obj.Subtract);
39. obj.RaiseEvent(20, 10);
40. Console.ReadLine();
41. }
42. }
43. }

//MathOperations.cs file

namespace \_29\_MultiCastDelegate

{

public class MathOperations

{

/// <summary>

/// Sum method

/// </summary>

/// <param name="number1">first number</param>

/// <param name="number2">second number</param>

public void Sum(int number1,int number2) {

ForegroundColor = DarkBlue;

WriteLine("\nYou are in Sum method");

WriteLine($"Sum result  is :{number1 + number2} ");

}

/// <summary>

/// Substract method

/// </summary>

/// <param name="number1">first number</param>

/// <param name="number2">second number</param>

public void Substract(int number1, int number2)

{

ForegroundColor = DarkGreen;

WriteLine("\nYou are in Substract method");

WriteLine($"Substarct result  is :{number1 - number2} ");

}

/// <summary>

/// Multiple method

/// </summary>

/// <param name="number1">first number</param>

/// <param name="number2">second number</param>

public void Multiple(int number1, int number2)

{

ForegroundColor = DarkCyan;

WriteLine("\nYou are in Multiple method");

WriteLine($"Multiple result  is :{number1 \* number2} ");

}

/// <summary>

/// Division method

/// </summary>

/// <param name="number1">first number</param>

/// <param name="number2">second number</param>

public void Division(int number1, int number2)

{

ForegroundColor = DarkYellow;

WriteLine("\nYou are in Division method");

WriteLine($"Substarct result  is :{(int)(number1 / number2)} ");

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using static System.Console;

using static System.ConsoleColor;

namespace \_29\_MultiCastDelegate

{

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

class Program

{

static void Main(string[] args)

{

MathOperations mathOperations = new MathOperations();

DelMethod delmethod = new DelMethod(mathOperations.Sum);

delmethod += new DelMethod(mathOperations.Substract); //Substract method add the chain

ForegroundColor = DarkGray;

WriteLine("delmethod(20,30)");

WriteLine("\*\*\*\*\* Call the Sum and Substract methods from the chain  \*\*\*\*\*\*");

delmethod(20,30);  ///call the Sum and Substract methods

delmethod += new DelMethod(mathOperations.Division); //Division method add the chain

delmethod += new DelMethod(mathOperations.Multiple); //Multiple method add the chain

ForegroundColor = DarkMagenta;

WriteLine("\*\*\*\*\* Call the all methods from the chain  \*\*\*\*\*\*");

delmethod(20, 30);  ///call the Sum, Substract, Multiple and  Division methods

ForegroundColor = DarkRed;

WriteLine("\*\*\*\*\* remove the Multiple method from the chain  \*\*\*\*\*\*");

//remove the Division method from the chain

delmethod -= new DelMethod(mathOperations.Division); //Division method removed the chain

delmethod(20, 30);  ///call the Sum, Substract and  Multiple methods

delmethod += new DelMethod(mathOperations.Division); //Division method added the chain again

delmethod(20, 30);  ///call the Sum, Substract and  Division methods

ReadKey();

}

}

}

/////////////////////////////////////////////////////////////////////////////////

**C# GENERIC DELEGATE**

namespace \_30\_GenericDelegate

{

public delegate void MyGenericDelegate<T>(T arg); //call the void and single parameter methods

class Program

{

/// <summary>

/// Void return type

/// </summary>

/// <param name="arg">string type argument</param>

static void StringTarget(string arg)

{

WriteLine($"arg in upper case {arg.ToUpper()}"); //with c# 6

//WriteLine("arg in uppaercase{0}",arg);

}

// <summary>

/// Void return type

/// </summary>

/// <param name="arg">int type argument</param>

static void IntTarget(int arg)

{

WriteLine($"++arg in {++arg}"); //with c# 6

//WriteLine($"++arg in uppaercase{++arg}");

}

// <summary>

/// Void return type

/// </summary>

/// <param name="arg">double type argument</param>

static void DoubleTarget(double arg)

{

WriteLine($"++arg in double {(double)1.25+arg}"); //with c# 6

}

static void Main(string[] args)

{

//string type  delegate

MyGenericDelegate<string> strTarget = new MyGenericDelegate<string>(StringTarget);

strTarget("Some string data...");

//int type delegate

MyGenericDelegate<int> intTarget = new MyGenericDelegate<int>(IntTarget);

//MyGenericDelegate<int> intTarget = new MyGenericDelegate<int>(Program.IntTarget);

intTarget(229);

//double type delegate

MyGenericDelegate<double> doubleTarget = new MyGenericDelegate<double>(DoubleTarget);

doubleTarget(12.200);

ReadKey();

}

}

}

C# - Func

We have learned in the previous section, that a [delegates](https://www.tutorialsteacher.com/csharp/csharp-delegates) can be defined as shown below.

Example: C# Delegate

public delegate int SomeOperation(int i, int j);

class Program

{

static int Sum(int x, int y)

{

return x + y;

}

static void Main(string[] args)

{

SomeOperation add = Sum;

int result = add(10, 10);

Console.WriteLine(result);

}

}

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-JKJWrN)

Output:

20

C# 3.0 includes built-in generic delegate types Func and Action, so that you don't need to define custom delegates as above.

Func is a generic delegate included in the System namespace. It has zero or more *input* parameters and one *out* parameter. The last parameter is considered as an out parameter.

For example, a Func delegate that takes one input parameter and one out parameter is defined in the System namespace as below:

**Signature: Func**

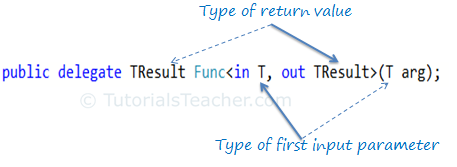
namespace System

{

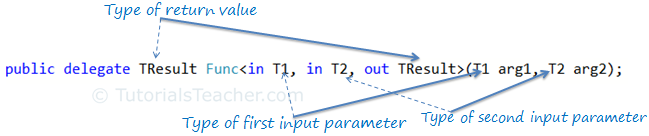
public delegate TResult Func<in T, out TResult>(T arg);

}

The last parameter in the angle brackets <> is considered as the return type and remaining parameters are considered as input parameter types as shown in the following figure.

[](https://www.tutorialsteacher.com/Content/images/csharp/func-delegate.png)Func delegate

A Func delegate with two input parameters and one out parameters will be represent as below.

[](https://www.tutorialsteacher.com/Content/images/csharp/func-delegate2.png)Func delegate

The following Func type delegate is the same as the above SomeOperation delegate, where it takes two input parameters of int type and returns a value of int type:

Func<int, int, int> sum;

You can assign any method to the above func delegate that takes two *int* parameters and returns an *int* value. Now, you can take Func delegate instead of someOperation delegate in the first example.

Example: Func

class Program

{

static int Sum(int x, int y)

{

return x + y;

}

static void Main(string[] args)

{

Func<int,int, int> add = Sum;

int result = add(10, 10);

Console.WriteLine(result);

}

}

Output:

20

A Func delegate type can include 0 to 16 input parameters of different types. However, it must include one out parameter for result. For example, the following func delegate doesn't have any input parameter, it includes only a out parameter.

namespace \_32\_Func\_Delegate

{

class Program

{

static int Add(int x,int y)

{

return x + y;

}

static string SumToString(int x,int y) {

return (x + y).ToString();

}

static void Main(string[] args)

{

Func<int, int, int> funcTarget1 = new Func<int, int, int>(Add);

int result = funcTarget1(25,35);

WriteLine("25+35={0}",result);

///WriteLine($"25+35={result}");

Func<int, int, string> functTarget2 = new Func<int, int, string>(SumToString);

string stringresult =functTarget2(56,35);

WriteLine("56+35={0}", stringresult);

///WriteLine($"25+35={stringresult}"); //c#6

ReadKey();

}

}

}

Example: Func with Zero Input Parameter

Func<int> getRandomNumber;

**C# Func with an Anonymous Method**

You can assign an anonymous method to the Func delegate by using the delegate keyword.

Example: Func with Anonymous Method

Func<int> getRandomNumber = delegate()

{

Random rnd = new Random();

return rnd.Next(1, 100);

};

namespace \_32\_Func\_With\_Anonymous\_Method

{

class Program

{

static void Main(string[] args)

{

Func<int, int, int> Add = delegate (int x, int y)

{

return x + y;

};

int result = Add(25,35);

WriteLine("25+35={0}",result);

///WriteLine($"25+35={result}");

Func<int, int, string> SumToString = delegate (int x, int y)

{

return (x + y).ToString();

};

int stringresult = Add(56, 35);

WriteLine("25+35={0}", stringresult);

///WriteLine($"25+35={stringresult}"); //c#6

ReadKey();

} } }

**Func with Lambda Expression**

A Func delegate can also be used with a lambda expression, as shown below:

Example: Func With Lambda Expression

Func<int> getRandomNumber = () => new Random().Next(1, 100);

//Or

Func<int, int, int> Sum = (x, y) => x + y;

namespace \_34\_Func\_Lambda\_Expression

{

class Program

{

static void Main(string[] args)

{

Func<int, int, int> Add = (x, y) => x + y;

int intResult = Add(25, 35);

WriteLine($"25+35={intResult}");

Func<int, int, string> SumToString = (x, y) => (x + y).ToString();

string stringResult = SumToString(100,50);

WriteLine("100+50  = {0}",stringResult);

ReadKey();

}

}

}

 Points to Remember :

1. Func is built-in delegate type.
2. Func delegate type must return a value.
3. Func delegate type can have zero to 16 input parameters.
4. Func delegate does not allow ref and out parameters.
5. Func delegate type can be used with an [anonymous method](https://www.tutorialsteacher.com/csharp/csharp-anonymous-method) or [lambda expression](https://www.tutorialsteacher.com/linq/linq-lambda-expression).

**C# - Action Delegate**

Action is also a delegate type defined in the System namespace. An Action type delegate is the same as [Func delegate](https://www.tutorialsteacher.com/csharp/csharp-func-delegate) except that the Action delegate doesn't return a value (void). In other words, an Action delegate can be used with a method that has a void return type.

. In many cases, you simply want “some delegate” that takes a set of arguments and possibly has a return value other than void. In these cases, you can use the framework’s built-in Action<> and Func<> delegate types. To illustrate their usefulness, create a new Console Application project named ActionAndFuncDelegates.  
The generic Action<> delegate is defined in the System namespaces of the mscorlib.dll and System.Core.dll assemblies.

You can use this generic delegate to “point to” a method that takes up to 16 arguments (that ought to be enough!) and returns void. Now recall, because Action<> is a generic delegate, you will need to specify the underlying types of each parameter as well.  
Update your Program class to define a new static method that takes three (or so) unique parameters.

For example, the following delegate prints an int value.

Example: C# Delegate

public delegate void Print(int val);

static void ConsolePrint(int i)

{

Console.WriteLine(i);

}

static void Main(string[] args)

{

Print prnt = ConsolePrint;

prnt(10);

}

Output:

10

You can use an Action delegate instead of defining the above Print delegate, for example:

**Example: Action delegate**

static void ConsolePrint(int i)

{

Console.WriteLine(i);

}

static void Main(string[] args)

{

Action<int> printActionDel = ConsolePrint;

printActionDel(10);

}

You can initialize an Action delegate using the new keyword or by directly assigning a method:

Action<int> printActionDel = ConsolePrint;

//Or

Action<int> printActionDel = new Action<int>(ConsolePrint);

namespace \_31\_Action\_Delegate

{

class Program

{

static void DisplayMessage(string message,ConsoleColor textColor,int printCount)

{

// Set color of console text.

ConsoleColor previousColor = Console.ForegroundColor;

//change text color(Console.ForegroundColor) property

Console.ForegroundColor = textColor;

for (int i = 0; i < printCount; i++)

{

WriteLine(message);

}

//Restore text color(Console.ForegroundColor) property

Console.ForegroundColor = previousColor;

Console.WriteLine("ForegroundColor property chnaged");

}

static void Main(string[] args)

{

Action<string, ConsoleColor, int> action = new Action<string, ConsoleColor, int>(DisplayMessage);

action("Delegate message", ConsoleColor.DarkYellow, 10);

WriteLine("After delegate");

ReadKey();

}

}

}

An Action delegate can take up to 16 input parameters of different types.

An Anonymous method can also be assigned to an Action delegate, for example:

**Example: Anonymous method with Action delegate**

static void Main(string[] args)

{

Action<int> printActionDel = delegate(int i)

{

Console.WriteLine(i);

};

printActionDel(10);

}

namespace \_35\_Action\_With\_Anonymous\_Method

{

class Program

{

static void Main(string[] args)

{

Action<string, ConsoleColor, int> DisplayMessage =

delegate (string message, ConsoleColor textColor, int printCount)

{

ConsoleColor previousColor = Console.ForegroundColor;

Console.ForegroundColor = textColor;

for (int i = 0; i < printCount; i++)

{

WriteLine(i+1+" . "+message);

}

Console.ForegroundColor = previousColor;

//Restore text color(Console.ForegroundColor) property

WriteLine("Restore text color");

};

DisplayMessage("Delegate message 1",ConsoleColor.DarkGreen,20);

ReadKey();

}

}

}

Output:

10

A Lambda expression also can be used with an Action delegate:

Example: Lambda expression with Action delegate

static void Main(string[] args)

{

Action<int> printActionDel = i => Console.WriteLine(i);

printActionDel(10);

}

Thus, you can use any method that doesn't return a value with Action delegate types.

Advantages of Action and Func Delegates

1. Easy and quick to define delegates.
2. Makes code short.
3. Compatible type throughout the application.

 Points to Remember :

1. Action delegate is same as func delegate except that it does not return anything. Return type must be void.
2. Action delegate can have 0 to 16 input parameters.
3. Action delegate can be used with [anonymous methods](https://www.tutorialsteacher.com/csharp/csharp-anonymous-method) or [lambda expressions](https://www.tutorialsteacher.com/linq/linq-lambda-expression).

C# - Predicate Delegate

A predicate is also a delegate like [Func](https://www.tutorialsteacher.com/csharp/csharp-func-delegate) and [Action](https://www.tutorialsteacher.com/csharp/csharp-action-delegate) delegates. It represents a method that contains a set of criteria and checks whether the passed parameter meets those criteria or not. A predicate delegate methods must take one input parameter and return a boolean - true or false.

The Predicate delegate is defined in the System namespace as shown below:

Predicate signature: public delegate bool Predicate<in T>(T obj);

Same as other delegate types, Predicate can also be used with any method, anonymous method or lambda expression.

Example: Predicate delegate

static bool IsUpperCase(string str)

{

return str.Equals(str.ToUpper());

}

static void Main(string[] args)

{

Predicate<string> isUpper = IsUpperCase;

bool result = isUpper("hello world!!");

Console.WriteLine(result);

}

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-6jIeuz)

Output:

false

An anonymous method can also be assigned to a Predicate delegate type as shown below.

Example: Predicate delegate with anonymous method

static void Main(string[] args)

{

Predicate<string> isUpper = delegate(string s) { return s.Equals(s.ToUpper());};

bool result = isUpper("hello world!!");

}

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-GjU9b5)

A lambda expression can also be assigned to a Predicate delegate type as shown below.

Example: Predicate delegate with lambda expression

static void Main(string[] args)

{

Predicate<string> isUpper = s => s.Equals(s.ToUpper());

bool result = isUpper("hello world!!");

}

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-e0oBtk)

 Points to Remember:

1. Predicate delegate takes one input parameter and boolean return type.
2. [Anonymous method](https://www.tutorialsteacher.com/csharp/csharp-anonymous-method) and [Lambda expression](https://www.tutorialsteacher.com/linq/linq-lambda-expression) can be assigned to the predicate delegate.

////////////////////////////////////////////////////////////////////////////////////

C# - Object Initializer Syntax

C# 3.0 (.NET 3.5) introduced *Object Initializer Syntax*, a new way to initialize an object of a class or collection. Object initializers allow you to assign values to the fields or properties at the time of creating an object without invoking a constructor.

Example: Object Initializer Syntax

public class Student

{

public int StudentID { get; set; }

public string StudentName { get; set; }

public int Age { get; set; }

public string Address { get; set; }

}

class Program

{

static void Main(string[] args)

{

Student std = new Student() { StudentID = 1,

StudentName = "Bill",

Age = 20,

Address = "New York"

};

}

}

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-4nuEbC)

In the above example, Student class is defined without any constructors. In the Main() method, we have created Student object and assigned values to all or some properties in the curly bracket at the same time. This is called object initializer syntax.

The compiler compiles the above initializer into something like the following.

Example: Object Initializer Syntax at Compile time

Student \_\_student = new Student();

\_\_student.StudentID = 1;

\_\_student.StudentName = "Bill";

\_\_student.Age = 20;

\_\_student.StandardID = 10;

\_\_student.Address = "Test";

Student std = \_\_student;

Collection Initializer Syntax

Collection can be initialized the same way as class objects using collection initializer syntax.

Example: Object initializer Syntax

var student1 = new Student() { StudentID = 1, StudentName = "John" };

var student2 = new Student() { StudentID = 2, StudentName = "Steve" };

var student3 = new Student() { StudentID = 3, StudentName = "Bill" } ;

var student4 = new Student() { StudentID = 3, StudentName = "Bill" };

var student5 = new Student() { StudentID = 5, StudentName = "Ron" };

IList<Student> studentList = new List<Student>() {

student1,

student2,

student3,

student4,

student5

};

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-0ttgkw)

You can also initialize collections and objects at the same time.

Example: Collection initializer Syntax

IList<Student> studentList = new List<Student>() {

new Student() { StudentID = 1, StudentName = "John"} ,

new Student() { StudentID = 2, StudentName = "Steve"} ,

new Student() { StudentID = 3, StudentName = "Bill"} ,

new Student() { StudentID = 3, StudentName = "Bill"} ,

new Student() { StudentID = 4, StudentName = "Ram" } ,

new Student() { StudentID = 5, StudentName = "Ron" }

};

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-t1uClU)

You can also specify null as an element:

Example: Collection initializer Syntax

IList<Student> studentList = new List<Student>() {

new Student() { StudentID = 1, StudentName = "John"} ,

null

};

Advantages of Initializers

* Initializer syntax makes a code more readable, easy to add elements into the collection.
* Useful in multi-threading.

/////////////////////////////////////////////////////////////////////////////////////

Nullable Type in C#

As you know, a value type cannot be assigned a null value. For example, *int i = null* will give you a compile time error.

C# 2.0 introduced nullable types that allow you to assign null to value type variables. You can declare nullable types using Nullable<t> where T is a type.

Example: Nullable type

Nullable<int> i = null;

A nullable type can represent the correct range of values for its underlying value type, plus an additional *null* value. For example, Nullable<int> can be assigned any value from -2147483648 to 2147483647, or a null value.

The Nullable types are instances of System.Nullable<T> struct. Think it as something like the following structure.

Example: Nullable struct

[Serializable]

public struct Nullable<T> where T : struct

{

public bool HasValue { get; }

public T Value { get; }

// other implementation

}

A nullable of type *int* is the same as an ordinary *int* plus a flag that says whether the *int* has a value or not (is null or not). All the rest is compiler magic that treats "null" as a valid value.

Example: HasValue

static void Main(string[] args)

{

Nullable<int> i = null;

if (i.HasValue)

Console.WriteLine(i.Value); // or Console.WriteLine(i)

else

Console.WriteLine("Null");

}

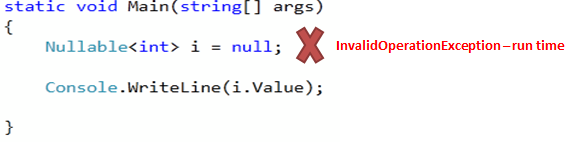
[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-oAqEz3)

Output:

Null

The HasValue returns **true** if the object has been assigned a value; if it has not been assigned any value or has been assigned a null value, it will return **false**.

Accessing the value using NullableType.value will throw a runtime exception if nullable type is null or not assigned any value. For example, i.Value will throw an exception if i is null:

[](https://www.tutorialsteacher.com/Content/images/csharp/nullabletype-error.png)Invalid use of Nullable Type

Use the GetValueOrDefault() method to get an actual value if it is not null and the default value if it is null. For example:

Example: GetValueOrDefault()

static void Main(string[] args)

{

Nullable<int> i = null;

Console.WriteLine(i.GetValueOrDefault());

}

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-r7yeK0)

Shorthand Syntax for Nullable Types

You can use the '?' operator to shorthand the syntax e.g. int?, long? instead of using Nullable<T>.

Example: Shorthand syntax for Nullable types

int? i = null;

double? D = null;

?? Operator

Use the '??' operator to assign a nullable type to a non-nullable type.

Example: ?? operator with Nullable Type

int? i = null;

int j = i ?? 0;

Console.WriteLine(j);

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-4m9elt)

Output:

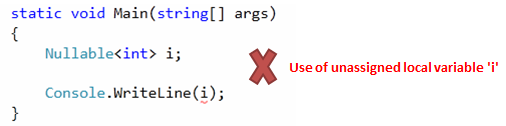
0

In the above example, i is a nullable int and if you assign it to the non-nullable int j then it will throw a runtime exception if i is null. So to mitigate the risk of an exception, we have used the '??' operator to specify that if i is null then assign 0 to j.

Assignment Rules

A nullable type has the same assignment rules as a value type. It must be assigned a value before using it if nullable types are declared in a function as local variables. If it is a field of any class then it will have a null value by default.

For example, the following nullable of int type is declared and used without assigning any value. The compiler will give **"Use of unassigned local variable 'i'"**error:

[](https://www.tutorialsteacher.com/Content/images/csharp/unassigned-nullabletype.png)Unassigned nullable type-error

In the following example, a nullable of int type is a field of the class, so it will not give any error.

Example: Nullable type as Class Field

class MyClass

{

public Nullable<int> i;

}

class Program

{

static void Main(string[] args)

{

MyClass mycls = new MyClass();

if(mycls.i == null)

Console.WriteLine("Null");

}

}

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-bYvkTO)

Output:

Null

Nullable Helper Class

Null is considered to be less than any value. So comparison operators won't work against null. Consider the following example where i is neither less than j, greater than j nor equal to j:

Example: Nullable Type Comparison

static void Main(string[] args)

{

int? i = null;

int j = 10;

if (i < j)

Console.WriteLine("i < j");

else if( i > 10)

Console.WriteLine("i > j");

else if( i == 10)

Console.WriteLine("i == j");

else

Console.WriteLine("Could not compare");

}

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-u1SUrv)

Output:

Could not compare

Nullable static class is a helper class for Nullable types. It provides a compare method to compare nullable types. It also has a GetUnderlyingType method that returns the underlying type argument of nullable types.

