**Properties** are named members of classes, structures, and interfaces. Member variables or methods in a class or structures are called **Fields**. Properties are an extension of fields and are accessed using the same syntax. Properties do not name the storage locations. Instead, they have **accessors** that read, write, or compute their values.

Properties in C# and .NET have various access levels that is defined by an access modifier. Properties can be read-write, read-only, or write-only. The read-write property implements both, a get and a set accessor. A write-only property implements a set accessor, but no get accessor. A read-only property implements a get accessor, but no set accessor.

Properties Access Modifiers

Access modifiers defines the access level of a property whether a property can be accessed by any caller program, within an assembly, or just within a class.

The following table describes access level modifiers.

* public - The type or member can be accessed by any other code in the same assembly or another assembly that references it.
* Private - The type or member can be accessed only by code in the same class or struct.
* protected - The type or member can be accessed only by code in the same class, or in a class that is derived from that class.
* internal - The type or member can be accessed by any code in the same assembly, but not from another assembly.
* protected internal - The type or member can be accessed by any code in the assembly in which it is declared, or from within a derived class in another assembly.
* private protected - The type or member can be accessed only within its declaring assembly, by code in the same class or in a type that is derived from that class.

1. **namespace** myprop
2. {
3. **public** **class** Person
4. {
5. **private** **string** mName = **string**.Empty;
6. **private** **int** mAge = 0;
8. **public** **string** Name
9. {
10. **get**
11. {
12. **return** mName;
13. }
14. **set**
15. {
16. mName = value;
17. }
18. }
20. **public** **int** Age
21. {
22. **get**
23. {
24. **return** mAge;
25. }
26. **set**
27. {
28. mAge = value;
29. }
30. }
31. }
32. }

Create Readonly Property

We can also create a read only property. Read only means that we can access the value of a property but we can't assign a value to it. When a property does not have a set accessor then it is a read only property. For example in the person class we have a Gender property that has only a get accessor and doesn't have a set accessor. The Person class is:

1. **public** **class** Person
2. {
3. **public** **string** Gender
4. {
5. **get** { **return** "Male";}
6. }
7. }

Create WriteOnly Property

We can also create a write only property. A write only property is a property that we can assign a value to but can't get that value because that property doesn't have a get accessor. For example we have a Person class that has the property FirstName that has a set accessor but doesn't have a get accessor so it is a write only property.

1. **public** **class** Person
2. {
3. **private** **string** mFirstName = **string**.Empty;
4. **public** **string** FirstName
5. {
6. **set**{mFirstName = value;}
7. }
8. }

Validate Property Value

We can validate a value of a property before it's set to a variable; in other words, we can check a value to be assigned to a private field and if it is correct then it will be assigned to the private field, otherwise it will give an error.  
  
Suppose you are a citizen of India and participate as a voter in a parliament election so your age should be greater than or equal to 18 otherwise you can't vote. To implement this function we have a Voter class and that class has an Age property. The Age property can have a value greater than or equal to 18 otherwise it will show 0 for the age with a message. Our Voter class is:

1. **using** System;
2. **namespace** PropertyExample
3. {
4. **public** **class** Voter
5. {
6. **private** **int** mAge = 0;
7. **public** **int** Age
8. {
9. **get**
10. {
11. **return** mAge;
12. }
13. **set**
14. {
15. **if** (value >= 18)
16. {
17. mAge = value;
18. }
19. **else**
20. {
21. Console.WriteLine("You are not eligible for voting");
22. }
23. }
24. }
25. }
26. }

Now create an executable program by which we assign an age value of a Voter.

1. **using** System;
2. **namespace** PropertyExample
3. {
4. **class** Program
5. {
6. **static** **void** Main(**string**[] args)
7. {
8. Voter objVoter = **new** Voter();
9. Console.WriteLine("Please enter your age");
10. objVoter.Age = Convert.ToInt32(Console.ReadLine());
11. Console.WriteLine("Your age is :{0} years", objVoter.Age);
12. Console.ReadKey();
13. }
14. }
15. }

Use of Property as a method

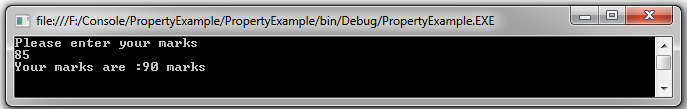
We can also use a property instead of a method that has one parameter and a return value, because we can define custom logic in the get and set accessors.  
  
Suppose we have a rule for a student that when a student gets marks greater than or equal to 80 then they get 5 bonus marks and when they get marks greater than or equal to 70 but less than or equal to 79 then they get 2 bonus marks. So we define logic in the get accessor that adds marks depending on the student's original marks and thereafter the total marks (original marks + bonus marks) of the student will be returned.

