A DEEP DIVE INTO C# PROPERTY

A Property acts as a wrapper around a field. It is used to assign and read the value from a field by using set and get accessors. The code block for the get accessor is executed when the property is read and the code block for the set accessor is executed when the property is assigned a new value. A property can be created for a public, private, protected and internal field.
Unlike fields, properties do not denote storage locations and you cannot pass a property as a ref or out parameter.

```
1. using System;
2.
3. class Example
4. {
5.  string name;
6.  public string Name
7.  {
8.  get { return name; }
9.  set { name = value; }
10.   }
11.  }
12.
13.  class Program
14.  {
15.   static void Main()
16.   {
17.   Example obj= new Example();
18.   obj.Name = "Dot Net Tricks"; // called set { }
19.   Console.WriteLine(obj.Name); // called get { }
20.   }
21.  }
```

When to use

1. Need to validate data before assigning it to a field.
2. Need to do intermediate computation on data before assigning or retrieving it to a field.
3. Need to log all access for a field.

4. Need to protect a field by reading and writing .

Different types of properties

Properties can be divided into three categories read-only, write-only, and read-write properties.

1. Read-Only Property

A read-only property allows you to only retrieve the value of a field. To create a read-only property, you should define the get accessor.

```
1. class Example
2. {
3.  string name;
4.  public string Name
5.  {
6.  get { return name; }
7.  }
8. }
```

2. Write-Only Property

A write-only property allows you to only change the value of a field. To create a write-only property, you should define the set accessor.

```
1. class Example
2. {
3.  string name;
4.  public string Name
5.  {
6.  set{ name = value; }
7.  }
8. }
```

3. Read-Write Property

A read-write property allows you to assign and read the value of a field. To create a read-write property, you should define the set and get accessors.

```
1. class Example
2. {
3.  string name;
4.  public string Name
5.  {
```

```
6.  get { return name; }
7.  set { name = value; }
8.  }
9. }
```

Auto-implemented property

Auto-implemented properties was introduced with C# 3.0, which make property declaration more concise. Unlike standard property, in auto-implemented property, wrapped or backing field is automatically created by the compiler but it is not available for use by the class's members.

```
1. public int Name { get; set; }
```

To make an auto-implemented property read-only or write-only, you need to specify both get and set accessors.

```
1. public int ReadOnly { get; private set; }
2. public int WriteOnly { private get; set; }
```

When to use Auto-implemented property

Simply, you need to store a value. No additional functionality can be added to either of the accessors.

Static property

You can also declare a property static. To make a static property you must ensure that the backing store field is also static. Typically, a static property is used to make a singleton class.

```
1. public class Singleton
2. {
3.  private static Singleton instance = new Singleton();
4.  private Singleton() { }
5.
6.  public static Singleton GetInstance
7.  {
8.  get { return instance; }
9.  }
10.  }
```

Abstract property

An abstract property declaration does not provide an implementation of the property accessors. It leaves the accessors implementation to derived classes. Abstract properties are declared with in a abstract class or interface.

```
1. public abstract class Person
2. {
3.  public abstract string Name{ get; set;}
4.
5. }
6.
7. class Student : Person
8. {
9.  private string name;
10.
11.   // Override Name property
12.   public override string Name
13.   {
14.   get { return name; }
15.   set { name = value; }
16.   }
17.  }
```

```
1. public interface IPerson
2. {
3.  string Name { get; set; }
4.
5. }
6.
7. class Student : IPerson
8. {
9.  private string name;
10.
11.   // implement Name property
12.   public string Name
```

```
13.   {
14.   get { return name; }
15.   set { name = value; }
16.   }
17.  }
```

Source : http://www.dotnet-tricks.com/Tutorial/csharp/KM1c261113-A-Deep-Dive-into-C#-Property.html