Example: Helper Class

static void Main(string[] args)

{

int? i = null;

int j = 10;

if (Nullable.Compare<int>(i, j) < 0)

Console.WriteLine("i < j");

else if (Nullable.Compare<int>(i, j) > 0)

Console.WriteLine("i > j");

else

Console.WriteLine("i = j");

}

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=cs-x7aTXG)

Output:

i < j

Characteristics of Nullable Types

1. Nullable types can only be used with value types.
2. The Value property will throw an InvalidOperationException if value is null; otherwise it will return the value.
3. The HasValue property returns true if the variable contains a value, or false if it is null.
4. You can only use == and != operators with a nullable type. For other comparison use the Nullable static class.
5. Nested nullable types are not allowed. Nullable<Nullable<int>> i; will give a compile time error.

 Points to Remember :

1. Nullable<T> type allows assignment of null to value types.
2. **?** operator is a shorthand syntax for Nullable types.
3. Use value property to get the value of nullable type.
4. Use *HasValue* property to check whether value is assigned to *nullable type* or not.
5. *Static Nullable* class is a helper class to compare nullable types.

////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////

**MVC**

**/////////////////////////////////////**

MVC Architecture

In this section, you will get an overview of MVC architecture. The MVC architectural pattern has existed for a long time in software engineering. All most all the languages use MVC with slight variation, but conceptually it remains the same.

Let's understand the MVC architecture in ASP.NET.

MVC stands for Model, View and Controller. MVC separates application into three components - Model, View and Controller.

**Model**: Model represents shape of the data and business logic. It maintains the data of the application. Model objects retrieve and store model state in a database.

Model is a data and business logic.

**View**: View is a user interface. View display data using model to the user and also enables them to modify the data.

View is a User Interface.

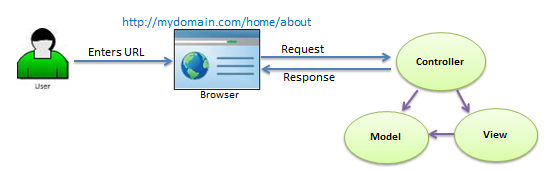
**Controller**: Controller handles the user request. Typically, user interact with View, which in-turn raises appropriate URL request, this request will be handled by a controller. The controller renders the appropriate view with the model data as a response.

Controller is a request handler.

The following figure illustrates the interaction between Model, View and Controller.

[](https://www.tutorialsteacher.com/Content/images/mvc/mvc-architecture.png)MVC Architecture

The following figure illustrates the flow of the user's request in ASP.NET MVC.

[](https://www.tutorialsteacher.com/Content/images/mvc/request-handling-in-mvc.png)Request/Response in MVC Architecture

As per the above figure, when the user enters a URL in the browser, it goes to the server and calls appropriate controller. Then, the Controller uses the appropriate View and Model and creates the response and sends it back to the user. We will see the details of the interaction in the next few sections.

Visit MSDN to learn [MVC](https://msdn.microsoft.com/en-us/library/dd381412(v=vs.108).aspx) in detail.

 Points to Remember :

1. MVC stands for Model, View and Controller.
2. Model is responsible for maintaining application data and business logic.
3. View is a user interface of the application, which displays the data.
4. Controller handles user's requests and renders appropriate View with Model data.

**////////////////////////////////////////**

ASP.NET MVC Version History

Microsoft had introduced ASP.NET MVC in .Net 3.5, since then lots of new features have been added.

The following table list brief history of ASP.NET MVC.

| MVC Version | Visual Studio | .Net Version | Release date | Features |
| --- | --- | --- | --- | --- |
| MVC 1.0 | VS2008 | .Net 3.5 | 13-Mar-2009 | * MVC architecture with webform engine * Routing * HTML Helpers * Ajax Helpers * Auto binding |
| MVC 2.0 | VS 2008, | .Net 3.5/4.0 | 10-Mar-2010 | * Area * Asynchronous controller * Html helper methods with lambda expression * DataAnnotations attributes * Client side validation * Custom template * Scaffolding |
| MVC 3.0 | VS 2010 | .Net 4.0 | 13-Jan-2011 | * Unobtrusive javascript validation * Razor view engine * Global filters * Remote validation * Dependency resolver for IoC * ViewBag |
| MVC 4.0 | VS 2010 SP1, VS 2012 | .NET 4.0/4.5 | 15-Aug-2012 | * Mobile project template * Bundling and minification * Support for Windows Azure SDK |
| MVC 5.0 | VS 2013 | .NET 4.5 | 17-oct-2013 | * Authentication filters * Bootstrap support * New scaffolding items * ASP.Net Identity |
| **MVC 5.2** - Current | VS 2013 | .NET 4.5 | 28-Aug-2014 | * Attribute based routing * bug fixes and minor features upate |

**//////////////////////////////////////**

Create First ASP.NET MVC Application

In this section, we will create a new MVC web application using Visual Studio and understand the basic building blocks of the ASP.NET MVC Application.

First, setup a development environment to develop an ASP.NET MVC 5 application.

Setup Development Environment

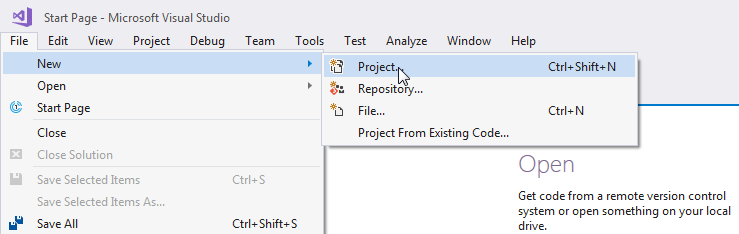
You can develop ASP.NET MVC application with appropriate version of Visual Studio and .NET framework, as you have seen in the previous section of version history.

Here, we will use MVC v5.2, Visual Studio 2017 Community edition and .NET framework 4.6 to create our first MVC application.

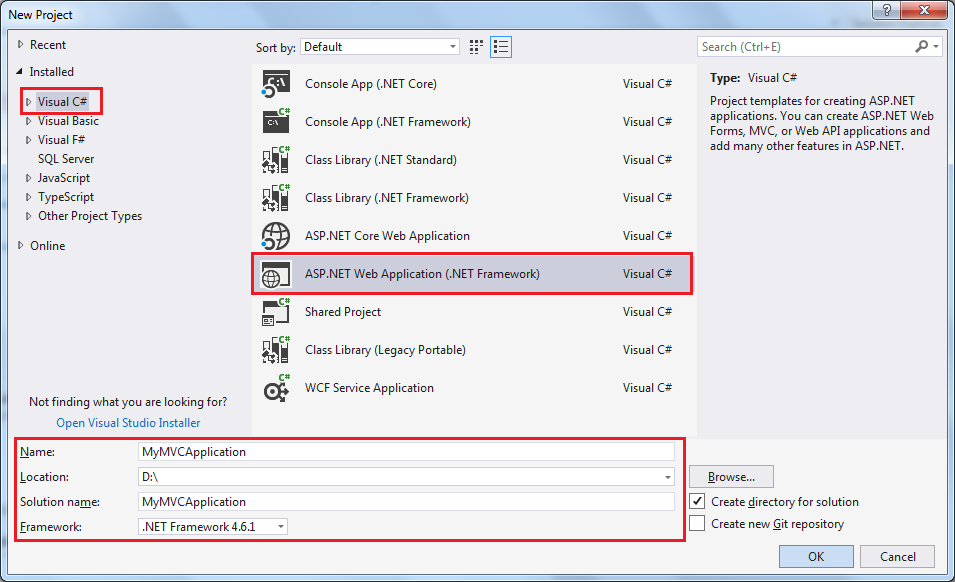
Download the latest version of Visual Studio from [https://visualstudio.microsoft.com/downloads](https://visualstudio.microsoft.com/downloads/).

Create first simple MVC application

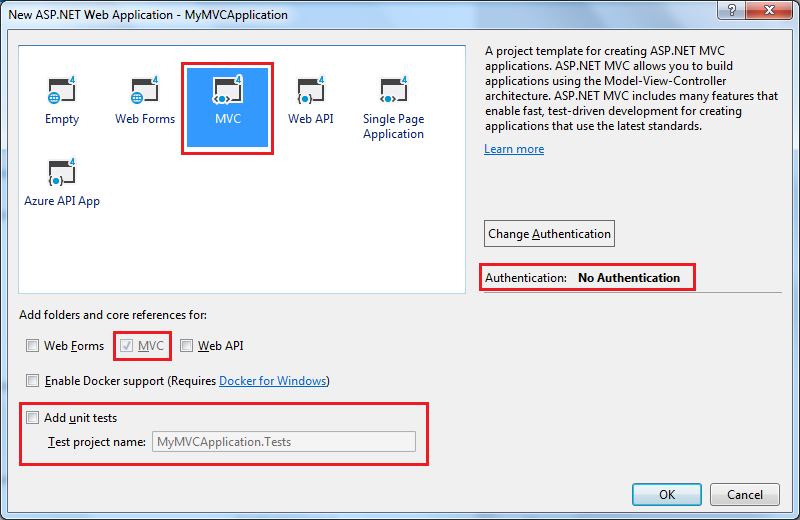
Open a Visual Studio 2017 Community edition and select **File menu** -> **New** -> **Project** as shown below.

[](https://www.tutorialsteacher.com/Content/images/mvc/create-mvc-project.png)Create a New Project in Visual Studio

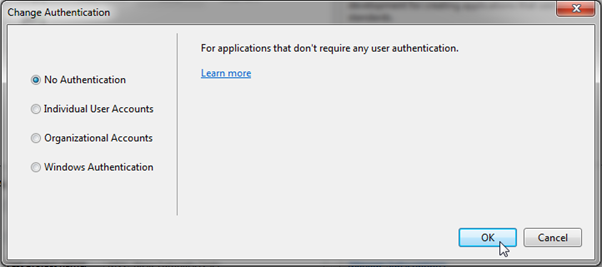
From the **New Project** dialog as shown below, expand Visual C# node and select **Web** in the left pane, and then select **ASP.NET Web Application (.NET Framework)** in the middle pane. Enter the name of your project MyMVCApplication. (You can give any appropriate name for your application). Also, you can change the location of the MVC application by clicking on **Browse..** button. Finally, click **OK.**

[](https://www.tutorialsteacher.com/Content/images/mvc/MVC-template.png)Create MVC Project in Visual Studio

From the **New ASP.NET Web Application** dialog, select MVC (if not selected already) as shown below.

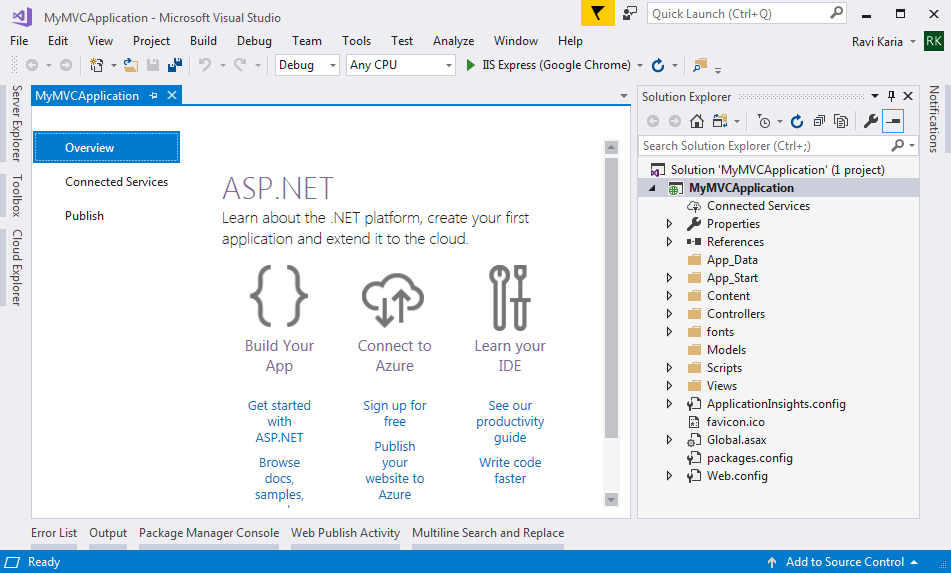
[](https://www.tutorialsteacher.com/Content/images/mvc/Create-MVC-Project2.png)Create MVC Application

You can also change the authentication by clicking on **Change Authentication** button. You can select appropriate authentication mode for your application as shown below.

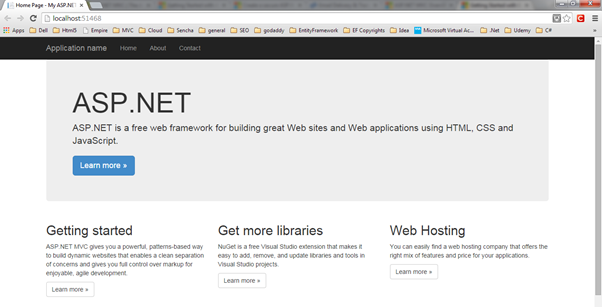
[](https://www.tutorialsteacher.com/Content/images/mvc/first-mvc-app4.png)Select Authenctication Type

Here, we are keeping the default authentication for our application which is No Authentication. Click **OK** to continue.

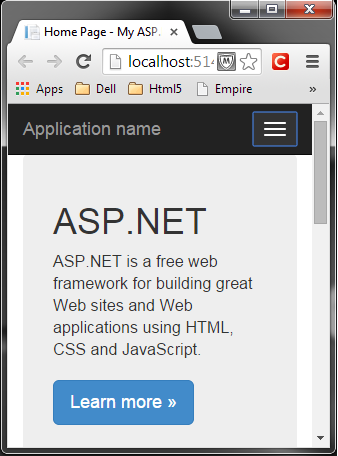
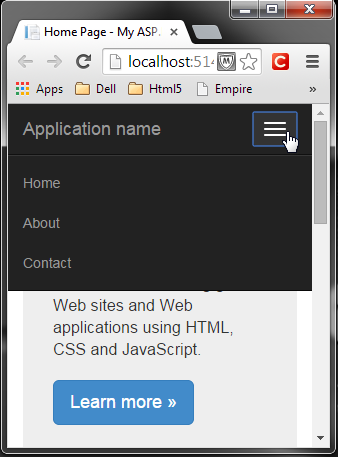
Wait for some time till Visual Studio creates a simple MVC project using default template as shown below.

[](https://www.tutorialsteacher.com/Content/images/mvc/MVC-application.png)First MVC Application

Now, press F5 to run the project in debug mode or Ctrl + F5 to run the project without debugging. It will open home page in the browser as shown below.

[](https://www.tutorialsteacher.com/Content/images/mvc/first-mvc-app7.png)Run MVC Application

MVC 5 project includes JavaScript and CSS files of bootstrap 3.0 by default. So you can create responsive web pages. This responsive UI will change its look and feel based on the screen size of the different devices. For example, top menu bar will be changed in the mobile devices as shown below.

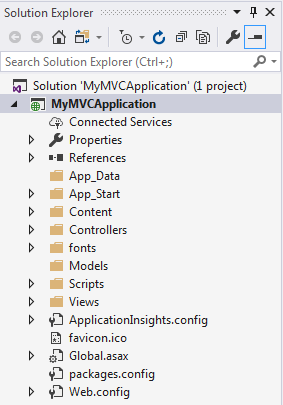
[](https://www.tutorialsteacher.com/Content/images/mvc/first-mvc-app8.png)Responsive MVC Application[](https://www.tutorialsteacher.com/Content/images/mvc/first-mvc-app9.png)Responsive MVC Application

So in this way, you can create your first MVC 5 application using Visual Studio 2013 for Web.

**//////////////////////////////////////**

ASP.NET MVC Folder Structure

We have created our first MVC 5 application in the previous section. Visual Studio creates the following folder structure for MVC application by default.

[](https://www.tutorialsteacher.com/Content/images/mvc/mvc-folder-structure.png)MVC Folder Structure

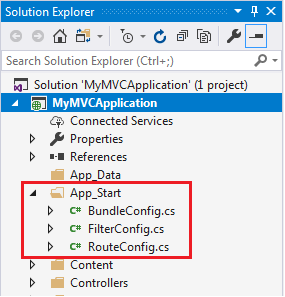
Let's see significance of each folder.

App\_Data

App\_Data folder can contain application data files like LocalDB, .mdf files, xml files and other data related files. IIS will never serve files from App\_Data folder.

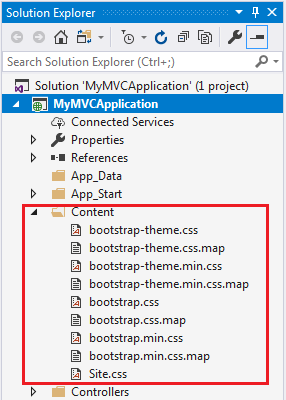
App\_Start

App\_Start folder can contain class files which will be executed when the application starts. Typically, these would be config files like AuthConfig.cs, BundleConfig.cs, FilterConfig.cs, RouteConfig.cs etc. MVC 5 includes BundleConfig.cs, FilterConfig.cs and RouteConfig.cs by default. We will see significance of these files later.

[](https://www.tutorialsteacher.com/Content/images/mvc/appstart.png)App\_Start Folder

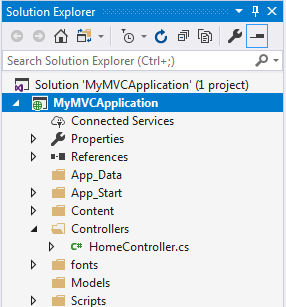
Content

Content folder contains static files like css files, images and icons files. MVC 5 application includes bootstrap.css, bootstrap.min.css and Site.css by default.

[](https://www.tutorialsteacher.com/Content/images/mvc/contentfolder.png)Content Folder

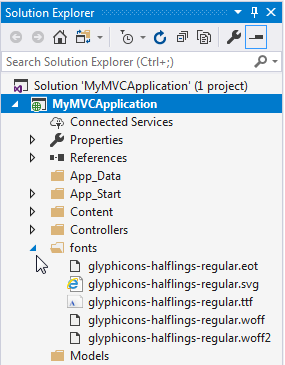
Controllers

Controllers folder contains class files for the controllers. Controllers handles users' request and returns a response. MVC requires the name of all controller files to end with "Controller". You will learn about the controller in the next section.

[](https://www.tutorialsteacher.com/Content/images/mvc/controllerfolder.png)Controller Folder

fonts

Fonts folder contains custom font files for your application.

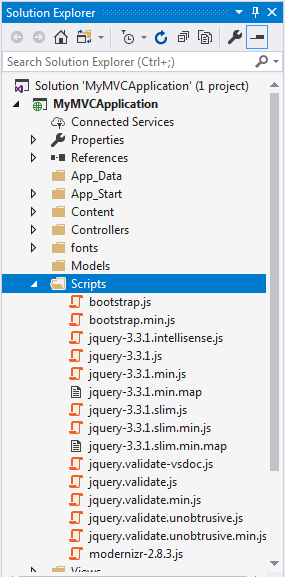
[](https://www.tutorialsteacher.com/content/images/mvc/fontsfolder.png)Fonts folder

Models

Models folder contains model class files. Typically model class includes public properties, which will be used by application to hold and manipulate application data.

Scripts

Scripts folder contains JavaScript or VBScript files for the application. MVC 5 includes javascript files for bootstrap, jquery 1.10 and modernizer by default.

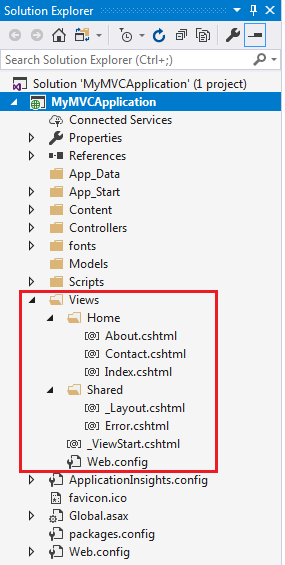
[](https://www.tutorialsteacher.com/Content/images/mvc/scriptsfolder.png)Scripts Folder

Views

Views folder contains html files for the application. Typically view file is a .cshtml file where you write html and C# or VB.NET code.

Views folder includes separate folder for each controllers. For example, all the .cshtml files, which will be rendered by HomeController will be in View > Home folder.

Shared folder under View folder contains all the views which will be shared among different controllers e.g. layout files.

[](https://www.tutorialsteacher.com/Content/images/mvc/viewfolder.png)View Folder

Additionally, MVC project also includes following configuration files:

Global.asax

Global.asax allows you to write code that runs in response to application level events, such as Application\_BeginRequest, application\_start, application\_error, session\_start, session\_end etc.

Packages.config

Packages.config file is managed by NuGet to keep track of what packages and versions you have installed in the application.

Web.config

Web.config file contains application level configurations.

**//////////////////////////////////////**

Routing in MVC

In the ASP.NET Web Forms application, every URL must match with a specific .aspx file. For example, a URL http://domain/studentsinfo.aspx must match with the file studentsinfo.aspx that contains code and markup for rendering a response to the browser.

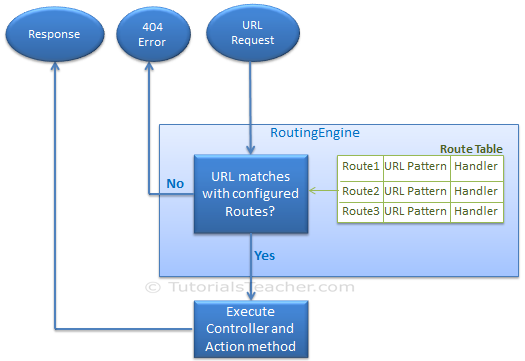
Routing is not specific to MVC framework. It can be used with ASP.NET Webform application or MVC application.

ASP.NET introduced Routing to eliminate needs of mapping each URL with a physical file. Routing enable us to define URL pattern that maps to the request handler. This request handler can be a file or class. In ASP.NET Webform application, request handler is .aspx file and in MVC, it is Controller class and Action method. For example, http://domain/students can be mapped to http://domain/studentsinfo.aspx in ASP.NET Webforms and the same URL can be mapped to Student Controller and Index action method in MVC.