1. **using** System;
2. **namespace** PropertyExample
3. {
4. **public** **class** Student
5. {
6. **private** **int** mMarks = 0;
7. **public** **int** Marks
8. {
9. **get**
10. {
11. **if**(mMarks >= 80)
12. {
13. mMarks+=5;
14. }
15. **else** **if**(mMarks <= 79 && mMarks>=70 )
16. {
17. mMarks+=2;
18. }
19. **return** mMarks;
20. }
21. **set**
22. {
23. mMarks = value;
24. }
25. }
26. }
27. }

Now create a program by which we can input student marks.

1. **using** System;
2. **namespace** PropertyExample
3. {
4. **class** Program
5. {
6. **static** **void** Main(**string**[] args)
7. {
8. Student objStudent = **new** Student();
9. Console.WriteLine("Please enter your marks");
10. objStudent.Marks = Convert.ToInt32(Console.ReadLine());
11. Console.WriteLine("Your marks are :{0} marks", objStudent.Marks);
12. Console.ReadKey();
13. }
14. }
15. }

Now we enter more than 80 marks and get the total marks with bonus.



Auto mapped property

We can also create auto mapped property. When we create auto mapped property then we don't need define to local private field in get and set accessors. For example we have an employee class which has two properties,  one in EmployeeId and another is Name. To define these properties we don't need to create a private field for these two properties and directly declare it.

1. **namespace** PropertyExample
2. {
3. **public** **class** Employee
4. {
5. **public** **int** EmployeeId { **get**; **set**; }
6. **public** **string** Name { **get**; **set**; }
7. }
8. }

Now we access Employee class property in executable program.

1. **using** System;
2. **namespace** PropertyExample
3. {
4. **class** Program
5. {
6. **static** **void** Main(**string**[] args)
7. {
8. Employee objEmployee = **new** Employee()
9. {
10. EmployeeId = 1001,
11. Name = "Sandeep"
12. };
13. Console.WriteLine("Employee Id is :{0} and Employee Name is :{1}", objEmployee.EmployeeId, objEmployee.Name);
14. Console.ReadKey();
15. }
16. }
17. }

## Static Properties

C# also supports static properties, which belongs to the class rather than to the objects of the class. All the rules applicable to a static member are applicable to static properties also.

1. //C# : static Property
2. **using** System;
3. **class** MyClass
4. {
5. **private** **static** **int** x;
6. **public** **static** **int** X
7. {
8. **get**
9. {
10. **return** x;
11. }
12. **set**
13. {
14. x = value;
15. }
16. }
17. }
18. **class** MyClient
19. {
20. **public** **static** **void** Main()
21. {
22. MyClass.X = 10;
23. **int** xVal = MyClass.X;
24. Console.WriteLine(xVal);//Displays 10
25. }
26. }
27. **using** System;
28. **class** Author {
29. // Read-write properties
30. **public** **string** Name {
31. **get**;
32. **set**;
33. }
34. **public** **string** Publisher {
35. **get**;
36. **set**;
37. }
38. **public** **string** Book {
39. **get**;
40. **set**;
41. }
42. **public** Int16 Year {
43. **get**;
44. **set**;
45. }
46. **public** **double** Price {
47. **get**;
48. **set**;
49. }
50. **public** **string** PriceInString {
51. **get** {
52. **return** **string**.Format("${0}", Price);
53. }
54. }
55. // Read-only properties
56. **public** **string** Names {
57. **get**;
58. }
59. // Initialization of a property
60. **public** **double** AuthorCount {
61. **get**;
62. **private** **set**;
63. } = 99;
64. // Class constructor
65. **public** Author(**string** name, **string** publisher, **string** book, Int16 year, **double** price) {
66. Name = name;
67. Publisher = publisher;
68. Book = book;
69. Year = year;
70. Price = price;
71. }
72. // Public methods
73. **public** **string** AuthorDetails() {
74. **return** **string**.Format("{0} is an author of {1} published by {2} in year {3}. Price: ${4}", Name, Book, Publisher, Year, Price);
75. }
76. **public** **double** CostOfThousandBooks() {
77. **if** (Price > 0) **return** Price \* 1000;
78. **return** 0;
79. }
80. }

The code in Listing is creates an instance of the class and calls its methods and properties.

1. **class** Program
2. {
3. **static** **void** Main()
4. {
5. Author author = **new** Author("Mahesh Chand", "Apress", "Programming C#", 2003, 49.95);
6. Console.WriteLine(author.AuthorDetails());
7. Console.WriteLine("Author published his book in {0}", author.Year);
8. author.Price = 50;
9. Console.WriteLine(author.CostOfThousandBooks().ToString());
10. Console.ReadKey();
11. }
12. }