C# Properties (GET, SET)

In c#, **Property** is an extension of the class variable (/tutorial/csharp/csharp-variables-with-examples). It provides a mechanism to read, write, or change the class variable's value without affecting the external way of accessing it in our applications.

In c#, properties can contain one or two code blocks called **accessors,** and those are called a `get` accessor and `set` accessor. By using `get` and `set` accessors, we can change the internal implementation of class variables (/tutorial/csharp/csharp-variables-with-examples) and expose them without affecting the external way of accessing it based on our requirements.

Generally, in object-oriented programming languages like c#, you need to define fields as **private** and then use properties to access their values in a **public** way with `get` and `set` accessors.

Following is the syntax of defining a **property** with `get` and `set` accessor in c# programming language.

```
<access_modifier> <return_type> <property_name>
{
    get
    {
       //Return the property value
    }
    set
    {
      //Set a new value
    }
}
```

If you observe the above syntax, we used an access modifier (/tutorial/csharp/csharp-access-modifiers-public-private-protected-internal) and **return type** to define a property along with `get` and `set` accessors to make required modifications to the class variables (/tutorial/csharp/csharp-variables-with-examples) based on our requirements.

Here, the `get` accessor code block will be executed whenever the property is read, and the code block of `set` accessor will be executed whenever the property is assigned to a new value.

In c#, the properties are categorized into three types, those are.

Type	Description
Read-Write	A property that contains a both `get` and `set` accessors, then we will call it a read-write property.
Read-Only	A property that contains only `get` accessor, then we will call it a read-only property.
Write-Only	A property that contains only `set` accessor, then we will call it a write-only property.

In c#, Properties won't accept any parameters, and we should not pass a property as a ref (/tutorial/csharp/csharp-pass-by-reference-ref-with-examples) or out (/tutorial/csharp/csharp-out-parameter-with-examples) parameter in our application.

Following is a simple example of defining a private variable and a property in the c# programming language.

```
class User
{
    private string name;
    public string Name
    {
        get { return name; }
        set { name = value; }
    }
}
```

If you observe the above example, we defined a property called "**Name**" and we used a `get` accessor to return a property value and `set` accessors to set a new value. Here, the **value** keyword (/tutorial/csharp/csharp-keywords-reserved-contextual) in `set` accessor is used to define a value that is being assigned by `set` accessor.

In c#, the `get` accessor needs to be used only to return the field value or to compute it and return it, but we should not use it for changing the state of an object.

As discussed, we can extend the behavior of class variables (/tutorial/csharp/csharp-variables-with-examples) using properties `get` and `set` accessors. Following is the example of extending the behavior of **private** variable (/tutorial/csharp/csharp-variables-with-examples) in property using `get` and `set` accessors in c# programming language.

```
class User
{
    private string name = "Suresh Dasari";
    public string Name
    {
      get
      {
          return name.ToUpper();
      }
      set
      {
          if (value == "Suresh")
             name = value;
      }
    }
}
```

If you observe the above example, we are extending the behavior of **private** variable (/tutorial/csharp/csharp-variables-with-examples) **name** using a property called **Name** with `get` and `set` accessors by performing some validations like making sure the **Name** value equals to only "**Suresh**" using `set` accessor and converting property text to uppercase with get accessor.

Here the field "**name**" is marked as **private,** so if you want to make any changes to this field, we can do it only by calling the property (**Name**).

In c# properties, the `get` accessor will be invoked while reading the value of a property, and when we assign a new value to the property, then the `set` accessor will be invoked by using an argument that provides the new value.

Following is an example of invoking `get` and `set` accessors of properties in c# programming language.

```
User u = new User();
u.Name = "Rohini"; // set accessor will invoke
Console.WriteLine(u.Name); // get accessor will invoke
```

In the above example, when we assign a new value to the property, then the `set` accessor will be invoked and the `get` accessor will be invoked when we try to read the value from the property.