Route

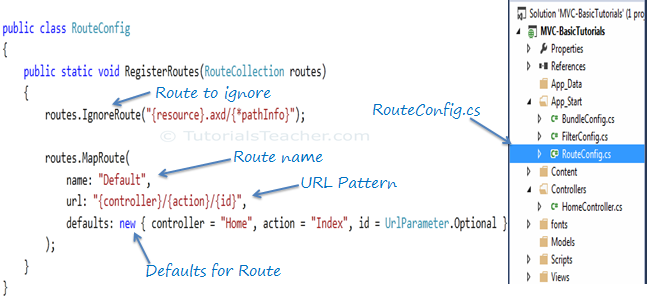
Route defines the URL pattern and handler information. All the configured routes of an application stored in RouteTable and will be used by Routing engine to determine appropriate handler class or file for an incoming request.

The following figure illustrates the Routing process.

[](https://www.tutorialsteacher.com/Content/images/mvc/routing-process.png)Routing in MVC

Configure a Route

Every MVC application must configure (register) at least one route, which is configured by MVC framework by default. You can register a route in **RouteConfig** class, which is in RouteConfig.cs under **App\_Start** folder. The following figure illustrates how to configure a Route in the RouteConfig class .

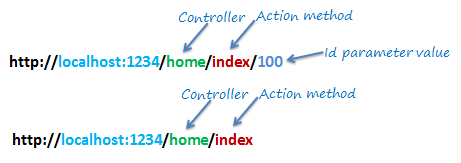
[](https://www.tutorialsteacher.com/Content/images/mvc/routeconfig.png)Configure Route in MVC

As you can see in the above figure, the route is configured using the MapRoute() extension method of RouteCollection, where name is "Default", url pattern is *"{controller}/{action}/{id}"* and defaults parameter for controller, action method and id parameter. Defaults specifies which controller, action method or value of id parameter should be used if they do not exist in the incoming request URL.

The same way, you can configure other routes using MapRoute method of RouteCollection. This RouteCollection is actually a property of [RouteTable](https://msdn.microsoft.com/en-us/library/system.web.routing.routetable(v=vs.110).aspx" \t "_blank) class.

URL Pattern

The URL pattern is considered only after domain name part in the URL. For example, the URL pattern *"{controller}/{action}/{id}"* would look like localhost:1234/{controller}/{action}/{id}. Anything after "localhost:1234/" would be considered as controller name. The same way, anything after controller name would be considered as action name and then value of id parameter.

[](https://www.tutorialsteacher.com/Content/images/mvc/url-routing.png)Routing in MVC

If the URL doesn't contain anything after domain name then the default controller and action method will handle the request. For example, http://localhost:1234 would be handled by HomeController and Index method as configured in the defaults parameter.

The following table shows which Controller, Action method and Id parameter would handle different URLs considering above default route.

| URL | Controller | Action | Id |
| --- | --- | --- | --- |
| http://localhost/home | HomeController | Index | null |
| http://localhost/home/index/123 | HomeController | Index | 123 |
| http://localhost/home/about | HomeController | About | null |
| http://localhost/home/contact | HomeController | Contact | null |
| http://localhost/student | StudentController | Index | null |
| http://localhost/student/edit/123 | StudentController | Edit | 123 |

Multiple Routes

You can also configure a custom route using MapRoute extension method. You need to provide at least two parameters in MapRoute, route name and url pattern. The Defaults parameter is optional.

You can register multiple custom routes with different names. Consider the following example where we register "Student" route.

Example: Custom Routes

public class RouteConfig

{

public static void RegisterRoutes(RouteCollection routes)

{

routes.IgnoreRoute("{resource}.axd/{\*pathInfo}");

routes.MapRoute(

name: "Student",

url: "students/{id}",

defaults: new { controller = "Student", action = "Index"}

);

routes.MapRoute(

name: "Default",

url: "{controller}/{action}/{id}",

defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }

);

}

}

As shown in the above code, URL pattern for the Student route is *students/{id}*, which specifies that any URL that starts with domainName/students, must be handled by StudentController. Notice that we haven't specified {action} in the URL pattern because we want every URL that starts with student should always use Index action of StudentController. We have specified default controller and action to handle any URL request which starts from domainname/students.

MVC framework evaluates each route in sequence. It starts with first configured route and if incoming url doesn't satisfy the URL pattern of the route then it will evaluate second route and so on. In the above example, routing engine will evaluate Student route first and if incoming url doesn't starts with /students then only it will consider second route which is default route.

The following table shows how different URLs will be mapped to Student route:

| URL | Controller | Action | Id |
| --- | --- | --- | --- |
| http://localhost/student/123 | StudentController | Index | 123 |
| http://localhost/student/index/123 | StudentController | Index | 123 |
| http://localhost/student?Id=123 | StudentController | Index | 123 |

Route Constraints

You can also apply restrictions on the value of parameter by configuring route constraints. For example, the following route applies a restriction on id parameter that the value of an id must be numeric.

Example: Route Constraints

routes.MapRoute(

name: "Student",

url: "student/{id}/{name}/{standardId}",

defaults: new { controller = "Student", action = "Index", id = UrlParameter.Optional, name = UrlParameter.Optional, standardId = UrlParameter.Optional },

constraints: new { id = @"\d+" }

);

So if you give non-numeric value for id parameter then that request will be handled by another route or, if there are no matching routes then *"The resource could not be found"* error will be thrown.

Register Routes

Now, after configuring all the routes in RouteConfig class, you need to register it in the Application\_Start() event in the Global.asax. So that it includes all your routes into RouteTable.

Example: Route Registration

public class MvcApplication : System.Web.HttpApplication

{

protected void Application\_Start()

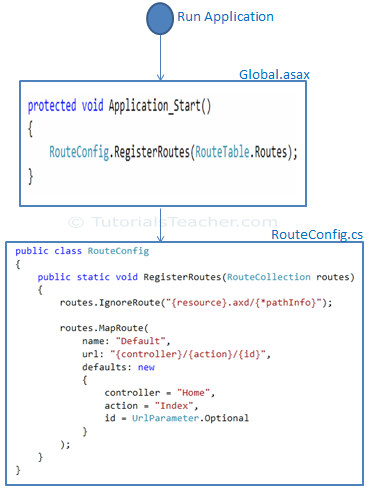
{

RouteConfig.RegisterRoutes(RouteTable.Routes);

}

}

The following figure illustrate Route registration process.

[](https://www.tutorialsteacher.com/Content/images/mvc/Route-configuration-process.png)Register Route

Thus, routing plays important role in MVC framework.

 Points to Remember :

1. Routing plays important role in MVC framework. Routing maps URL to physical file or class (controller class in MVC).
2. Route contains URL pattern and handler information. URL pattern starts after domain name.
3. Routes can be configured in RouteConfig class. Multiple custom routes can also be configured.
4. Route constraints applies restrictions on the value of parameters.
5. Route must be registered in Application\_Start event in Global.ascx.cs file.

**//////////////////////////////////////**

Controller

In this section, you will learn about the Controller in ASP.NET MVC.

The Controller in MVC architecture handles any incoming URL request. Controller is a class, derived from the base class System.Web.Mvc.Controller. Controller class contains public methods called **Action** methods. Controller and its action method handles incoming browser requests, retrieves necessary model data and returns appropriate responses.

In ASP.NET MVC, every controller class name must end with a word "Controller". For example, controller for home page must be HomeController and controller for student must be StudentController. Also, every controller class must be located in Controller folder of MVC folder structure.

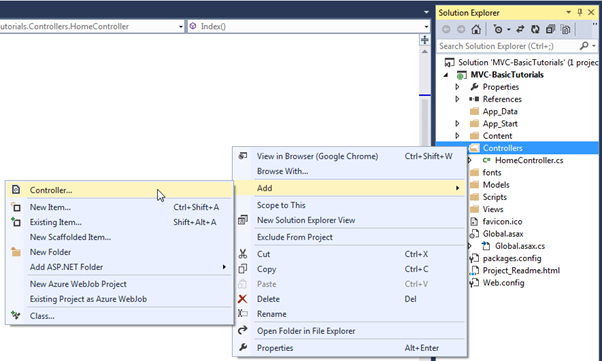
Adding a New Controller

Now, let's add a new empty controller in our MVC application in Visual Studio.

MVC will throw "The resource cannot be found" error when you do not append "Controller" to the controller class name.

In the previous section we learned how to create our first MVC application, which in turn created a default HomeController. Here, we will create a new StudentController.

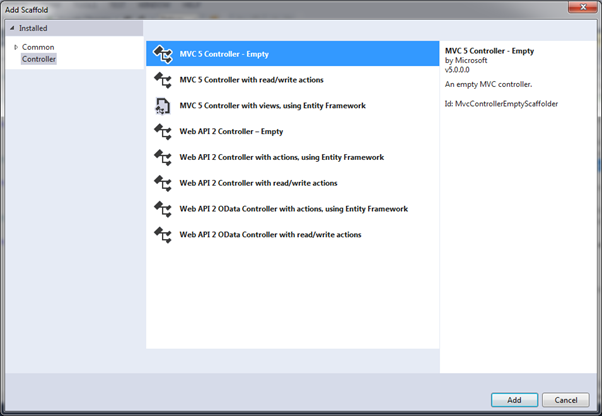
In the Visual Studio, right click on the Controller folder -> select **Add** -> click on **Controller..**

[](https://www.tutorialsteacher.com/Content/images/mvc/mvc-controller-1.png)Add New Controller

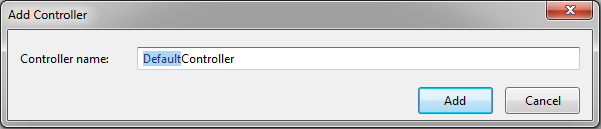
This opens Add Scaffold dialog as shown below.

 Note:

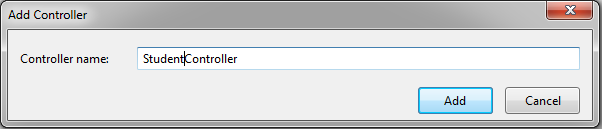
Scaffolding is an automatic code generation framework for ASP.NET web applications. Scaffolding reduces the time taken to develop a controller, view etc. in MVC framework. You can develop a customized scaffolding template using T4 templates as per your architecture and coding standard.

[](https://www.tutorialsteacher.com/Content/images/mvc/mvc-controller-2.png)Adding Controller

Add Scaffold dialog contains different templates to create a new **controller**. We will learn about other templates later. For now, select **"MVC 5 Controller - Empty"** and click **Add**. It will open Add Controller dialog as shown below

[](https://www.tutorialsteacher.com/Content/images/mvc/mvc-controller-3.png)Adding Controller

In the Add Controller dialog, enter the name of the controller. Remember, controller name must end with Controller. Let's enter StudentController and click **Add**.

[](https://www.tutorialsteacher.com/Content/images/mvc/mvc-controller-4.png)Adding Controller

This will create StudentController class with Index method in StudentController.cs file under Controllers folder, as shown below.

Example: Controller

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace MVC\_BasicTutorials.Controllers

{

public class StudentController : Controller

{

// GET: Student

public ActionResult Index()

{

return View();

}

}

}

As you can see above, the StudentController class is derived from Controller class. Every controller in MVC must derived from this abstract Controller class. This base Controller class contains helper methods that can be used for various purposes.

Now, we will return a dummy string from Index action method of above StudentController. Changing the return type of Index method from ActionResult to string and returning dummy string is shown below. You will learn about ActionResult in the next section.

Example: Controller

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace MVC\_BasicTutorials.Controllers

{

public class StudentController : Controller

{

// GET: Student

public string Index()

{

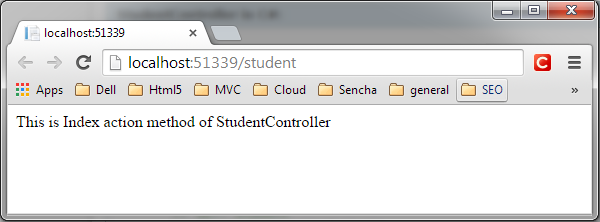
return "This is Index action method of StudentController";

}

}

}

We have already seen in the routing section that the URL request *http://localhost/student* or *http://localhost/student/index* is handled by the Index() method of StudentController class, shown above. So let's invoke it from the browser and you will see the following page in the browser.

[](https://www.tutorialsteacher.com/Content/images/mvc/mvc-controller-5.png)Controller

 Points to Remember :

1. A Controller handles incomming URL requests. MVC routing sends request to appropriate controller and action method based on URL and configured Routes.
2. All the public methods in the Controller class are called Action methods.
3. A Controller class must be derived from System.Web.Mvc.Controller class.
4. A Controller class name must end with "Controller".
5. New controller can be created using different scaffolding templates. You can create custom scaffolding template also.

**//////////////////////////////////////**

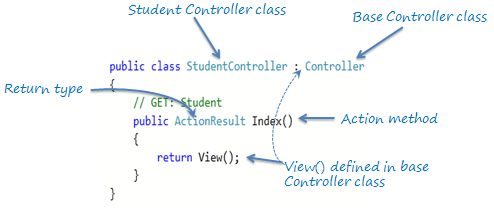
Action method

In this section, you will learn about the action method of controller class.

All the public methods of a Controller class are called Action methods. They are like any other normal methods with the following restrictions:

1. Action method must be public. It cannot be private or protected
2. Action method cannot be overloaded
3. Action method cannot be a static method.

The following is an example of Index action method of StudentController

[](https://www.tutorialsteacher.com/Content/images/mvc/action-method.png)Action Method

As you can see in the above figure, Index method is a public method and it returns ActionResult using the View() method. The View() method is defined in the Controller base class, which returns the appropriate ActionResult.

Default Action Method

Every controller can have default action method as per configured route in RouteConfig class. By default, Index is a default action method for any controller, as per configured default root as shown below.

Default Route:

routes.MapRoute(

name: "Default",

url: "{controller}/{action}/{id}/{name}",

defaults: new { controller = "Home",

action = "Index",

id = UrlParameter.Optional

});

However, you can change the default action name as per your requirement in RouteConfig class.

ActionResult

MVC framework includes various result classes, which can be return from an action methods. There result classes represent different types of responses such as html, file, string, json, javascript etc. The following table lists all the result classes available in ASP.NET MVC.

| Result Class | Description |
| --- | --- |
| ViewResult | Represents HTML and markup. |
| EmptyResult | Represents No response. |
| ContentResult | Represents string literal. |
| FileContentResult/ FilePathResult/ FileStreamResult | Represents the content of a file |
| JavaScriptResult | Represent a JavaScript script. |
| JsonResult | Represent JSON that can be used in AJAX |
| RedirectResult | Represents a redirection to a new URL |
| RedirectToRouteResult | Represent another action of same or other controller |
| PartialViewResult | Returns HTML from Partial view |
| HttpUnauthorizedResult | Returns HTTP 403 status |

The ActionResult class is a base class of all the above result classes, so it can be return type of action methods which returns any type of result listed above. However, you can specify appropriate result class as a return type of action method.

The Index() method of StudentController in the above figure uses View() method to return ViewResult (which is derived from ActionResult). The View() method is defined in base Controller class. It also contains different methods, which automatically returns particular type of result as shown in the below table.

| Result Class | Description | Base Controller Method |
| --- | --- | --- |
| ViewResult | Represents HTML and markup. | View() |
| EmptyResult | Represents No response. |  |
| ContentResult | Represents string literal. | Content() |
| FileContentResult, FilePathResult, FileStreamResult | Represents the content of a file | File() |
| JavaScriptResult | Represent a JavaScript script. | JavaScript() |
| JsonResult | Represent JSON that can be used in AJAX | Json() |
| RedirectResult | Represents a redirection to a new URL | Redirect() |
| RedirectToRouteResult | Represent another action of same or other controller | RedirectToRoute() |
| PartialViewResult | Returns HTML | PartialView() |
| HttpUnauthorizedResult | Returns HTTP 403 status |  |

As you can see in the above table, View method returns ViewResult, Content method returns string, File method returns content of a file and so on. Use different methods mentioned in the above table, to return different types of results from an action method.

Action Method Parameters

Every action methods can have input parameters as normal methods. It can be primitive data type or complex type parameters as shown in the below example.

Example: Action method parameters

[HttpPost]

public ActionResult Edit(Student std)

{

// update student to the database

return RedirectToAction("Index");

}

[HttpDelete]

public ActionResult Delete(int id)

{

// delete student from the database whose id matches with specified id

return RedirectToAction("Index");

}

Please note that action method paramter can be [Nullable Type](https://www.tutorialsteacher.com/csharp/csharp-nullable-types).

By default, the values for action method parameters are retrieved from the request's data collection. The data collection includes name/values pairs for form data or query string values or cookie values. Model binding in ASP.NET MVC automatically maps the URL query string or form data collection to the action method parameters if both names are matching. Visit [model binding](https://www.tutorialsteacher.com/mvc/model-binding-in-asp.net-mvc) section for more information on it.

 Points to Remember :

1. All the public methods in the Controller class are called Action methods.
2. Action method has following restrictions.  
       - Action method must be public. It cannot be private or protected.  
       - Action method cannot be overloaded.  
       - Action method cannot be a static method.
3. ActionResult is a base class of all the result type which returns from Action method.
4. Base Controller class contains methods that returns appropriate result type e.g. View(), Content(), File(), JavaScript() etc.
5. Action method can include [Nullable](https://www.tutorialsteacher.com/csharp/csharp-nullable-types) type parameters.

**//////////////////////////////////////**

Action Selectors

Action selector is the attribute that can be applied to the action methods. It helps routing engine to select the correct action method to handle a particular request. MVC 5 includes the following action selector attributes:

1. ActionName
2. NonAction
3. ActionVerbs

ActionName

ActionName attribute allows us to specify a different action name than the method name. Consider the following example.

Example: ActionName

public class StudentController : Controller

{

public StudentController()

{

}

[ActionName("find")]

public ActionResult GetById(int id)

{

// get student from the database

return View();

}

}

In the above example, we have applied ActioName("find") attribute to GetById action method. So now, action name is "find" instead of "GetById". This action method will be invoked on *http://localhost/student/find/1* request instead of *http://localhost/student/getbyid/1* request.

NonAction

NonAction selector attribute indicates that a public method of a Controller is not an action method. Use NonAction attribute when you want public method in a controller but do not want to treat it as an action method.

For example, the GetStudent() public method cannot be invoked in the same way as action method in the following example.

Example: NonAction

public class StudentController : Controller

{

public StudentController()

{

}

[NonAction]

public Student GetStudent(int id)

{

return studentList.Where(s => s.StudentId == id).FirstOrDefault();

}

}

 Points to Remember :

1. MVC framework routing engine uses Action Selectors attributes to determine which action method to invoke.
2. Three action selectors attributes are available in MVC 5  
      - ActionName  
      - NonAction  
      - ActionVerbs
3. ActionName attribute is used to specify different name of action than method name.
4. NonAction attribute marks the public method of controller class as non-action method. It cannot be invoked.

**//////////////////////////////////////**

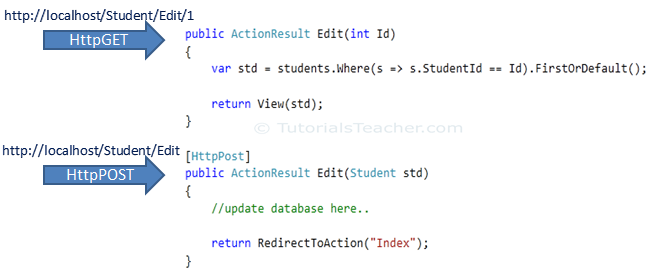
ActionVerbs

In this section, you will learn about the ActionVerbs selectors attribute.

The ActionVerbs selector is used when you want to control the selection of an action method based on a Http request method. For example, you can define two different action methods with the same name but one action method responds to an HTTP Get request and another action method responds to an HTTP Post request.

MVC framework supports different ActionVerbs, such as HttpGet, HttpPost, HttpPut, HttpDelete, HttpOptions & HttpPatch. You can apply these attributes to action method to indicate the kind of Http request the action method supports. If you do not apply any attribute then it considers it a GET request by default.

The following figure illustrates the HttpGET and HttpPOST action verbs.

[](https://www.tutorialsteacher.com/Content/images/mvc/actionverbs.png)ActionVerbs

The following table lists the usage of http methods:

| **Http method** | **Usage** |
| --- | --- |
| GET | To retrieve the information from the server. Parameters will be appended in the query string. |
| POST | To create a new resource. |
| PUT | To update an existing resource. |
| HEAD | Identical to GET except that server do not return message body. |
| OPTIONS | OPTIONS method represents a request for information about the communication options supported by web server. |
| DELETE | To delete an existing resource. |
| PATCH | To full or partial update the resource. |

Visit [W3.org](https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html) for more information on Http Methods.

The following example shows different action methods supports different ActionVerbs:

Example: ActionVerbs

public class StudentController : Controller

{

public ActionResult Index()

{

return View();

}

[HttpPost]

public ActionResult PostAction()

{

return View("Index");

}

[HttpPut]

public ActionResult PutAction()

{

return View("Index");

}

[HttpDelete]

public ActionResult DeleteAction()

{

return View("Index");

}

[HttpHead]

public ActionResult HeadAction()

{

return View("Index");

}

[HttpOptions]

public ActionResult OptionsAction()

{

return View("Index");

}

[HttpPatch]

public ActionResult PatchAction()

{

return View("Index");

}

}

You can also apply multiple http verbs using AcceptVerbs attribute. GetAndPostAction method supports both, GET and POST ActionVerbs in the following example:

Example: AcceptVerbs

[AcceptVerbs(HttpVerbs.Post | HttpVerbs.Get)]

public ActionResult GetAndPostAction()

{

return RedirectToAction("Index");

}

 Points to Remember :

1. ActionVerbs are another Action Selectors which selects an action method based on request methods e.g POST, GET, PUT etc.
2. Multiple action methods can have same name with different action verbs. Method overloading rules are applicable.
3. Multiple action verbs can be applied to a single action method using AcceptVerbs attribute.

**//////////////////////////////////////**

Model in ASP.NET MVC

In this section, you will learn about the Model in ASP.NET MVC framework.

Model represents domain specific data and business logic in MVC architecture. It maintains the data of the application. Model objects retrieve and store model state in the persistance store like a database.

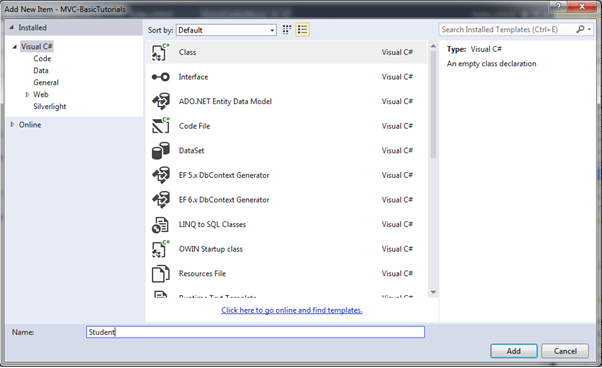
Model class holds data in public properties. All the Model classes reside in the Model folder in MVC folder structure.

Let's see how to add model class in ASP.NET MVC.

Adding a Model

Open our first MVC project created in previous step in the Visual Studio. Right click on Model folder -> Add -> click on Class..

In the Add New Item dialog box, enter class name 'Student' and click **Add**.

[](https://www.tutorialsteacher.com/Content/images/mvc/create-model.png)Create Model Class

This will add new Student class in model folder. Now, add Id, Name, Age properties as shown below.

Example: Model class

namespace MVC\_BasicTutorials.Models

{

public class Student

{

public int StudentId { get; set; }

public string StudentName { get; set; }

public int Age { get; set; }

}

}

So in this way, you can create a model class which you can use in View. You will learn how to implement validations using model later.

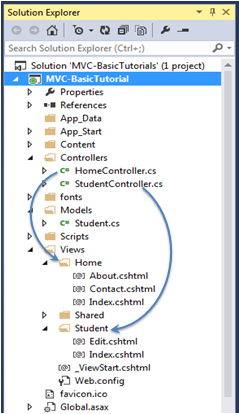
**//////////////////////////////////////**

View in ASP.NET MVC

In this section, you will learn about the View in ASP.NET MVC framework.

View is a user interface. View displays data from the model to the user and also enables them to modify the data.

ASP.NET MVC views are stored in **Views** folder. Different action methods of a single controller class can render different views, so the Views folder contains a separate folder for each controller with the same name as controller, in order to accommodate multiple views. For example, views, which will be rendered from any of the action methods of HomeController, resides in Views > Home folder. In the same way, views which will be rendered from StudentController, will resides in Views > Student folder as shown below.

[](https://www.tutorialsteacher.com/Content/images/mvc/view-folder-for-controllers.png)View folders for Controllers

 Note:

Shared folder contains views, layouts or partial views which will be shared among multiple views.

Razor View Engine

Microsoft introduced the Razor view engine and packaged with MVC 3. You can write a mix of html tags and server side code in razor view. Razor uses @ character for server side code instead of traditional <% %>. You can use C# or Visual Basic syntax to write server side code inside razor view. Razor view engine maximize the speed of writing code by minimizing the number of characters and keystrokes required when writing a view. Razor views files have .cshtml or vbhtml extension.

ASP.NET MVC supports following types of view files:

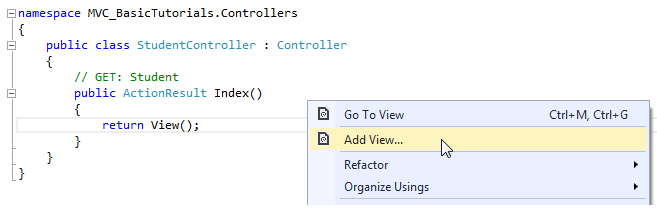
| **View file extension** | **Description** |
| --- | --- |
| .cshtml | C# Razor view. Supports C# with html tags. |
| .vbhtml | Visual Basic Razor view. Supports Visual Basic with html tags. |
| .aspx | ASP.Net web form |
| .ascx | ASP.NET web control |

Learn [Razor syntax](https://www.tutorialsteacher.com/mvc/razor-syntax) in the next section. Let's see how to create a new view using Visual Studio 2013 for Web with MVC 5.

Create New View

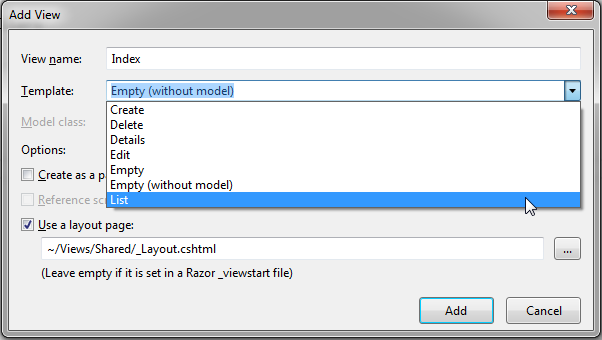
We have already created StudentController and Student model in the previous section. Now, let's create a Student view and understand how to use model into view.

We will create a view, which will be rendered from Index method of StudentContoller. So, open a StudentController class -> right click inside Index method -> click **Add View..**

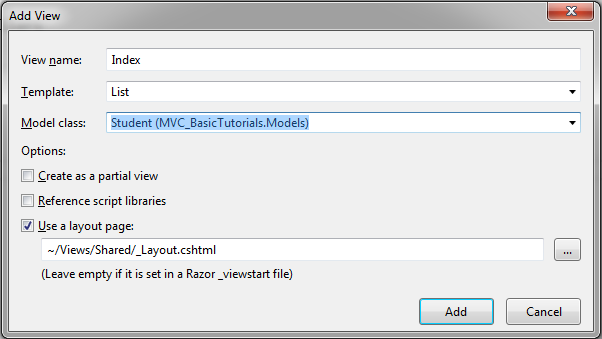
[](https://www.tutorialsteacher.com/Content/images/mvc/add-view-1.png)Create a View

In the Add View dialogue box, keep the view name as Index. It's good practice to keep the view name the same as the action method name so that you don't have to specify view name explicitly in the action method while returning the view.

Select the scaffolding template. Template dropdown will show default templates available for Create, Delete, Details, Edit, List or Empty view. Select "List" template because we want to show list of students in the view.

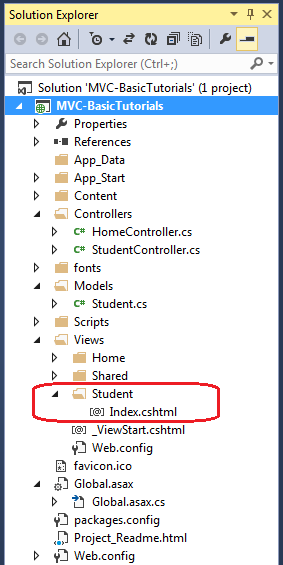
[](https://www.tutorialsteacher.com/Content/images/mvc/addview2.png)View

Now, select Student from the Model class dropdrown. Model class dropdown automatically displays the name of all the classes in the Model folder. We have already created Student Model class in the previous section, so it would be included in the dropdown.

[](https://www.tutorialsteacher.com/Content/images/mvc/addview3.png)View

Check "Use a layout page" checkbox and select \_Layout.cshtml page for this view and then click **Add** button. We will see later what is layout page but for now think it like a master page in MVC.

This will create Index view under View -> Student folder as shown below:

[](https://www.tutorialsteacher.com/Content/images/mvc/addview4.png)View

The following code snippet shows an Index.cshtml created above.

Views\Student\Index.cshtml:

@model IEnumerable<MVC\_BasicTutorials.Models.Student>

@{

ViewBag.Title = "Index";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h2>Index</h2>

<p>

@Html.ActionLink("Create New", "Create")

</p>

<table class="table">

<tr>

<th>

@Html.DisplayNameFor(model => model.StudentName)

</th>

<th>

@Html.DisplayNameFor(model => model.Age)

</th>

<th></th>

</tr>

@foreach (var item in Model) {

<tr>

<td>

@Html.DisplayFor(modelItem => item.StudentName)

</td>

<td>

@Html.DisplayFor(modelItem => item.Age)

</td>

<td>

@Html.ActionLink("Edit", "Edit", new { id=item.StudentId }) |

@Html.ActionLink("Details", "Details", new { id=item.StudentId }) |

@Html.ActionLink("Delete", "Delete", new { id = item.StudentId })

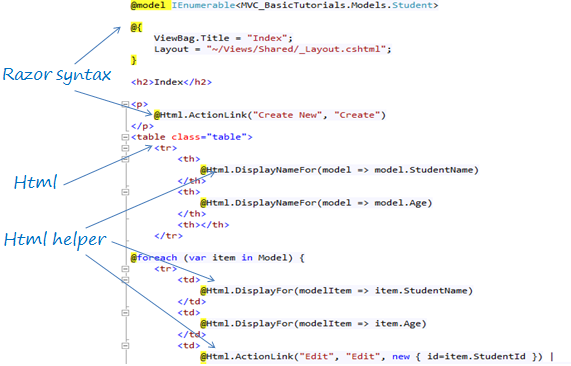
</td>

</tr>

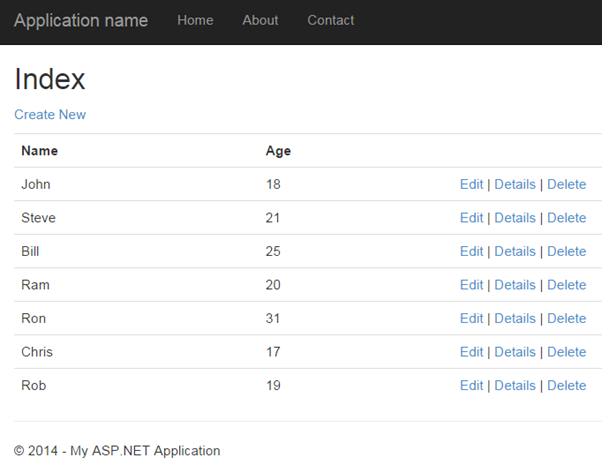
}

</table>

As you can see in the above Index view, it contains both Html and razor codes. Inline razor expression starts with @ symbol. @Html is a helper class to generate html controls. You will learn razor syntax and html helpers in the coming sections.

[](https://www.tutorialsteacher.com/Content/images/mvc/razor-view.png)Index.cshtml

The above Index view would look like below.

[](https://www.tutorialsteacher.com/Content/images/mvc/index-view.png)Index View

 Note:

Every view in the ASP.NET MVC is derived from WebViewPage class included in System.Web.Mvc namespace.

 Points to Remember :

1. View is a User Interface which displays data and handles user interaction.
2. Views folder contains separate folder for each controller.
3. ASP.NET MVC supports Razor view engine in addition to traditional .aspx engine.
4. Razor view files has .cshtml or .vbhtml extension.

**//////////////////////////////////////**

Integrate Controller, View and Model

We have already created StudentController, model and view in the previous sections, but we have not integrated all these components in-order to run it.

The following code snippet shows StudentController and Student model class & view created in the previous sections.

Example: StudentController

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using MVC\_BasicTutorials.Models;

namespace MVC\_BasicTutorials.Controllers

{

public class StudentController : Controller

{

// GET: Student

public ActionResult Index()

{

return View();

}

}

}

Example: Student Model class

namespace MVC\_BasicTutorials.Models

{

public class Student

{

public int StudentId { get; set; }

public string StudentName { get; set; }

public int Age { get; set; }

}

}

Example: Index.cshtml to display student list

@model IEnumerable<MVC\_BasicTutorials.Models.Student>

@{

ViewBag.Title = "Index";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h2>Index</h2>

<p>

@Html.ActionLink("Create New", "Create")

</p>

<table class="table">

<tr>

<th>

@Html.DisplayNameFor(model => model.StudentName)

</th>

<th>

@Html.DisplayNameFor(model => model.Age)

</th>

<th></th>

</tr>

@foreach (var item in Model) {

<tr>

<td>

@Html.DisplayFor(modelItem => item.StudentName)

</td>

<td>

@Html.DisplayFor(modelItem => item.Age)

</td>

<td>

@Html.ActionLink("Edit", "Edit", new { id=item.StudentId }) |

@Html.ActionLink("Details", "Details", new { id=item.StudentId }) |

@Html.ActionLink("Delete", "Delete", new { id = item.StudentId })

</td>

</tr>

}

</table>

Now, to run it successfully, we need to pass a model object from controller to Index view. As you can see in the above Index.cshtml, it uses IEnumerable of Student as a model object. So we need to pass IEnumerable of Student model from the Index action method of StudentController class as shown below.

Example: Passing Model from Controller

public class StudentController : Controller

{

// GET: Student

public ActionResult Index()

{

var studentList = new List<Student>{

new Student() { StudentId = 1, StudentName = "John", Age = 18 } ,

new Student() { StudentId = 2, StudentName = "Steve", Age = 21 } ,

new Student() { StudentId = 3, StudentName = "Bill", Age = 25 } ,

new Student() { StudentId = 4, StudentName = "Ram" , Age = 20 } ,

new Student() { StudentId = 5, StudentName = "Ron" , Age = 31 } ,

new Student() { StudentId = 4, StudentName = "Chris" , Age = 17 } ,

new Student() { StudentId = 4, StudentName = "Rob" , Age = 19 }

};

// Get the students from the database in the real application

return View(studentList);

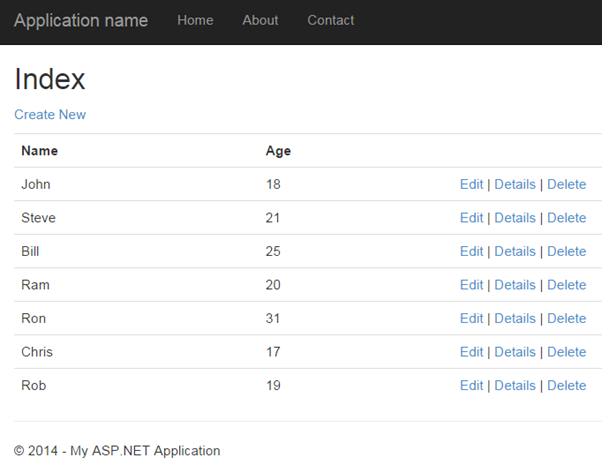
}

}

[Try it](https://www.tutorialsteacher.com/codeeditor?cid=mvc-Dioxf9)

As you can see in the above code, we have created a List of student objects for an example purpose (in real life applicatoin, you can fetch it from the database). We then pass this list object as a parameter in the View() method. The View() method is defined in base Controller class, which automatically binds model object to the view.

Now, you can run the MVC project by pressing F5 and navigate to *http://localhost/Student*. You will see following view in the browser.

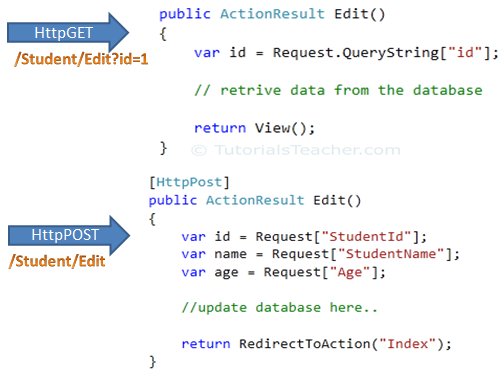
[](https://www.tutorialsteacher.com/Content/images/mvc/index-view.png)

**//////////////////////////////////////**

Model Binding

In this section, you will learn about model binding in MVC framework.

To understand the model binding in MVC, first let's see how you can get the http request values in the action method using traditional ASP.NET style. The following figure shows how you can get the values from HttpGET and HttpPOST request by using the Request object directly in the action method.

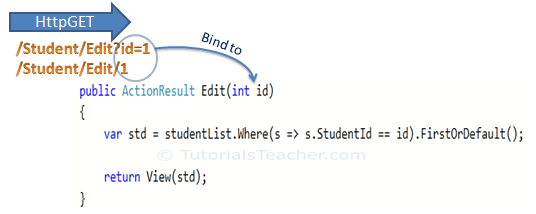
[](https://www.tutorialsteacher.com/Content/images/mvc/request-data.png)Accessing Request Data

As you can see in the above figure, we use the Request.QueryString and Request (Request.Form) object to get the value from HttpGet and HttpPOST request. Accessing request values using the Request object is a cumbersome and time wasting activity.

With model binding, MVC framework converts the http request values (from query string or form collection) to action method parameters. These parameters can be of primitive type or complex type.

Binding to Primitive type

HttpGET request embeds data into a query string. MVC framework automatically converts a query string to the action method parameters. For example, the query string "id" in the following GET request would automatically be mapped to the id parameter of the Edit() action method.

[](https://www.tutorialsteacher.com/Content/images/mvc/model-binding-1.png)Model Binding

This binding is case insensitive. So "id" parameter can be "ID" or "Id".

You can also have multiple parameters in the action method with different data types. Query string values will be converted into paramters based on matching name.

For example,*http://localhost/Student/Edit?id=1&name=John*would map to id and name parameter of the following Edit action method.

Example: Convert QueryString to Action Method Parameters

public ActionResult Edit(int id, string name)

{

// do something here

return View();

}

Binding to Complex type

Model binding also works on complex types. Model binding in MVC framework automatically converts form field data of HttpPOST request to the properties of a complex type parameter of an action method.

Consider the following model classes.

Example: Model classes in C#

public class Student

{

public int StudentId { get; set; }

[Display(Name="Name")]

public string StudentName { get; set; }

public int Age { get; set; }

public Standard standard { get; set; }

}

public class Standard

{

public int StandardId { get; set; }

public string StandardName { get; set; }

}

Now, you can create an action method which includes Student type parameter. In the following example, Edit action method (HttpPost) includes Student type parameter.

Example: Action Method with Class Type Parameter

[HttpPost]

public ActionResult Edit(Student std)

{

var id = std.StudentId;

var name = std.StudentName;

var age = std.Age;

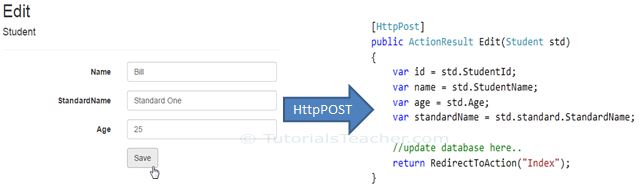
var standardName = std.standard.StandardName;

//update database here..

return RedirectToAction("Index");

}

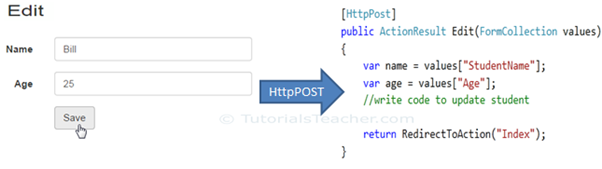
So now, MVC framework will automatically maps Form collection values to Student type parameter when the form submits http POST request to Edit action method as shown below.

[](https://www.tutorialsteacher.com/Content/images/mvc/model-class-binding.png)Model Binding

So thus, it automatically binds form fields to the complex type parameter of action method.

FormCollection

You can also include FormCollection type parameter in the action method instead of complex type, to retrieve all the values from view form fields as shown below.

[](https://www.tutorialsteacher.com/Content/images/mvc/formcollection.png)Model Binding

Bind Attribute

ASP.NET MVC framework also enables you to specify which properties of a model class you want to bind. The [Bind] attribute will let you specify the exact properties a model binder should include or exclude in binding.

In the following example, Edit action method will only bind StudentId and StudentName property of a Student model.

Example: Binding Parameters

[HttpPost]

public ActionResult Edit([Bind(Include = "StudentId, StudentName")] Student std)

{

var name = std.StudentName;

//write code to update student

return RedirectToAction("Index");

}

You can also use Exclude properties as below.

Example: Exclude Properties in Binding

[HttpPost]

public ActionResult Edit([Bind(Exclude = "Age")] Student std)

{

var name = std.StudentName;

//write code to update student

return RedirectToAction("Index");

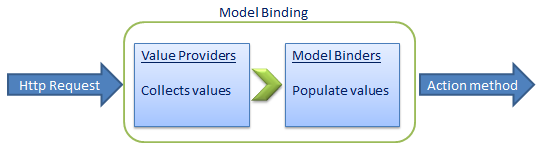
}

The Bind attribute will improve the performance by only bind properties which you needed.

Inside Model Binding

As you have seen that Model binding automatically converts request values into a primitive or complex type object. Model binding is a two step process. First, it collects values from the incoming http request and second, populates primitive type or complex type with these values.

Value providers are responsible for collecting values from request and Model Binders are responsible for populating values.

[](https://www.tutorialsteacher.com/Content/images/mvc/model-binding-3.png)Model Binding in MVC

Default value provider collection evaluates values from the following sources:

1. Previously bound action parameters, when the action is a child action
2. Form fields (Request.Form)
3. The property values in the JSON Request body (Request.InputStream), but only when the request is an AJAX request
4. Route data (RouteData.Values)
5. Querystring parameters (Request.QueryString)
6. Posted files (Request.Files)

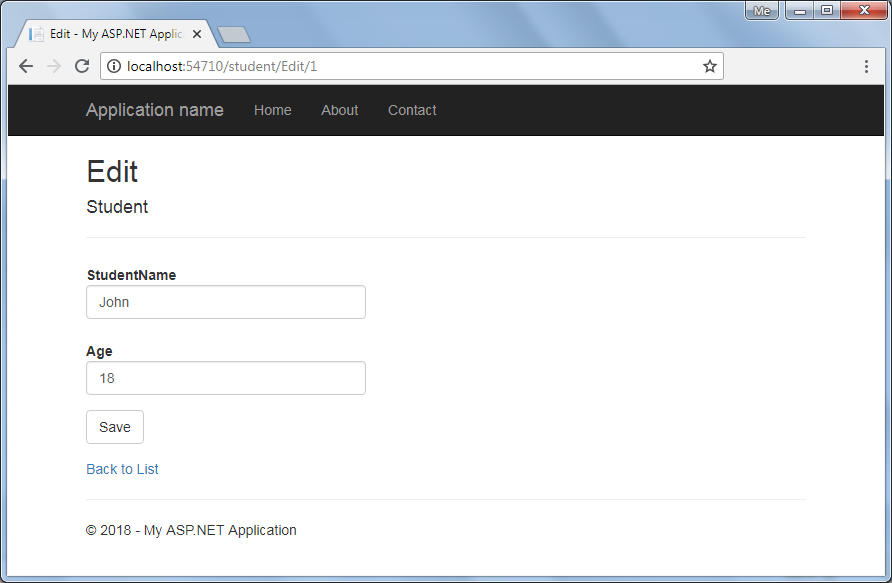
MVC includes [DefaultModelBinder](https://msdn.microsoft.com/en-us/library/system.web.mvc.defaultmodelbinder(v=vs.118).aspx" \t "_blank) class which effectively binds most of the model types.