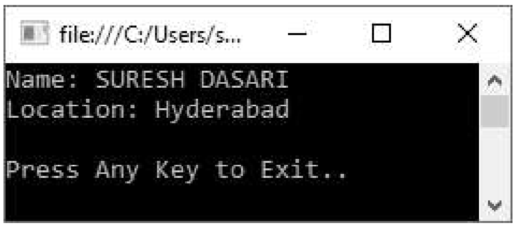
C# Properties (Get, Set) Example

Following is an example of defining **properties** with `get` and `set` accessors to implement required validations without affecting the external way of using it in the c# programming language.

```
using System;

namespace Tutlane
{
     class User
     {
        private string location;
        private string name = "Suresh Dasari";
        public string Location
        {
            get { return location; }
            set { location = value; }
        }
        public string Name
        {
           get
           {
               return name.ToUpper();
           }
           set
           {
               if (value == "Suresh")
                  name = value;
           }
        }
     }
     class Program
     {
        static void Main(string[] args)
        {
            User u = new User();
            // set accessor will invoke
            u.Name = "Rohini";
            // set accessor will invoke
            u.Location = "Hyderabad";
            // get accessor will invoke
            Console.WriteLine("Name: " + u.Name);
            // get accessor will invoke
            Console.WriteLine("Location: " + u.Location);
            Console.WriteLine("\nPress Enter Key to Exit..");
            Console.ReadLine();
        }
    }
}
```

If you observe the above example, we are extending the behavior of private variables (/tutorial/csharp/csharp-variables-with-examples) (**name**, **location**) using properties (**Name**, **Location**) with `get` and `set` accessors by performing some validations like making sure **Name** value equals to only "**Suresh**" using `set` accessor and converting property text to uppercase with `get` accessor.

When you execute the above c# program, we will get the result below.



```
Name: SURESH DASARI
Location: Hyderabad

Press Any Key to Exit..
```

If you observe the above example, our variable text converted to upper case, and even after we set the variable text as "**Rohini**", it displayed text as "**Suresh Dasari**" because of the `set` accessor validation fails in the property.

C# Create Read-Only Properties

As discussed, if a property contains the only `get` accessor, then we will call it a read-only (/tutorial/csharp/csharp-readonly-property) property. Following is the example of creating read-only (/tutorial/csharp/csharp-readonly-property) properties in the c# programming language.

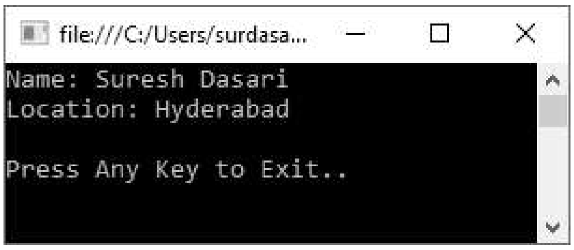
```
using System;

namespace Tutlane
{
    class User
    {
       private string name;
       private string location;
       public User(string a, string b)
       {
          name = a;
          location = b;
       }
       public string Name
       {
          get
          {
             return name;
          }
       }
       public string Location
       {
          get
```

```
            {
               return location;
            }
         }
      }
      class Program
      {
         static void Main(string[] args)
         {
             User u = new User("Suresh Dasari", "Hyderabad");
             // compile error
             // u.Name = "Rohini";
             // get accessor will invoke
             Console.WriteLine("Name: " + u.Name);
             // get accessor will invoke
             Console.WriteLine("Location: " + u.Location);
             Console.WriteLine("\nPress Enter Key to Exit..");
             Console.ReadLine();
         }
      }
}
```

If you observe the above example, we created properties using only `get` accessor to make the properties read-only (/tutorial/csharp/csharp-readonly-property) based on our requirements.

If we uncomment the commented code, we will get a compile error because our **Name** property doesn't contain any `set` accessor to set a new value. It's just read-only (/tutorial/csharp/csharp-readonly-property) property.

When you execute the above c# program, you will get a result as shown below.



```
Name: Suresh Dasari
Location: Hyderabad

Press Any Key to Exit..
```

This is how we can create read-only (/tutorial/csharp/csharp-readonly-property) properties in c# applications based on our requirements.