Visit MSDN for detailed information on [Model binding](https://msdn.microsoft.com/en-us/magazine/hh781022.aspx)

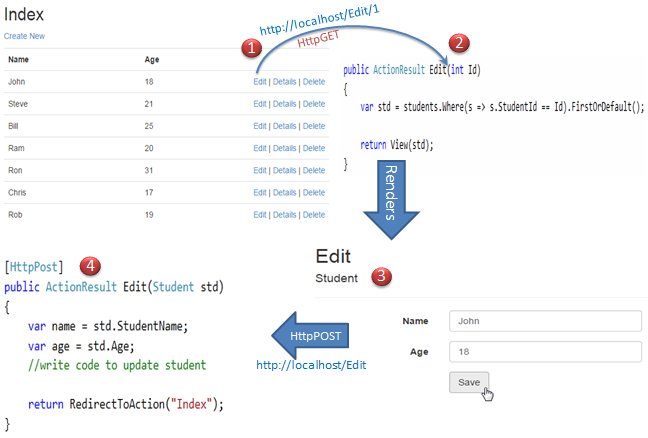
**//////////////////////////////////////**

Create Edit View in ASP.NET MVC

We have already created the Index view in the previous section. In this section, we will create the Edit view using a default scaffolding template as shown below. The user can update existing student data using the Edit view.

[](https://www.tutorialsteacher.com/Content/images/mvc/editview.png)Edit View

The Edit view will be rendered on the click of the Edit button in Index view. The following figure describes the complete set of editing steps.

[](https://www.tutorialsteacher.com/Content/images/mvc/edit-process.png)Editing Steps in MVC

The above figure illustrates the following steps.

1. The user clicks on the Edit link in Index view which will send HttpGET request *http://localhost/student/edit/{Id}* with corresponding Id parameter in the query string. This request will be handled by HttpGET Edit action method.(by default action method handles HttpGET request if no attribute specified)

2. HttpGet Edit action method will fetch student data from the database, based on the supplied Id parameter and render the Edit view with that particular Student data.

3. The user can edit the data and click on the Save button in the Edit view. The Save button will send a HttpPOST request *http://localhost/Student/Edit* with the Form data collection.

4. The HttpPOST Edit action method in StudentController will finally update the data into the database and render an Index page with the refreshed data using the RedirectToAction method as a fourth step.

So this will be the complete process in order to edit the data using Edit view in ASP.NET MVC.

So let's start to implement above steps.

We will be using following Student model class for our Edit view.

Student Model - C#:

public class Student

{

public int StudentId { get; set; }

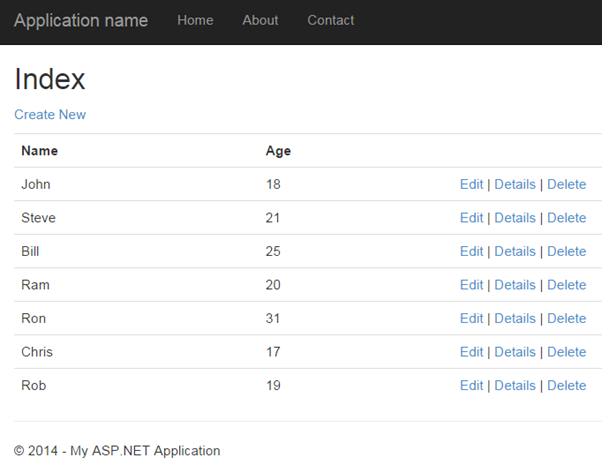
[Display( Name="Name")]

public string StudentName { get; set; }

public int Age { get; set; }

}

Step: 1  
We have already created an Index view in the [previous section](https://www.tutorialsteacher.com/mvc/mvc-view) using a List scaffolding template which includes an Edit action link as shown below.

[](https://www.tutorialsteacher.com/Content/images/mvc/index-view.png)Index View

An Edit link sends HttpGet request to the Edit action method of StudentController with corresponding StudentId in the query string. For example, an Edit link with student John will append a StudentId=1 query string to the request url because John's StudentId is 1. Likewise all the Edit link will include a respective StudentId in the query string.

Step 2:  
Now, create a HttpGET Edit action method in StudentController. The Index view shown above will send the StudentId parameter to the HttpGet Edit action method on the click of the Edit link.

The HttpGet Edit() action method must perform two tasks, first it should fetch the student information from the underlaying data source, whose StudentId matches with the StudentId in the query string. Second, it should render Edit view with the student information so that the user can update it.

So, the Edit() action method should have a StudentId parameter. MVC framework will automatically bind a query string to the parameters of an action method if the name is matches. Please make sure that parameter name matches with the query string.

Example: HttpGet Edit() Action method - C#

using MVC\_BasicTutorials.Models;

namespace MVC\_BasicTutorials.Controllers

{

public class StudentController : Controller

{

IList<Student> studentList = new List<Student>() {

new Student(){ StudentId=1, StudentName="John", Age = 18 },

new Student(){ StudentId=2, StudentName="Steve", Age = 21 },

new Student(){ StudentId=3, StudentName="Bill", Age = 25 },

new Student(){ StudentId=4, StudentName="Ram", Age = 20 },

new Student(){ StudentId=5, StudentName="Ron", Age = 31 },

new Student(){ StudentId=6, StudentName="Chris", Age = 17 },

new Student(){ StudentId=7, StudentName="Rob", Age = 19 }

};

public ActionResult Edit(int Id)

{

//Get the student from studentList sample collection for demo purpose.

//Get the student from the database in the real application

var std = studentList.Where(s => s.StudentId == Id).FirstOrDefault();

return View(std);

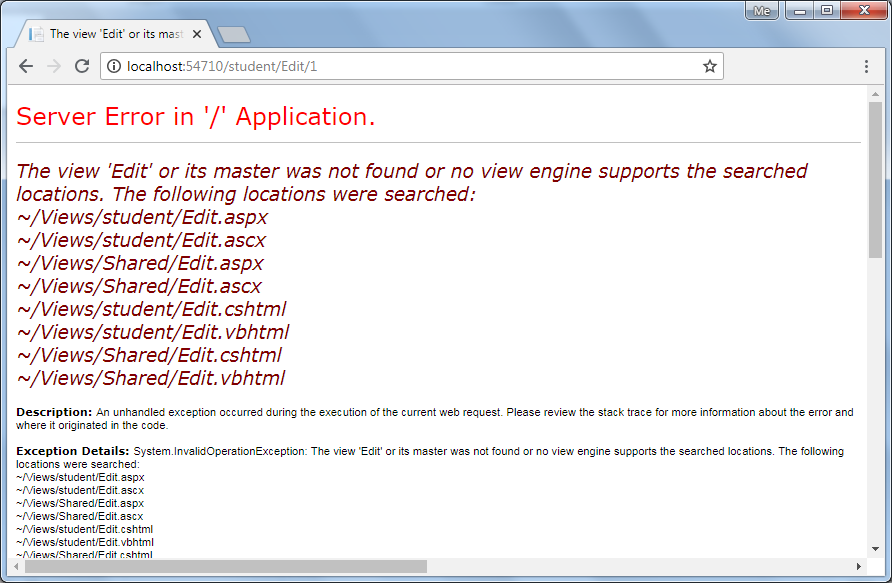
}

}

}

As you can see in the above Edit method, we have used a LINQ query to get the Student from the sample studentList collection whose StudentId matches with supplied StudentId, and then we inject that Student object into View. In a real life application, you can get the student from the database instead of sample collection.

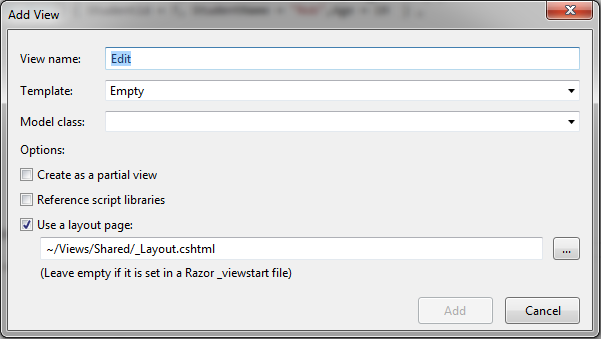
Now, if you click on the Edit link from Index view then you will get following error.

[](https://www.tutorialsteacher.com/Content/images/mvc/editview-error.png)Edit View Error

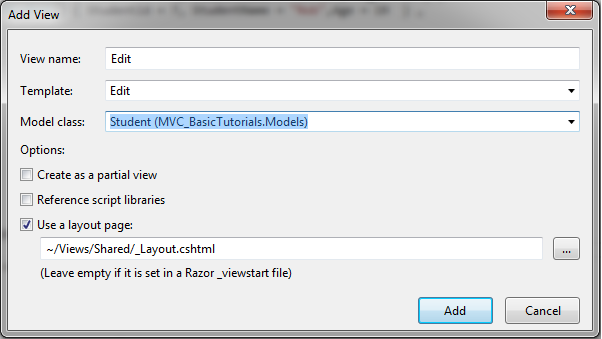
The above error occurred because we have not created an Edit view yet. By default, MVC framework will look for Edit.cshtml or Edit.vbhtml or Edit.aspx or Edit.ascx file in View -> Student or Shared folder.

Step 3:  
To create Edit view, right click inside Edit action method and click on **Add View..** It will open Add View dialogue.

In the Add View dialogue, keep the view name as Edit. (You can change as per your requirement.)

[](https://www.tutorialsteacher.com/Content/images/mvc/edit-view-2.png)Edit View

Select Edit in the Template dropdown and also select Student for Model class as shown below.

[](https://www.tutorialsteacher.com/Content/images/mvc/edit-view-3.png)Edit View

Now, click Add to generate Edit.cshtml view under View/Student folder as shown below.

Edit.cshtml:

@model MVC\_BasicTutorials.Models.Student

@{

ViewBag.Title = "Edit";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h2>Edit</h2>

@using (Html.BeginForm())

{

@Html.AntiForgeryToken()

<div class="form-horizontal">

<h4>Student</h4>

<hr />

@Html.ValidationSummary(true, "", new { @class = "text-danger" })

@Html.HiddenFor(model => model.StudentId)

<div class="form-group">

@Html.LabelFor(model => model.StudentName, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.StudentName, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.StudentName, "", new { @class = "text-danger" })

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.Age, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.Age, new { htmlAttributes = new { @class = "form-control" } })

@Html.ValidationMessageFor(model => model.Age, "", new { @class = "text-danger"< })

</div>

</div>

<div class="form-group">

<div class="col-md-offset-2 col-md-10">

<input type="submit" value="Save" class="btn btn-default" />

</div>

</div>

</div>

}

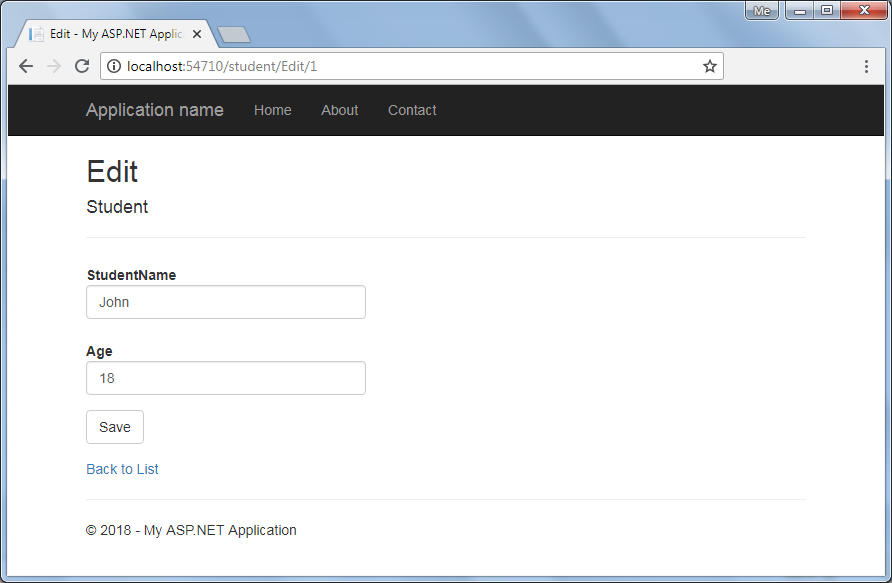
<div>

@Html.ActionLink("Back to List", "Index")

</div>

Please notice that Edit.cshtml includes HtmlHelper method @using (Html.BeginForm()) to create a html form element. Html.BeginForm sends a HttpPost request by default.

Now, click on the Edit link of any student in the Index view. Edit view will be display student information whose Edit link clicked, as shown below.

[](https://www.tutorialsteacher.com/Content/images/mvc/editview.png)Edit View

You can edit the Name or Age of Student and click on Save. Save method should send a HttpPOST request because the POST request sends form data as a part of the request, not in the querystring. So write a POST method as fourth step.

Step 4:  
Now, write POST Edit action method to save the edited student as shown below.

Example: POST Method in MVC

[HttpPost]

public ActionResult Edit(Student std)

{

//write code to update student

return RedirectToAction("Index");

}

As you can see in the above code, the Edit() method requires a Student object as an input parameter. The Edit() view will automatically binds form's data collection to the student model parameter. Please visit [Model Binding](https://www.tutorialsteacher.com/mvc/model-binding-in-asp.net-mvc) section for more information. Here, you can update the information to the database and redirect it to Index action. (we have not written code to update database here for demo purpose)

Now, clicking on the Save button in the Edit view will save the updated information and redirect it to the Index() action method.

In this way, you can provide edit functionality using a default scaffolding Edit template. However, you can also create an Edit view without using an Edit scaffolding template.

The following example demonstrates the StudentController class with all the action methods.

Example: Controller in C#

using MVC\_BasicTutorials.Models;

namespace MVC\_BasicTutorials.Controllers

{

public class StudentController : Controller

{

IList<Student> studentList = new List<Student>() {

new Student(){ StudentId=1, StudentName="John", Age = 18 },

new Student(){ StudentId=2, StudentName="Steve", Age = 21 },

new Student(){ StudentId=3, StudentName="Bill", Age = 25 },

new Student(){ StudentId=4, StudentName="Ram", Age = 20 },

new Student(){ StudentId=5, StudentName="Ron", Age = 31 },

new Student(){ StudentId=6, StudentName="Chris", Age = 17 },

new Student(){ StudentId=7, StudentName="Rob", Age = 19 }

};

// GET: Student

public ActionResult Index()

{

return View(studentList);

}

public ActionResult Edit(int Id)

{

//Get the student from studentList sample collection for demo purpose.

//Get the student from the database in the real application

var std = studentList.Where(s => s.StudentId == Id).FirstOrDefault();

return View(std);

}

[HttpPost]

public ActionResult Edit(Student std)

{

//write code to update student

return RedirectToAction("Index");

}

}

}

**//////////////////////////////////////**

Razor Syntax

Razor is one of the view engine supported in ASP.NET MVC. Razor allows you to write mix of HTML and server side code using C# or Visual Basic. Razor view with visual basic syntax has .vbhtml file extension and C# syntax has .cshtml file extension.

Razor syntax has following Characteristics:

* **Compact**: Razor syntax is compact which enables you to minimize number of characters and keystrokes required to write a code.
* **Easy to Learn**: Razor syntax is easy to learn where you can use your familiar language C# or Visual Basic.
* **Intellisense**: Razor syntax supports statement completion within Visual Studio.

Now, let's learn how to write razor code.

Inline expression

Start with @ symbol to write server side C# or VB code with Html code. For example, write @Variable\_Name to display a value of a server side variable. For example, DateTime.Now returns a current date and time. So, write @DateTime.Now to display current datetime as shown below. A single line expression does not require a semicolon at the end of the expression.

C# Razor Syntax

<h1>Razor syntax demo</h1>

<h2>@DateTime.Now.ToShortDateString()</h2>

Output:

**Razor syntax demo**

08-09-2014

Multi-statement Code block

You can write multiple line of server side code enclosed in braces @{ ... }. Each line must ends with semicolon same as C#.

Example: Server side Code in Razor Syntax

@{

var date = DateTime.Now.ToShortDateString();

var message = "Hello World";

}

<h2>Today's date is: @date </h2>

<h3>@message</h3>

Output:

Today's date is: 08-09-2014

Hello World!

Display Text from Code Block

Use @: or <text>/<text> to display texts within code block.

Example: Display Text in Razor Syntax

@{

var date = DateTime.Now.ToShortDateString();

string message = "Hello World!";

@:Today's date is: @date <br />

@message

}

Output:

Today's date is: 08-09-2014

Hello World!

Display text using <text> within a code block as shown below.

Example: Text in Razor Syntax

@{

var date = DateTime.Now.ToShortDateString();

string message = "Hello World!";

<text>Today's date is:</text> @date <br />

@message

}

Output:

Today's date is: 08-09-2014

Hello World!

if-else condition

Write if-else condition starting with @ symbol. The if-else code block must be enclosed in braces { }, even for single statement.

Example: if else in Razor

@if(DateTime.IsLeapYear(DateTime.Now.Year) )

{

@DateTime.Now.Year @:is a leap year.

}

else {

@DateTime.Now.Year @:is not a leap year.

}

Output:

2014 is not a leap year.

for loop

Example: for loop in Razor

@for (int i = 0; i < 5; i++) {

@i.ToString() <br />

}

Output:

0

1

2

3

4

Model

Use @model to use model object anywhere in the view.

Example: Use Model in Razor

@model Student

<h2>Student Detail:</h2>

<ul>

<li>Student Id: @Model.StudentId</li>

<li>Student Name: @Model.StudentName</li>

<li>Age: @Model.Age</li>

</ul>

Output:

**Student Detail:**

- Student Id: 1

- Student Name: John

- Age: 18

Declare Variables

Declare a variable in a code block enclosed in brackets and then use those variables inside html with @ symbol.

Example: Variable in Razor

@{

string str = "";

if(1 > 0)

{

str = "Hello World!";

}

}

<p>@str</p>

Output:

Hello World!

So this was some of the important razor syntaxes. Visit asp.net to learn [razor syntax](https://www.asp.net/web-pages/overview/getting-started/introducing-razor-syntax-(c)" \o "Learn Razor Syntax" \t "_blank) in detail.

 Points to Remember :

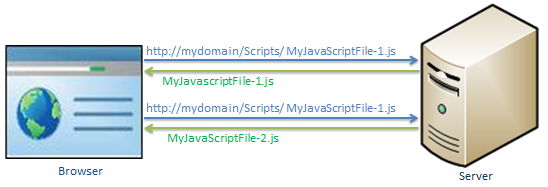
1. Use @ to write server side code.
2. Server side code block starts with @{\* code \* }
3. Use @: or <text></<text> to display text from code block.
4. if condition starts with @if{ }
5. for loop starts with @for
6. @model allows you to use model object anywhere in the view.

**//////////////////////////////////////**

Bundling

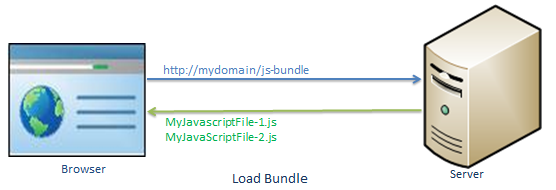
Bundling and minification techniques were introduced in MVC 4 to improve request load time. Bundling allow us to load the bunch of static files from the server into one http request.

The following figure illustrates the bundling techniques:

[](https://www.tutorialsteacher.com/Content/images/mvc/bundling-1.png)Load script files in separate requests

In the above figure, browser sends two separate requests to load two different JavaScript file MyJavaScriptFile-1.js and MyJavaScriptFile-2.js.

Bundling technique in MVC 4 allows us to load more than one JavaScript file, MyJavaScriptFile-1.js and MyJavaScriptFile-2.js in one request as shown below.

[](https://www.tutorialsteacher.com/Content/images/mvc/bundling-2.png)

Minification

Minification technique optimizes script or css file size by removing unnecessary white space and comments and shortening variable names to one character.

For example, consider following JavaScript function.

Example: JavaScript

sayHello = function(name){

//this is comment

var msg = "Hello" + name;

alert(msg);

}

The above JavaScript will be optimized and minimized into following script.

Example: Minified JavaScript

sayHello=function(n){var t="Hello"+n;alert(t)}

As you can see above, it has removed unnecessary white space, comments and also shortening variable names to reduce the characters which in turn will reduce the size of JavaScript file.

Bundling and minification impacts on the loading of the page, it loads page faster by minimizing size of the file and number of requests.

Bundle Types

MVC 5 includes following bundle classes in System.web.Optimization namespace:

**ScriptBundle**: ScriptBundle is responsible for JavaScript minification of single or multiple script files.

**StyleBundle**: StyleBundle is responsible for CSS minification of single or multiple style sheet files.

**DynamicFolderBundle**: Represents a Bundle object that ASP.NET creates from a folder that contains files of the same type.

All the above bundle classes are included in *System.Web.Optimization.Bundle* namespace and derived from [Bundle class](https://msdn.microsoft.com/en-us/library/system.web.optimization.bundle(v=vs.110).aspx).

Learn about ScriptBundle in the next section.

 Points to Remember :

1. Bundling and Minification minimize static script or css files loading time therby minimize page loading time.
2. MVC framework provides ScriptBundle, StyleBundle and DynamicFolderBundle classes.
3. ScriptBundle does minification of JavaScript files.
4. StyleBundle does minification of CSS files.

**//////////////////////////////////////**

ScriptBundle in ASP.NET MVC

We have learned how bundling technique works in ASP.NET MVC. Here, we will learn how to create a bundle of multiple JavaScript files in one http request.

ASP.NET MVC API includes [ScriptBundle](https://msdn.microsoft.com/en-us/library/system.web.optimization.scriptbundle(v=vs.110).aspx" \t "_blank) class that does JavaScript minification and bundling.

Open App\_Start\BundleConfig.cs file in the MVC folders. The BundleConfig.cs file is created by MVC framework by default. You should write your all bundling code in the BundleConfig.RegisterBundles() method. (you can create your own custom class instead of using BundleConfig class, but it is recommended to follow standard practice.) The following code shows a portion of the RegisterBundles method.

Example: BundleConfig.RegisterBundle()

using System.Web;

using System.Web.Optimization;

public class BundleConfig

{

public static void RegisterBundles(BundleCollection bundles)

{

// create an object of ScriptBundle and

// specify bundle name (as virtual path) as constructor parameter

ScriptBundle scriptBndl = new ScriptBundle("~/bundles/bootstrap");

//use Include() method to add all the script files with their paths

scriptBndl.Include(

"~/Scripts/bootstrap.js",

"~/Scripts/respond.js"

);

//Add the bundle into BundleCollection

bundles.Add(scriptBndl);

BundleTable.EnableOptimizations = true;

}

}

In the above example, we have created a bundle of two JavaScript files, bootstrap.js and respond.js using ScriptBundle for demo purposes.

1. First of all create an instance of ScriptBundle class by specifing the bundle name as a constructor parameter. This bundle name is a virtual path starting with ~/. You can give anything in virtual path but it's recommended to give a path that will be easy to identify as a bundle. Here, we have given "~/bundles/bootstrap" path, so that we can easily identify that this bundle includes bootstrap related files.
2. Use Include method to add one or more JS files into a bundle with its relative path after root path using ~ sign.
3. Final, add the bundle into BundleCollection instance, which is specified as a parameter in RegisterBundle() method.
4. Last, BundleTable.EnableOptimizations = true enables bundling and minification in debug mode. If you set it to false then it will not do bundling and minification.

You can also use IncludeDirectory method of bundle class to add all the files under particular directory as shown below.

ScriptBundle Example:

public static void RegisterBundles(BundleCollection bundles)

{

bundles.Add(new ScriptBundle("~/bundles/scripts").IncludeDirectory("~/Scripts/","\*.js",true));

}

Thus, you can create a bundle of JavaScript files using ScriptBundle. MVC framework invokes BundleConfig.RegisterBundle() method from the Application\_Start event in Global.asax.cs file, so that it can add all the bundles into BundleCollection at the starting of an application.

Example: Invoke RegisterBundle() in Application\_Start event

public class MvcApplication : System.Web.HttpApplication

{

protected void Application\_Start()

{

BundleConfig.RegisterBundles(BundleTable.Bundles);

}

}

Using Wildcards

Sometime third party script files includes versions in a name of script file. So it is not advisable to changes the code whenever you upgrade the version of script file. With the use of wildcards, you don't have to specify a version of a script file. It automatically includes files with the version available.

For example, Jquery files includes the version in a name. So you can use {version} wildcard to pickup a version based on available version.

Example: Wildcard with bundle

public class BundleConfig

{

public static void RegisterBundles(BundleCollection bundles)

{

bundles.Add(new ScriptBundle("~/bundles/jquery")

.Include( "~/Scripts/jquery-{version}.js"));

}

}

Now, it will pick up jquery file added in a project. If you have included jquery-1.7.1.js then it will render this file and when you upgrade jquery file to jquery-1.10.2.js then it will automatically render 1.10 version file without changing or compiling code.

Using CDN

You can also use Content Delivery Network to load script files. For example, you can load jquery library from CDN as shown below.

Example: Load files from CDN

public class BundleConfig

{

public static void RegisterBundles(BundleCollection bundles)

{

var cdnPath = "http://ajax.aspnetcdn.com/ajax/jQuery/jquery-1.7.1.min.js";

bundles.Add(new ScriptBundle("~/bundles/jquery", cdnPath)

.Include( "~/Scripts/jquery-{version}.js"));

}

}

In the above code, jquery will be requested from the CDN while in release mode and in the debug mode, jquery library will be loaded from a local source. Please note that you should have a fallback mechanism to deal with a CDN request failure.

Now, let's see how to use the bundle into a razor view.

Include ScriptBundle in Razor View

We have create a script bundle above. Now, we will learn how to include bundle into razor view.

The script bundles can be included using static [Scripts](https://msdn.microsoft.com/en-us/library/system.web.optimization.scripts(v=vs.110).aspx" \t "_blank) class. Use Scripts.Render() method to include specified script bundle at runtime.

Example: Scripts.Render()

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>@ViewBag.Title</title>

**@Scripts.Render("~/bundles/bootstrap")**

</head>

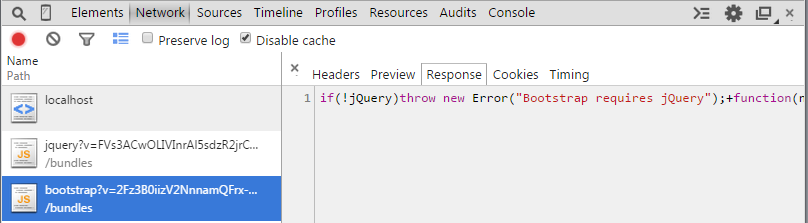
<body>

@\*html code removed for clarity \*@

</body>

</html>

Now, if you run the above example then you will find two script files is combined, minified and loaded in a single request. Please make sure that you have set debug = false in web.config <compilation debug="false" targetFramework="4.5"/>

[](https://www.tutorialsteacher.com/Content/images/mvc/bundling-3.png)Load Bundle in Browser

As you can see in the above figure that bootstrap bundle is loaded in a single request. It has also combined and minified two JS files for bootstrap.

 Points to Remember :

1. Bundling and Minification minimize static script or css files loading time therby minimize page loading time.
2. ScriptBundle does minification of JavaScript files.
3. Create script or css bundles in BundleConfig class included in App\_Start folder.
4. Use wildcard {version} to render available version files at runtime.
5. Use Scripts.Render("bundle name") method to include script bundle in a razor view.

**//////////////////////////////////////**

StyleBundle

You have learned how to create a bundle of JavaScript files in the previous section. Here, you will learn how to create a bundle of style sheet files (CSS).

ASP.NET MVC API includes [StyleBundle](https://msdn.microsoft.com/en-us/library/system.web.optimization.stylebundle(v=vs.110).aspx" \t "_blank) class that does CSS minification and bundling. StyleBundle is also derived from a Bundle class so it supports same methods as ScriptBundle.

As mentioned in the previous section, you should create bundles of script and css files in the RegisterBundles() method of BundleConfig class contained in App\_Start -> BundleConfig.cs file.

The following code shows a portion of the RegisterBundles() method.

Use ScriptsInclude or IncludeDerictory method to add css files into bundle as shown below:

Example: StyleBundle

public class BundleConfig

{

public static void RegisterBundles(BundleCollection bundles)

{

bundles.Add(new StyleBundle("~/bundles/css").Include(

"~/Content/bootstrap.css",

"~/Content/site.css"

));

// add ScriptBundle here..

}

}

As you can see in the above example, we have created StyleBundle instance with bundle name as virtual path. The bundle name (virtual path) must start with ~/. Use Include() or IncludeDirectory() method with css file names as a string.

You can use wildcard and CDN path the same way as ScriptBundle as shown in the previous section.

Include Style Bundle in Razor View

You can use StyleBundle in a layout view and render bunch of css files in a single request using static [Styles](https://msdn.microsoft.com/en-us/library/system.web.optimization.styles(v=vs.110).aspx" \t "_blank) class. Styles is a helper class to render css bundles.

Example: Include Style Bundle in View

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>@ViewBag.Title - My ASP.NET Application</title>

**@Styles.Render("~/bundles/css")**

</head>

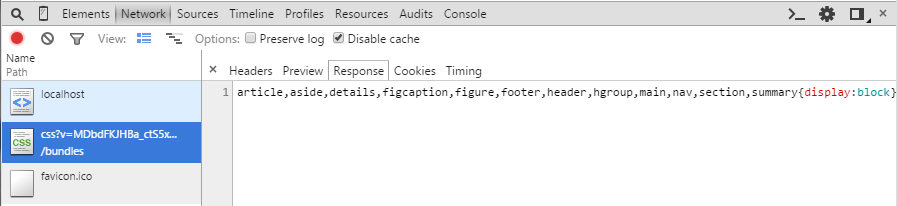
<body>

@\*html code removed for clarity \*@

</body>

</html>

As shown in the above example, use Styles.Render() method to include specified css bundle at runtime. Open developer tool of the browser and check that it has minified and loaded css files as shown below.

[](https://www.tutorialsteacher.com/Content/images/mvc/stylebundle-demo.png)Load Bundle in Browser

Learn [how to set image path in StyleBundle](https://www.tutorialsteacher.com/articles/how-to-set-image-path-in-style-bundle).

 Points to Remember :

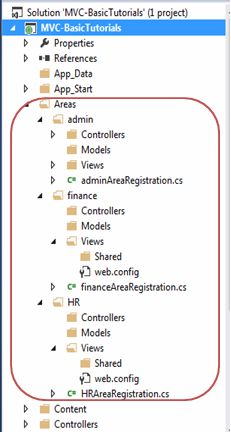
1. Bundling and Minification minimize static script or css files loading time therby minimize page loading time.
2. MVC framework provides ScriptBundle, StyleBundle and DynamicFolderBundle classes.
3. StyleBundle does minification of CSS files.
4. Create script or css bundles in the BundleConfig class included in App\_Start folder.
5. Use wildcard {version} to render available version files at runtime.
6. Use Styles.Render("bundle name") method to include style bundles in a razor view.

**//////////////////////////////////////**

Area

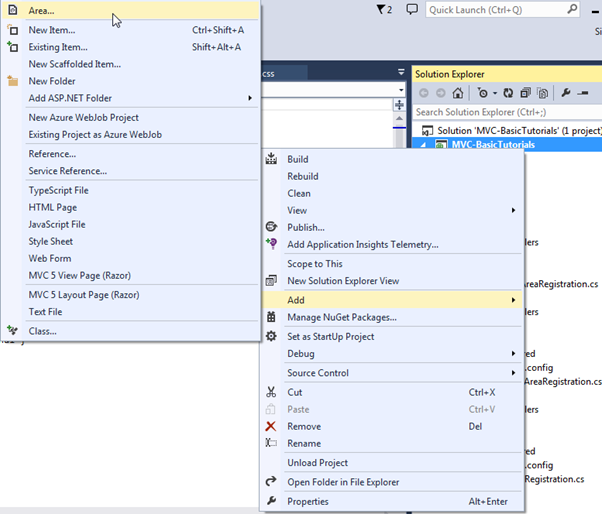
You have already learned that ASP.NET MVC framework includes separate folders for Model, View and Controller. However, large application can include a large number of controller, views and model classes. So to maintain a large number of views, models and controllers with the default ASP.NET MVC project structure can become unmanageable

ASP.NET MVC 2 introduced Area. Area allows us to partition large application into smaller units where each unit contains separate MVC folder structure, same as default MVC folder structure. For example, large enterprise application may have different modules like admin, finance, HR, marketing etc. So an Area can contain separate MVC folder structure for all these modules as shown below.

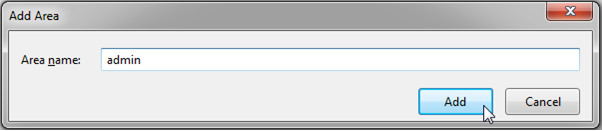
[](https://www.tutorialsteacher.com/Content/images/mvc/area-1.png)Area

Create Area

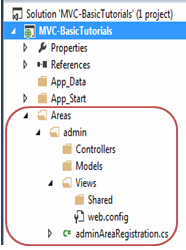
You can create an Area using ASP.NET MVC 5 and Visual Studio 2013 for web by right clicking on the project in the solution explorer -> Add -> Area..

[](https://www.tutorialsteacher.com/Content/images/mvc/create-area.png)Area

Enter Area name in Add Area dialogue box and click Add.

[](https://www.tutorialsteacher.com/Content/images/mvc/area-2.png)Area

This will add 'admin' folder under Area folder as shown below.

[](https://www.tutorialsteacher.com/Content/images/mvc/area-3.png)Area

As you can see, each area includes AreaRegistration class in {area name} + AreaRegistration.cs file.

The following is adminAreaRegistration class created with admin area.

Area Registration:

public class adminAreaRegistration : AreaRegistration

{

public override string AreaName

{

get

{

return "admin";

}

}

public override void RegisterArea(AreaRegistrationContext context)

{

context.MapRoute(

"admin\_default",

"admin/{controller}/{action}/{id}",

new { action = "Index", id = UrlParameter.Optional }

);

}

}

AreaRegistration class overrides RegisterArea method to map the routes for the area. In the above example, any URL that starts with **admin** will be handled by the controllers included in the admin folder structure under Area folder. For example, *http://localhost/admin/profile* will be handled by profile controller included in Areas/admin/controller/ProfileController folder.

Finally, all the area must be registered in Application\_Start event in Global.asax.cs as AreaRegistration.RegisterAllAreas();

So in this way, you can create and maintain multiple areas for the large application.

**//////////////////////////////////////**

**//////////////////////////////////////**

**//////////////////////////////////////**

# Asp.Net Mvc | Html Helpers

Link : <https://medium.com/@ulkutokmak/asp-net-mvc-html-helpers-484ae121e383>



Daha önceden kullananlar için WebForms da yer alan tools kısmı, designer ekranı malesef Asp.Net Mvc de bulunmamaktadır. Peki Asp.Net Mvc kontrol eklemeyi bize nasıl sağlıyor. İşte burada devreye HtmlHelpers’ lar giriyor.

Aslında yazdığımız, tanımladığımız kontrollerin hepsi html. Web Forms’da da öyleydi. Sürükleyip bıraktığımızda arka planda bir html kodu üretilerek kontroller sayfamıza ekleniyordu. İşte Html Helpersla da aynı mantık üzerinden gidiyoruz.

Html Helpers bize oluşturmak istediğimiz kontrolü tekrar tekrar tanımlamak yerine tek bir yerden defalarca tanımlamadan ulaşmamızı sağlar. Ayrıca Html Helper’in en can alıcı noktası oluşturacağımız kontrollerde değişiklik yapabilme imkanı sağlaması.

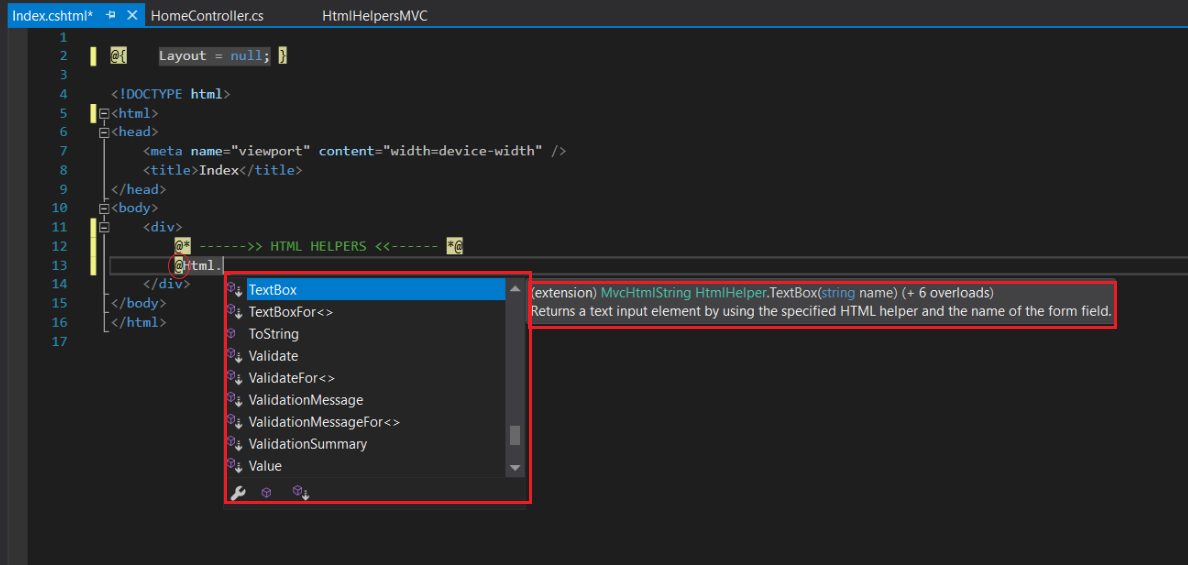
Misal bir textbox oluşturmak istiyorsunuz ve yazı rengi hep kırmızı olsun istiyorsunuz. Textbox kullanacağınızı her yerde style=” color:red” olarak tanımlamak zamanınızdan çalabilir. Bunu yerine Html Helper kullanarak textbox’ınızı özelleştirerek sadece renk değerini verip tanımlayabilirsiniz.

Asp: <asp:textbox id="txtAsp" />  
Html: <input type="text" id="txtHtml" value="Text" />  
Html Helpers: @Html.TextBox("txtHelpers");

Yukarıda yazdığım tanımlamalarda 3 farklı yazım şekli görüyoruz. Web Froms da sürükle bırak ile sayfamıza koyduğumuz kontroller asp ye çevriliyor. Asp ‘de arka planda html’ e çeviriyordu. Html Helpers’la da tanımlamalarımız bu döngüyü takıp eder. Çünkü her zaman kullanıcıya Html geri döner.

İşin tanımsal kısmını geçtiğimize göre bide bunları uygulama üzerinde görelim.

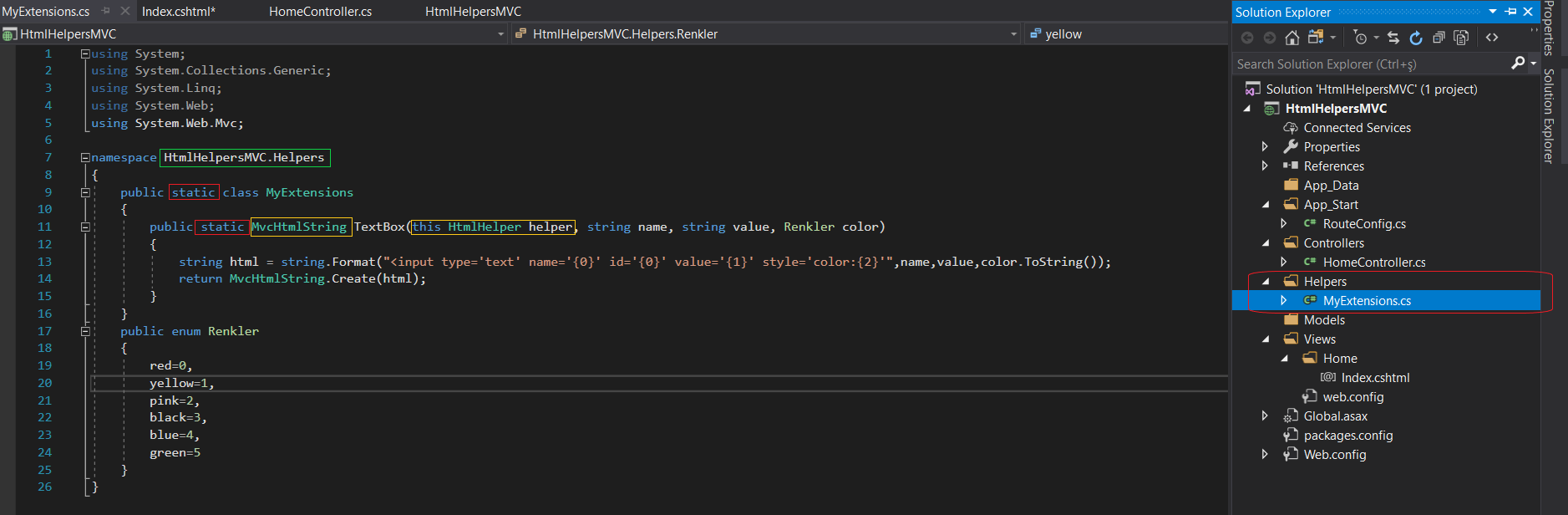
Oluşturduğum projenin adını HtmlHelpersMvc yaptım. Her zaman ki adımları takip ederek HomeController ve Index sayfamı oluşturdum.



Aslında kullanması oldukça basit. Html Helpers da @ işareti ile tanımlanır. Daha sonra eklemek istediğiniz kontrolü açılan listeden seçebilirsiniz. Tavsiyem bu listedekilere bakmanız hatta tek tek denemeniz. Html de link için kullandığımız <a href=””></a> kontrolünü HtmlHelpers’da @Html.ActionLink() ile kullanabiliriz.

Seçili olan kontrolün yanında hemen açıklaması da geliyor buda aslında işimizi çok kolaylaştırıyor. Benden ne istiyor biliyorum ve yazıyorum. Açıklama da fark ettiyseniz +6 overloads yazıyor. Bu demek oluyor ki ben TextBox’ımı 6 farklı şekilde tanımlayabilirim. Birinde sadece name aldırabilirken diyerinde html attributes de ekleyebilirsiniz ya da 7. overload’ı siz yazarsınız ve yukarıda bahsettiğim gibi renk değerini aldırır ve her istediğiniz yerde kullanabilirsiniz. Kendi HtmlHelpers methodumuzu yazdığımızda Costum Html Helper Metod olarak adlandırılıyor. Bizde hemen kendi Custom Html Helper metodumuzu yapalım.

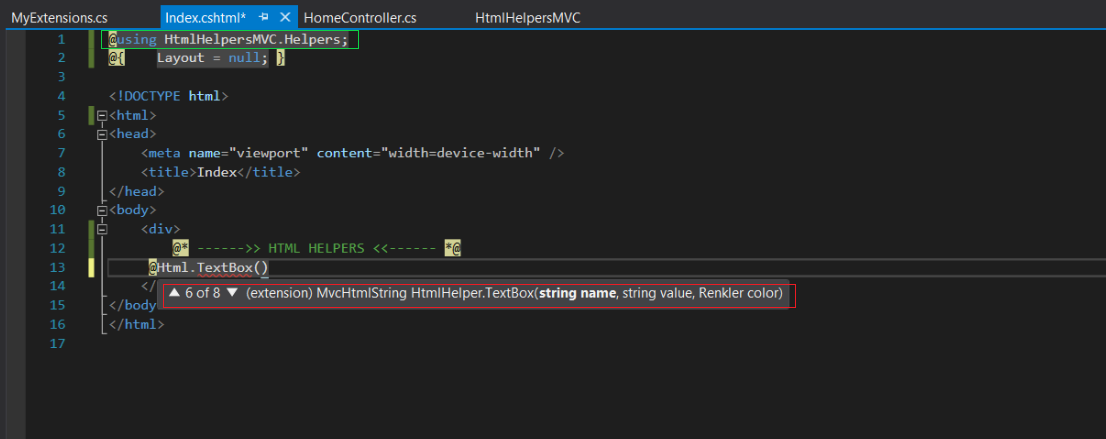
Öncelikle projemde Helpers adında bir klasör oluşturuyorum. Daha sonra bu klasöre Add-Class diyerek MyExtensions(genelde bu isim kullanılır) adında bir class ekliyorum. Bu classın ve içindeki property ve methodların static olması gerekmektedir.



Yukarıda işaretlediğim gibi sınıfımız ve metodları static olmalıdır. Şimdi sırayla işaretli alanları açıklayayım.

**MvcHtmlString** → HtmlHelpers metodları geriye MvcStringHtml döndürüyor. Bizde bu yüzden geri dönüş tipimizi MvcStringHtml olarak tanımlıyoruz. İlk yazdığınızda tanımayacaktır. İsterseniz üzerine gelerek show potential fixes diyerek ya da doğru yazdığınıza eminseniz Ctrl+. ile gerekli kütüphaneyi(System.Web.Mvc) sayfanıza ekleyin.  
**this HtmlHelper helper** → Tanımladığımız metodun HtmlHelper TexBox da çıkması için bunu yazıyoruz.  
**Renkler**→ Kullanılacak olan color’ı kullanıcıdan string olarak aldırsaydık misal purple yerine pruple yazabilirdi. Biz bunun önüne geçmek için enum tanımlayarak kullanıcıya bunu seçtiriyoruz ve yanlış girilmesini ortadan kaldırmış oluyoruz.