C# Create Write Only Properties

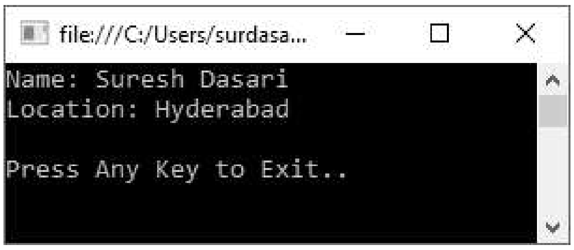
As discussed, if a property contains the only `set` accessor, then we will call it a **write-only** property. Following is the example of creating write-only properties in the c# programming language.

```
using System;

namespace Tutlane
{
     class User
     {
        private string name;
        public string Name
        {
           set
           {
              name = value;
           }
        }
        private string location;
        public string Location
        {
            set
            {
                location = value;
            }
        }
        public void GetUserDetails()
        {
           Console.WriteLine("Name: " + name);
           Console.WriteLine("Location: " + location);
        }
     }
     class Program
     {
         static void Main(string[] args)
         {
              User u = new User();
              u.Name = "Suresh Dasari";
              u.Location = "Hyderabad";
              // compile error
              //Console.WriteLine(u.Name);
              u.GetUserDetails();
              Console.WriteLine("\nPress Enter Key to Exit..");
              Console.ReadLine();
         }
     }
}
```

If you observe the above example, we created properties using only `set` accessor to make the properties are **write-only** based on our requirements.

If we uncomment the commented code, then we will get a compile error because our **Name** property doesn't contain any `get` accessor to return a value. It's a just write-only property.

When you execute the above c# program, you will get a result like as shown below.



```
Name: Suresh Dasari
Location: Hyderabad

Press Any Key to Exit..
```

This is how we can create **write-only** properties in c# applications based on our requirements.

C# Auto Implemented Properties

In c#, a property is called an auto-implemented property when it contains accessors (**get**, **set**) without having any logic implementation.

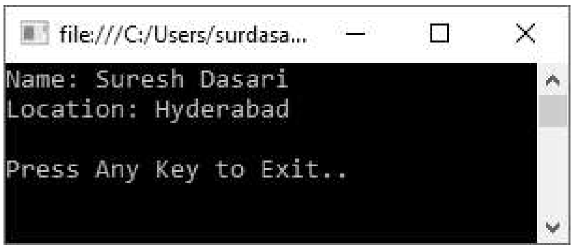
Generally, the auto-implemented properties are useful whenever there is no logic implementation required in property accessors.

Following is the example of creating auto-implemented properties in the c# programming language.

```
using System;

namespace Tutlane
{
     class User
     {
        public string Name { get; set; }
        public string Location { get; set; }
     }
     class Program
     {
         static void Main(string[] args)
         {
             User u = new User();
             u.Name = "Suresh Dasari";
             u.Location = "Hyderabad";
             Console.WriteLine("Name: " + u.Name);
             Console.WriteLine("Location: " + u.Location);
             Console.WriteLine("\nPress Enter Key to Exit..");
             Console.ReadLine();
         }
     }
}
```

If you observe the above example, we created properties with `get` and `set` accessors without having any logic implementation.

When you execute the above c# program, we will get a result like as shown below.



```
Name: Suresh Dasari
Location: Hyderabad

Press Any Key to Exit..
```

This is how we can create **auto-implemented** properties in c# applications based on our requirements.

C# Properties Overview

The following are the important points that we need to remember about properties in the c# programming language.

- In c#, properties will enable class variables to expose in a public way using `get` and `set` accessors by hiding implementation details.
- In properties, a `get` accessor is used to return a property value and a `set` accessor is used to assign a new value.
- The **value** keyword (/tutorial/csharp/csharp-keywords-reserved-contextual) in `set` accessor is used to define a value that is going to be assigned by the `set` accessor.
- In c#, the properties are categorized as read-write, read-only, or write-only.