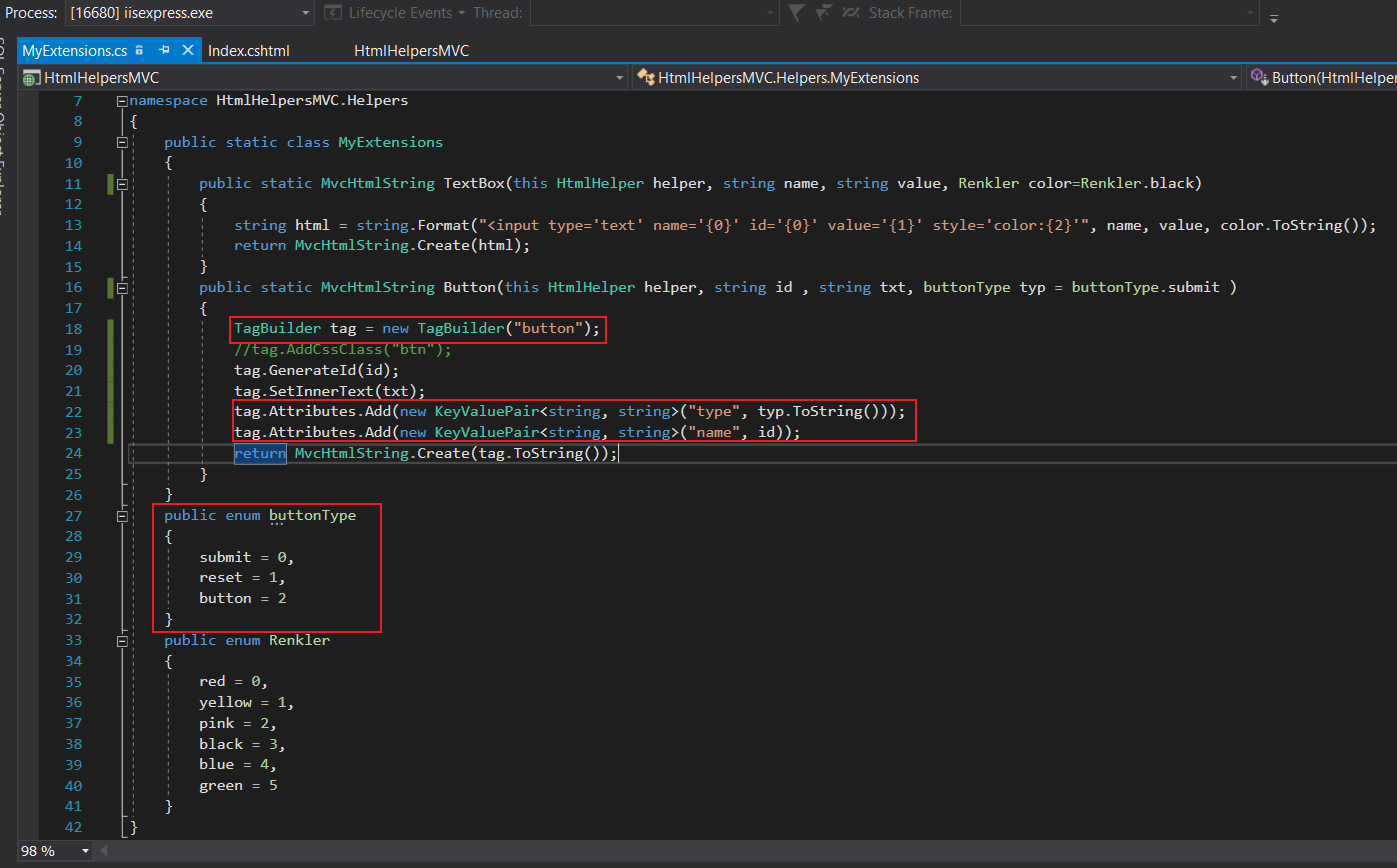
Şimdi yazdığımız helper metodunu sayfamızda kullanalım.



Tanımladığımız metoda sayfamızda ulaşmak için en önemli nokta classımızın yolunu using’e eklemektir. Using’ e eklediğiniz halde çıkmıyorsa sayfanızı kapatım projenizi Rebuild ederseniz gelecektir.

Resimde de görüldüğü gibi tanımladığımız metodumuz 6.overload olarak çıktı. Artık kendi yazdığımız textboxımızı her yerde kullanabiliriz. :)

Html Helper oluşturmanın 2. yolu olan TagBuilder’ı da anlatmak isterim. Yararlı olacağını düşünüyorum.

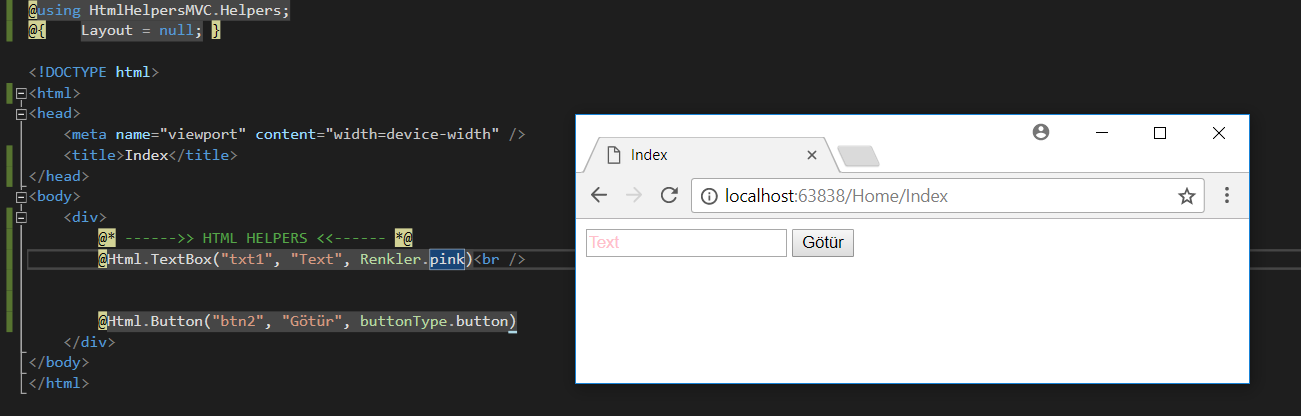


Normalde HtmlHelpers da tanımlı button kontrolü yok ben hem tag builder’ı anlattım hemde buttonumuzu nasıl oluştururuz onu göstermiş oldum.

Yine bir **enum**tanımladım ki button type yanlış girilmesin.  
**Tag Builder** da tanımladığınız özelliklere göre istediğimiz kontrolü bizim için oluşturur. AddCssClass ile bootstrap kullanıyorsanız onun csslerini kullnamıyorsanız kendi yazdığınız cssleri yazarak butonumuzu istediğimiz şekle sokuyoruz. Ben şimdilik css tanımlaması yapmadım.  
**GenerateId** ile kontrolün idsini veriyoruz.  
**SetInnerText**ile kontrolün textini veriyoruz.  
**Attributes** ile kontrole istediğimiz diğer özelliklerini ekliyoruz. Bunu da KeyValuePair ile yapıyoruz.

KeyValuePair nasıl çalışır? ilk örneğimde gelen değeri karşılık geldiği alana name={0} olarak tanımlamıştım. KeyValuePair de aynı mantıkla çalışır. İlk önce key’i sonra gelen value’yu veririz.

Şimdi sayfamda nasıl çağırmışım bir de ona bakalım.



HtmlHelper ile oluşturduğum buton ve Textbox’ın çıktısı bu şekildedir.

## **What are HTML helpers?**

An HTML helper allows you to create arbitrary HTML code. The Razor view engine already defines a lot of helpers for you. Examples are:

* @Html.TextBoxFor(model=>model.name)
* @Html.ActionLink()

In short, HTML helpers help to keep pages clean and readable by reducing an HTML block of multiple elements into a single line.

## **Creating a custom HTML helper**

When you have a specific use case that isn’t covered by the default HTML helpers, you can create your own helper. Creating your own helper has the advantage that the page is cleaner and more readable. An additional benefit is that we can now write a unit test for this particular HTML helper.

For example, imagine a website that allows a user to upload an image of their bike trip. If the user clicks on the image, a file selector popup shows up. This popup allows the user to upload a new image.

The HTML block without the HTML helper looks like this:

@model BikeWeb.ViewModels.BikeViewModel

<div class="pull-left upload-img-wrapper">

<label class="upload-img" data-content="Change Image">

<img class="img-responsive" height="250" src="@Model.ImageSource" width="250"></img>

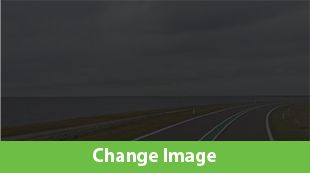
</label>

<input id="ImageName" name="ImageName" style="display:none;" type="file" value="58.jpg" />

</div>

<rest of the page excluded for brevity>

When you hover over the image, it fades out and the green banner at the bottom pops up to hint to the user that he can upload a new image.

To simplify the HTML block on both the edit and create-page, we need to create an ImageUpload HTML helper that can be called from our view that creates the HTML block. The image element needs the URL of the image and the label needs a translated text. We get both from the BikeViewModel.

## **Creating the ImageUpload HTML helper**

using BikeWeb.ViewModels;

using System.Text;

using System.Web;

using System.Web.Mvc;

using System.Web.Mvc.Html;

namespace BikeWeb.Extensions

{

public static class HtmlHelperExtensions

{

public static IHtmlString ImageUpload(this HtmlHelper<BikeViewModel> htmlHelper, BikeViewModel viewModel)

{

var outerDiv = new TagBuilder("div");

outerDiv.AddCssClass("pull-left upload-img-wrapper");

var label = new TagBuilder("label");

label.AddCssClass("upload-img");

label.MergeAttribute("data-content", viewModel.ButtonText);

var image = new TagBuilder("img");

image.AddCssClass("img-responsive");

image.MergeAttribute("src", viewModel.imageSource);

image.MergeAttribute("width", "250");

image.MergeAttribute("height", "250");

var textbox = InputExtensions.TextBoxFor(htmlHelper, m => m.ImageName, new { type = "file", style = "display:none" });

StringBuilder htmlBuilder = new StringBuilder();

htmlBuilder.Append(label.ToString(TagRenderMode.StartTag));

htmlBuilder.Append(image.ToString(TagRenderMode.Normal));

htmlBuilder.Append(label.ToString(TagRenderMode.EndTag));

htmlBuilder.Append(textbox.ToHtmlString());

outerDiv.InnerHtml = htmlBuilder.ToString();

var html = outerDiv.ToString(TagRenderMode.Normal);

return MvcHtmlString.Create(html);

}

}

}

A couple of notes on this code fragment:

* ImageUpload is a new C# extension method that we created, it is added dynamically to the HtmlHelper class by using the this keyword. This allows us to use ImageUpload in the view.
* The TagBuilder class is used to create the HTML elements.
* It was even possible to use the output of another HTML helper, like the TextBoxFor function, by calling it directly from InputExtensions.

Finally, the elements are merged together into a single div and are returned as an HTML encoded string.

## **Refactoring the view to include the ImageUpload helper**

The new view, with the HTML block refactored into an ImageUpload HTML Helper, looks like this:

@model BikeWeb.ViewModels.BikeViewModel

@using BikeWeb.Extensions

@Html.ImageUpload(Model);

<rest of the page excluded for brevity>

After the refactor we went from five lines to one line. This makes the view a lot cleaner, and as an additional bonus the ImageUpload helper can now be reused on other pages too! By storing the HTML code in a single place, we can easily update the HTML implementation and the changes will be reflected on all pages that use the ImageUpload helper.

Some notes on this refactored view:

* @using BikeWeb.Extensions imports our extension method into this page
* @Html.ImageUpload calls this extension method

## **Conclusion**

In this blog post we have shown you what an HTML helper is and how you can create one in ASP.NET MVC. More specifically, if you follow the steps above you should now be able to create your own ImageUpload HTML helper. Knowing what an HTML helper is and how to use it, will make it substantially easier for you to refactor your Razor view.

MVC Data Annotions’lar Ve Açıklamarıyla Bol Örnekler

MVC’de en çok kullanılan kavramlardan biride hiç şüphesiz Attribute’ler olmaktadır bu makalemizde Data Annotions namespace’i altında yer alan Attribute’leri göreceğiz

**MVC’de Data Annotions’lar ve açıklamarıyla bol örnekler**

DataAnnotions Attribute’lerini kullanmak için alttaki namespace’leri ekleyin.

|  |  |  |
| --- | --- | --- |
| 1  2 | using System.ComponentModel.DataAnnotations; //eklemeyi unutma  using System.ComponentModel.DataAnnotations.Schema; //eklemeyi unutma | |
| **Attribute** | | **Açıklama** | |
| Key | | Alanın birincil anahtar olduğu belirtilir ve veritabanı oluşturulurken otomatik artan sayı olarak ayarlanır. | |
| Required | | Alanın zorunlu olduğu belirtilir | |
| Column | | Alanın veritabanındaki alanı ile ilgili ayarları belirtilir örnek veritabanında oluşacak olan alanadı(Name), tür adı(TypeName) veya Order(Sira – Çoklu Key kullanımında kullanılır) | |
| DataType | | Alanın veritürü belirtili örnek DataType’lar şu şekildedir. Date,Time,Currency,EmailAdress,Password v.b | |
| HiddenInput | | Alanın sayfada gösterilmemesini sağlar ama veri server’a gönderilir sadece kullanıcıdan gizlenir. | |
| ReadOnly | | Alanın sadece okunabilir özelliğine sahip olması için kullanılır. | |
| DisplayFormat | | Sayfaya bastırılacak fotmat bilgisi belirlenir genellikle Tarih, Saat, Sayı formatlama gibi durumlarda kullanılır. | |
| Table | | Veritabanında oluşacak tablonun adı(Name) veya Şema(Schema)’sı belirtilir. | |
| StringLength | | Girilecek karakter sayısını sınırlamada kullanılır. | |
| RegularExpression | | Regex neredeyse tüm programlama dillerinde kullanılan bir doğrulama desenidir. | |
| Range | | Sayısal değerler için aralık belirtmede kullanılır. | |
| Remote | | Kaydın varolup olmadığını kontrol etme örnek sayfa yenilemeden girilen kullanıcı adının var olup olmadığını kontrol etme. | |
| Compare | | Karşılaştırma yapmak için kullanılır örnek girilen ili şifrenin aynı olup olmamasını karşılaştırma. | |
| ScaffoldColumn | | Düzenlenmesini istemediğimiz model özelliği için kullanılır. Böylece bu alan EditorForModel kullandığımızda gizlenecektir. Örneğin, modelin id sini kendimiz belirlemiyoruz. Bu özelliği veritabanına bırakıyoruz. Dolayısıyla bu alan için ScaffoldColumn niteliği kullanabiliriz. | |

Sırasıyla üstteki tabloda yer alan attribute’lerimizi açıklamaya çalışalım

using System.ComponentModel.DataAnnotations; //eklemeyi unutma

using System.ComponentModel.DataAnnotations.Schema; //eklemeyi unutma

**Key**

|  |  |
| --- | --- |
| 1  2 | [Key]  public string Id { get; set; } |

Üstteki kodda Id alanının primary key(birincil anahtar) özelliğine sahip olduğunu belirtiyoruz böylece oluşturulacak veri tabanında bu alan otomatik olarka birincil anahtar olacak ve otomatik artan sayı olarak ayarlanacaktır.

Sırasıyla üstteki tabloda yer alan attribute’lerimizi açıklamaya çalışalım

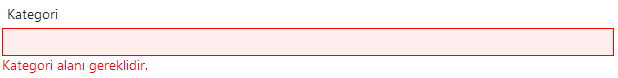
using System.ComponentModel.DataAnnotations; //eklemeyi unutma

using System.ComponentModel.DataAnnotations.Schema; //eklemeyi unutma

**Required**

|  |  |
| --- | --- |
| 1  2 | [Required(ErrorMessage = "{0} alanı gereklidir.")]  public string Kategori { get; set; } |

Üstteki şekilde bir tanımlama sonucunda Kategori alanının boş geçilemeyeceğini belirtiyoruz ve boş geçildiği taktirde Kategori alanı gereklidir şeklinde bir uyarı çıkacaktır. alttaki gibi



Eğer ErrorMessage tanımlamasakdık otomatik olarak şöyle bir cümle tanımlanmış ve gösterilmiş olacaktı “The Ad field is required.” null veya boş değer girilmeye çalışıldığında bu hata ile karşılaşmanız olasıdır.

**ReadOnly**

Bu niteliği kullandığımız özellik değiştirelemez, sadece görüntülenir. Kullanımı;

|  |  |
| --- | --- |
| 1  2 | [ReadOnly(true)]  public decimal Ucret { get; set; } |

**DisplayFormat**

using System.ComponentModel.DataAnnotations; //eklemeyi unutma

using System.ComponentModel.DataAnnotations.Schema; //eklemeyi unutma

Model özelliğini formatlı göstermek için kullanabiliriz. Örneğin;

|  |  |
| --- | --- |
| 1  2 | [DisplayFormat(ApplyFormatInEditMode=true, DataFormatString="{0:c}")]  public decimal Ucret { get; set; } |

Şeklinde bir kullanımda, bu alanın çıktısı şöyle olacaktır:

12.50 TL

ApplyFormatInEditMode özelliği varsayılan olarak false dur. Eğer bu formatlı görüntünün input içerisinde de görünmesini istiyorsak true yaparız.

Şimdi ise DogumTarihini dd/MM/yyyy formatında yazdıralım eğer format belirtmezsek dd/MM/yyyy hh:mm:nn formatında uzunca gereksiz rakamlar çıkacaktır.

|  |  |
| --- | --- |
| 1  2 | [DisplayFormat(DataFormatString="{0:d}", NullDisplayText = "Dogum Tarihi Girilmemiş")]  public DateTime DogumTarihi { get; set; } |

Tarih için üstteki formatı kullanırsanız çıktı şu şekilde olacaktır.

02-08-2014

NullDisplayText özelliğini kullanarak NULL dönen sonuçlar yerine bir not gösterilmesini sağlayabilirsiniz.

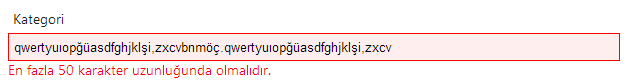
**StringLength**

Bu nitelik ile girilecek değerin karakter uzunluğunu belirleriz. Bu tanımlama ile aynı zamanda Code First geliştirmede veritabanın oluşturulacak alan içinde, bu string uzunluğu tanımlanacaktır.

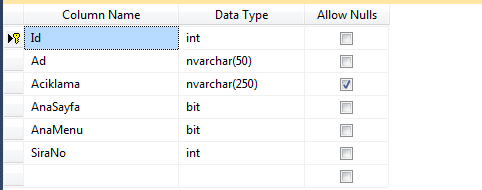
Eğer bu nitelik için maksimum karakter uzunluğu belirlenmezse, doğrulama işleminin yanı sıra veritabanı tablosundaki bu alan için nvarchar(MAX) şeklinde bir veri tipi oluşturulacaktır. Örneğin kategori sınıfı için;

|  |  |
| --- | --- |
| 1  2  3  4 | [Required(ErrorMessage = "{0} alanı gereklidir.")]  [StringLength(50, ErrorMessage = "En fazla {1} karakter  uzunluğunda olmalıdır.")]  public string Ad { get; set; } |

şeklinde bir tanımlama yaptığımızda, 50 karakterden fazla bir girdi yazarsak alacağımız mesaj;



gibi olacaktır. Ayrıca veritabanı tablosundaki ad alanının görüntüsüde aşağıdaki gibidir.



MinimumLength özelliğide isteğe bağlıdır. Bu özellik StringLength nitaliğine bağlı bir özelliktir. Kullanımıda aşağıdaki gibidir.

|  |  |
| --- | --- |
| 1  2  3  4 | [Required(ErrorMessage = "{0} alanı gereklidir.")]  [StringLength(50, ErrorMessage = "En fazla {1} karakter  uzunluğunda olmalıdır.",MinimumLength=3)]  public string Ad { get; set; } |

**RegularExpression**

Regex tüm dillerde olan bir string doğrulama desenidir. MVC de de regex için bir nitelik (attribute) vardır. Regex girilen string değerin, bizim belirlediğimiz bir kurala göre olup olmadıgını belirlemek amaçlı kullanılır.

Aslında bir mail adresinin gerçek olup olmadığı o adrese mail atıp cevap gelene kadar beklemeden anlaşılmaz. Ama en azından belli bir e-posta deseni vardır. @ sembolune kadar olan kısım kullanıcı kimligi, bu sembolden sonra, mail sirketinin kimliği, daha sonra bir nokta ve bir uzantı.

E-posta adresinin regex deseni [A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+.[A-Za-z]{2,4} şeklindedir. Bunu sınıfımızın özelliği için nitelik olarak belirlemek için;

|  |  |
| --- | --- |
| 1  2 | [RegularExpression(@"[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+.[A-Za-z]{2,4}")]  public string Eposta { get; set; } |

şeklinde kullanırız. Böylece MVC kullanıcı girdisinin bu desene uyup uymadıgını kontrol eder. Ve önceki örneklerimizde olduğu gibi bir hata mesajı verir. Tabi aslında Email için direk yazılmış nitalik var. Bunlara de deyineceğiz.

**Remote**

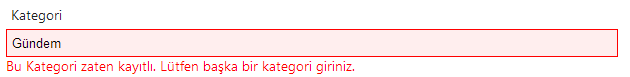
Bu nitelik sayısal değerler için aralık belirlemek için kullanılır. Örneğin bir yaş aralığı için; Bu nitelik, sunucu taraflı bir çağrı ile istemci taraflı bir doğrulama yapmak amacıyla kullanılır. Yani; örneğin, KullaniciAdi var mı yok mu diye, kontrol etmek için veritabanına gitmek gerekli. Bunu yaparken ise istemci taraflı bir doğrulama yapmak için bu niteliği kullanabiliriz. Örneğin kategori ismi için, aynı kategoriyi tekrar girilmesini istemiyorsak, kategori sınıfımızdaki Ad alanını Remote ekliyoruz;

|  |  |
| --- | --- |
| 1  2  3  4  5 | [Remote("KategoriVarmi","Kategori","Admin", ErrorMessage = "Bu {0} zaten kayıtlı. Lütfen başka bir kategori giriniz.")]  [Required(ErrorMessage = "{0} alanı gereklidir.")]  [StringLength(50, ErrorMessage = "En fazla {1} karakter uzunluğunda olmalıdır.")]  [Display(Name = "Kategori")]  public string Ad { get; set; } |

Daha sonra KategoriController sınıfına aşağıdaki metodu ekliyoruz.

|  |  |
| --- | --- |
| 1  2  3  4  5 | public JsonResult KategoriVarmi(string kategoriAd)  {      var result = kategoriServis.KategoriVarmi(kategoriAd);      return Json(result, JsonRequestBehavior.AllowGet);  } |

Kategori eğer sistemde varsa, sayfa yenilemesi olmadan bu karşılaştırma yapılarak, hata mesajı verecektir. Hata mesajı aşağıdaki gibidir.



**Range**

Range attribute’ünü özellikle yaş gibi bu durumlarda kullanabilirsiniz mesela projenizde kayıt işleminde yaş aralığı 18-65 arasındaki kişileri kabul ediyorsanız aşağıdaki gibi bir rande kullanımı tam ihtiyacınızı karşılayacaktır.

|  |  |
| --- | --- |
| 1 | [Range(18, 65, ErrorMessage = "Yaş 18'den küçük 65'ten büyük olamaz.")] |

Eğer iki tarih aralığında bir tarih girilmesini istersek o zaman aşağıdaki gibi bir yazım ile ihtiyacımızı karşılayabiliriz.

|  |  |
| --- | --- |
| 1 | [Range(typeof(DateTime), '01/01/2000', '01/01/2010')] |

Üstteki kodda yapılmak istenen çok basit DateTime türünde 01/01/2000 ile 01/01/2010 tarihleri arasındaki bir tarih girdirilmesini sağlayabiliriz kolaylıkla.

**Compare**

Bu nitelik ile örneğin kullanıcı şifresini tekrar girmesini istedigimizde bu iki şifre karşılaştırılır. Örnek kullanımı;

|  |  |
| --- | --- |
| 1  2  3  4 | public string Sifre { get; set; }    [Compare("Sifre")]  public string SifreTekrar { get; set; } |

şeklinde bir tanımlama yaparsak, iki textbox içine girilen değerler otomatik olarak karşılaştırılır.

**ScaffoldColumn**

Düzenlenmesini istemediğimiz model özelliği için kullanılır. Böylece bu alan EditorForModel kullandığımızda gizlenecektir. Örneğin, modelin id sini kendimiz belirlemiyoruz. Bu özelliği veritabanına bırakıyoruz. Dolayısıyla bu alan için ScaffoldColumn niteliği kullanabiliriz. Örnek kullanımı;

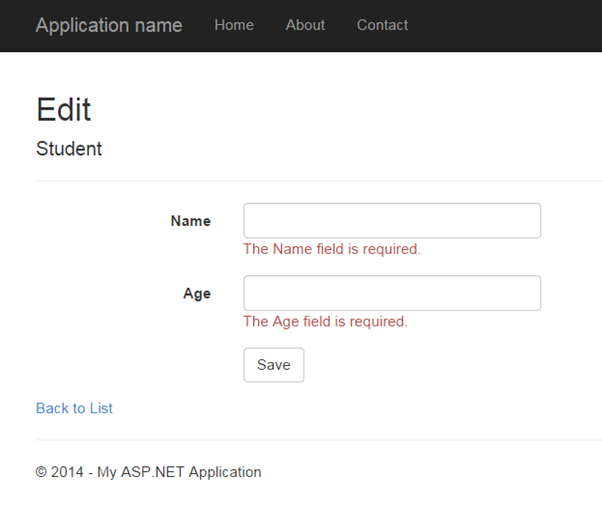
|  |  |
| --- | --- |
| 1  2 | [ScaffoldColumn(false)]  public int Id { get; set; } |

**//////////////////////////////////////////////////////////////////////////////**

Implement Data Validation in MVC

In this section, you will learn how to implement data validations in the ASP.NET MVC application.

We have created an Edit view for Student in the previous section. Now, we will implement data validation in the Edit view, which will display validation messages on the click of Save button, as shown below if Student Name or Age is blank.

[](https://www.tutorialsteacher.com/Content/images/mvc/validation.png)Validation

DataAnnotations

ASP.NET MVC uses DataAnnotations attributes to implement validations. DataAnnotations includes built-in validation attributes for different validation rules, which can be applied to the properties of model class. ASP.NET MVC framework will automatically enforce these validation rules and display validation messages in the view.

The DataAnnotations attributes included in *System.ComponentModel.DataAnnotations* namespace. The following table lists DataAnnotations validation attributes.

| **Attribute** | **Description** |
| --- | --- |
| Required | Indicates that the property is a required field |
| StringLength | Defines a maximum length for string field |
| Range | Defines a maximum and minimum value for a numeric field |
| RegularExpression | Specifies that the field value must match with specified Regular Expression |
| CreditCard | Specifies that the specified field is a credit card number |
| CustomValidation | Specified custom validation method to validate the field |
| EmailAddress | Validates with email address format |
| FileExtension | Validates with file extension |
| MaxLength | Specifies maximum length for a string field |
| MinLength | Specifies minimum length for a string field |
| Phone | Specifies that the field is a phone number using regular expression for phone numbers |

Let's start to implement validation in Edit view for student.

**Step 1:** First of all, apply DataAnnotation attribute on the properties of Student model class. We want to validate that StudentName and Age is not blank. Also, Age should be between 5 and 50. Visit [Model](https://www.tutorialsteacher.com/mvc/mvc-model) section if you don't know how to create a model class.

Example: Apply DataAnnotation Attributes

public class Student

{

public int StudentId { get; set; }

[Required]

public string StudentName { get; set; }

[Range(5,50)]

public int Age { get; set; }

}

You can also apply multiple DataAnnotations validation attributes to a single property if required.

In the above example, we have applied a ***Required*** attribute to the StudentName property. So now, the MVC framework will automatically display the default error message, if the user tries to save the Edit form without entering the Student Name. In the same way, the ***Range*** attribute is applied with a min and max value to the Age property. This will validate and display an error message if the user has either not entered Age or entered an age less than 5 or more than 50.

**Step 2:** Create the GET and POST Edit Action method in the same as previous section. The GET action method will render Edit view to edit the selected student and the POST Edit method will save edited student as shown below.

Example: Edit Action methods:

using MVC\_BasicTutorials.Models;

namespace MVC\_BasicTutorials.Controllers

{

public class StudentController : Controller

{

public ActionResult Edit(int id)

{

var std = studentList.Where(s => s.StudentId == StudentId)

.FirstOrDefault();

return View(std);

}

[HttpPost]

public ActionResult Edit(Student std)

{

if (ModelState.IsValid) {

//write code to update student

return RedirectToAction("Index");

}

return View(std);

}

}

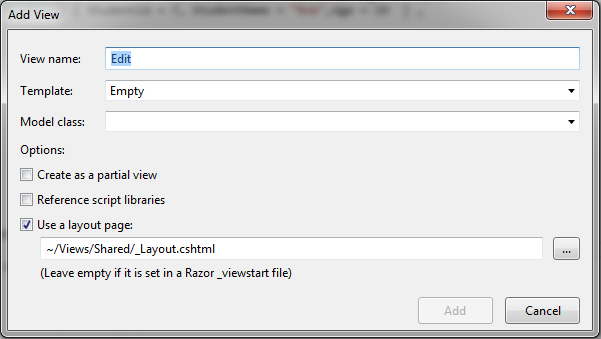
}

As you can see in the POST Edit method, we first check if the ModelState is valid or not. If ModelState is valid then update the student into database, if not then return Edit view again with the same student data.

ModelState.IsValid determines that whether submitted values satisfy all the DataAnnotation validation attributes applied to model properties.

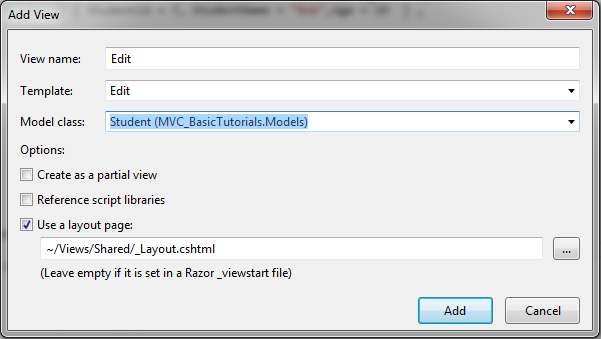
**Step 3:** Now, create an Edit view for Student.

To create an Edit view, right click inside Edit action method -> click **Add View..**

[](https://www.tutorialsteacher.com/Content/images/mvc/addview.png)Create Edit View

In the Add View dialogue, keep the view name as Edit. (You can change as per your requirement.)

Select the Edit template in the Template dropdown and also select Student Model class as shown below.

[](https://www.tutorialsteacher.com/Content/images/mvc/addview-2.png)Create Edit View

Now, click **Add** to generate Edit view under View/Student folder. Edit.cshtml will be generated as shown below.

Edit.cshtml:

@model MVC\_BasicTutorials.Models.Student

@{

ViewBag.Title = "Edit";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<h2>Edit</h2>

@using (Html.BeginForm())

{

@Html.AntiForgeryToken()

<div class="form-horizontal">

<h4>Student</h4>

<hr />

**@Html.ValidationSummary(true, "", new { @class = "text-danger" })**

@Html.HiddenFor(model => model.StudentId)

<div class="form-group">

@Html.LabelFor(model => model.StudentName, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.StudentName, new { htmlAttributes = new { @class = "form-control" } })

**@Html.ValidationMessageFor(model => model.StudentName, "", new { @class = "text-danger" })**

</div>

</div>

<div class="form-group">

@Html.LabelFor(model => model.Age, htmlAttributes: new { @class = "control-label col-md-2" })

<div class="col-md-10">

@Html.EditorFor(model => model.Age, new { htmlAttributes = new { @class = "form-control" } })

**@Html.ValidationMessageFor(model => model.Age, "", new { @class = "text-danger" })**

</div>

</div>

<div class="form-group">

<div class="col-md-offset-2 col-md-10">

<input type="submit" value="Save" class="btn btn-default" />

</div>

</div>

</div>

}

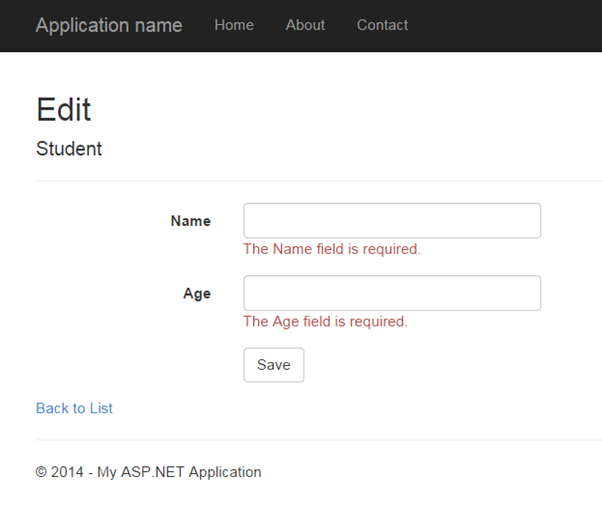
<div>

@Html.ActionLink("Back to List", "Index")

</div>

As you can see in the above Edit.cshtml, it calls Html Helper method **ValidationMessageFor** for every field and **ValidationSummary** method at the top. ValidationMessageFor is responsible to display error message for the specified field. ValidationSummary displays a list of all the error messages at once.

So now, it will display default validation message when you submit an Edit form without entering a Name or Age.

[](https://www.tutorialsteacher.com/Content/images/mvc/validation-editview.png)Validation

Thus, you can implement validations by applying various DataAnnotation attributes to the model class and using ValidationMessage() or ValidationMessageFor() method in the view.

Learn how to [implement client side validation in ASP.NET MVC](https://www.tutorialsteacher.com/articles/enable-client-side-valiation-in-mvc).

 Points to Remember :

1. ASP.NET MVC uses DataAnnotations attributes for validation.
2. DataAnnotations attributes can be applied to the properties of the model class to indicate the kind of value the property will hold.
3. The following validation attributes available by default
   1. Required
   2. StringLength
   3. Range
   4. RegularExpression
   5. CreditCard
   6. CustomValidation
   7. EmailAddress
   8. FileExtension
   9. MaxLength
   10. MinLength
   11. Phone
4. Use **ValidationSummary** to display all the error messages in the view.
5. Use **ValidationMessageFor** or **ValidationMessage** helper method to display field level error messages in the view.
6. Check whether the model is valid before updating in the action method using ModelState.IsValid.
7. Enable client side validation to display error messages without postback effect in the browser.

ASP.NET MVC: ValidationMessage

You have learned how to implement validation in a view in the presious section. Here, we will see the HtmlHelper extension method ValidtionMessage in detail.

The Html.ValidationMessage() is an extension method, that is a loosely typed method. It displays a validation message if an error exists for the specified field in the ModelStateDictionary object.

ValidationMessage() Signature

MvcHtmlString ValidateMessage(string modelName, string validationMessage, object htmlAttributes)

Visit MSDN to know all the [overloads of ValidationMessage() method](https://msdn.microsoft.com/en-us/library/system.web.mvc.html.validationextensions.validationmessage(v=vs.118).aspx" \t "_blank).

Example: ValidationMessage

@model Student

@Html.Editor("StudentName") <br />

@Html.ValidationMessage("StudentName", "", new { @class = "text-danger" })

In the above example, the first parameter in the ValidationMessage method is a property name for which we want to show the error message e.g. StudentName. The second parameter is for custom error message and the third parameter is for html attributes like css, style etc.

The ValidationMessage() method will only display an error, if you have configured the DataAnnotations attribute to the specifed property in the model class. The following is a Student model class where the DataAnnotations attribute "Required" is applied to the StudentName property.

Example: Student Model

public class Student

{

public int StudentId { get; set; }

[Required]

public string StudentName { get; set; }

public int Age { get; set; }

}

The above code will generate following html.

<input id="StudentName"

name="StudentName"

type="text"

value="" />

<span class="field-validation-valid text-danger"

data-valmsg-for="StudentName"

data-valmsg-replace="true">

</span>

Now, when the user submits a form without entering a StudentName, then ASP.NET MVC uses a data- attribute of Html5 for the validation and a default validation message will be injected, when the validation error occurs, as shown below.

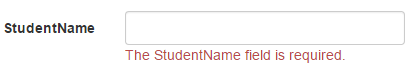
Html with Validation message:

<span class="field-validation-error text-danger"

data-valmsg-for="StudentName"

data-valmsg-replace="true">The StudentName field is required.</span>

The error message will look like below.

[](https://www.tutorialsteacher.com/Content/images/mvc/validationmessage.png)Output of ValidationMessage() method

Custom Error Message

You can display your own error message instead of the default error message as shown above. You can provide a custom error message either in the DataAnnotations attribute or ValidationMessage() method.

Use the parameter of the DataAnnotation attributes to provide your own custom error message as shown below.

Example: Custom error message in the Model

public class Student

{

public int StudentId { get; set; }

[Required(ErrorMessage="Please enter student name.")]

public string StudentName { get; set; }

public int Age { get; set; }

}

Also, you can specify a message as a second parameter in the ValidationMessage() method as shown below.

Example: Custom error message

@model Student

@Html.Editor("StudentName") <br />

@Html.ValidationMessage("StudentName", "Please enter student name.", new { @class = "text-danger" })

ASP.NET MVC: ValidationMessageFor

The Html.ValidationMessageFor() is a strongly typed extension method. It displays a validation message if an error exists for the specified field in the ModelStateDictionary object.

ValidationMessageFor() Signature

*MvcHtmlString ValidateMessage(Expression<Func<dynamic,TProperty>> expression, string validationMessage, object htmlAttributes)*

Visit MSDN to know all the [overloads of ValidationMessageFor() method](https://msdn.microsoft.com/en-us/library/system.web.mvc.html.validationextensions.validationmessagefor(v=vs.118).aspx" \t "_blank).

Consider the following ValidationMessageFor() example.

Example: ValidationMessageFor

@model Student

@Html.EditorFor(m => m.StudentName) <br />

@Html.ValidationMessageFor(m => m.StudentName, "", new { @class = "text-danger" })

In the above example, the first parameter in ValidationMessageFor method is a lambda expression to specify a property for which we want to show the error message. The second parameter is for custom error message and the third parameter is for html attributes like css, style etc.

The ValidationMessageFor() method will only display an error if you have configured DataAnnotations attribute to the specifed property in the model class. The following example is a Student model class where the DataAnnotations attribute "Required" is applied to the StudentName property.

Example: Student Model

public class Student

{

public int StudentId { get; set; }

[Required]

public string StudentName { get; set; }

public int Age { get; set; }

}

The above code will generate the following html.

Html Result:

<input id="StudentName"

name="StudentName"

type="text"

value="" />

<span class="field-validation-valid text-danger"

data-valmsg-for="StudentName"

data-valmsg-replace="true">

</span>

Now, when the user submits a form without entering the StudentName then ASP.NET MVC uses the data- attribute of Html5 for the validation and the default validation message will be injected when validation error occurs, as shown below.

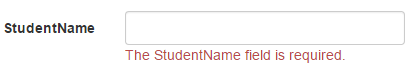
Html with Validation message:

<span class="field-validation-error text-danger"

data-valmsg-for="StudentName"

data-valmsg-replace="true">The StudentName field is required.</span>

The error message will appear as the image shown below.

[](https://www.tutorialsteacher.com/Content/images/mvc/validationmessage.png)Output of ValidationMessageFor() method

Custom Error Message

You can display your own error message instead of the default error message as above. You can provide a custom error message either in the DataAnnotations attribute or the ValidationMessageFor() method.

Use the ErrorMessage parameter of the DataAnnotation attributes to provide your own custom error message as shown below.

Example: Custom error message in the Model

public class Student

{

public int StudentId { get; set; }

[Required(ErrorMessage="Please enter student name.")]

public string StudentName { get; set; }

public int Age { get; set; }

}

Also, you can specify a message as a second parameter in the ValidationMessage() method as shown below.

Example: Custom error message

@model Student

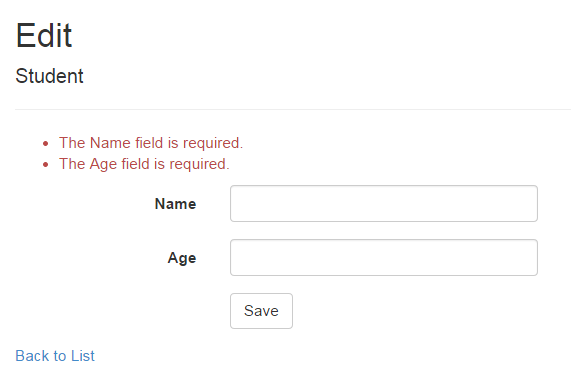
@Html.Editor("StudentName") <br />

@Html.ValidationMessageFor(m => m.StudentName, "Please enter student name.", new { @class = "text-danger" })

# ASP.NET MVC: ValidationSummary

The ValidationSummary helper method generates an unordered list (ul element) of validation messages that are in the ModelStateDictionary object.

The ValidationSummary can be used to display all the error messages for all the fields. It can also be used to display custom error messages. The following figure shows how ValidationSummary displays the error messages.

[](https://www.tutorialsteacher.com/Content/images/mvc/validationsummary.png)ValidationSummary

## **ValidationSummary() Signature**

*MvcHtmlString ValidateMessage(bool excludePropertyErrors, string message, object htmlAttributes)*

Visit MSDN to know all the [overloads of ValidationMessage() method](https://msdn.microsoft.com/en-us/library/system.web.mvc.html.validationextensions.validationsummary(v=vs.118).aspx" \t "_blank).

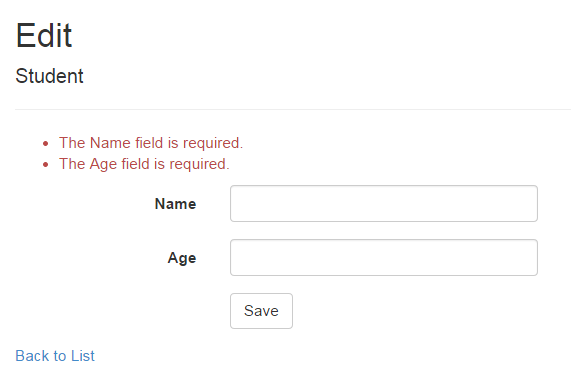
## **Display Field Level Error Messages using ValidationSummary**

By default, ValidationSummary filters out field level error messages. If you want to display field level error messages as a summary then specify excludePropertyErrors = false.

Example: ValidationSummary to display field errors

@Html.ValidationSummary(false, "", new { @class = "text-danger" })

So now, the following Edit view will display error messages as a summary at the top. Please make sure that you don't have a ValidationMessageFor method for each of the fields.

[](https://www.tutorialsteacher.com/Content/images/articles/validationsummary-demo.png)Error Message using ValidationSummary

## **Display Custom Error Messages**

You can also display a custom error message using ValidationSummary. For example, we want to display a message if Student Name already exists in the database.

To display a custom error message, first of all, you need to add custom errors into the ModelState in the appropriate action method.

Example: Add error in ModelState

if (ModelState.IsValid) {

//check whether name is already exists in the database or not

bool nameAlreadyExists = \* check database \*

if(nameAlreadyExists)

{

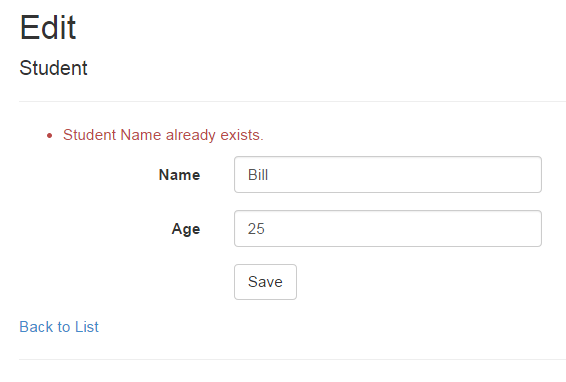
ModelState.AddModelError(string.Empty, "Student Name already exists.");

return View(std);

}

}

As you can see in the above code, we have added custom error messages using the ModelState.AddModelError method. The ValidationSummary method will automatically display all the error messages added into ModelState.

[](https://www.tutorialsteacher.com/Content/images/articles/validationsummary-demo2.png)Display error message using ValidationSymmary

Thus, you can use the ValidationSummary helper method to display error messages.

**//////////////////////////////////////////////////////////////////////////////**