What is the { get; set; } syntax in C#?

Asked 8 years, 7 months ago Active 2 months ago Viewed 1.1m times

536

161

I am learning ASP.NET MVC and I can read English documents, but I don't really understand what is happening in this code:

```
public class Genre
{
    public string Name { get; set; }
}
```

What does this mean: `{ get; set; }` ?

c# properties

edited May 23 at 9:02

Matthias Braun
16.2k 12 89 125

asked Feb 23 '11 at 20:49

kn3l
8,266 21 69 99

8 possible duplicate of What does this mean ? public Name {get; set;} – nawfal Jun 3 '13 at 19:05

4 In general remember--setters make your object mutable, a bad idea. getters violate "Tell an object what to do, don't ask it for information and manipulate it yourself". So in general, don't add setters and getters by default. You will need them, often, but you should always find a real need before you add them. In particular setters should almost never be used in production code (Strive for immutability wherever possible, and when mutation is needed you should ask it to mutate for you, not set a value). – Bill K Apr 2 '15 at 22:07

8 Just to add something... If you don't put `{get; set;}` you are creating a **Field** but if you put the `{get; set;}` you are creating a **Property**. Having a property could make some things easier especially when working with Reflection. – Seichi May 4 '15 at 21:40

@Seichi using a get-setter creates a Field too, but this one is hidden, declared as private and modified by the auto created properties; all of that made by the compiler. – Jonathan Ramos May 7 '17 at 23:12

1 aren't auto properties defeat the purpose of *private* fields? – mireazma Jan 22 '18 at 9:28

16 Answers

532

It's a so-called auto property, and is essentially a shorthand for the following (similar code will be generated by the compiler):

```
private string name;
public string Name
{
    get
    {
        return this.name;
    }
    set
    {
        this.name = value;
    }
}
```

answered Feb 23 '11 at 20:53

Klaus Byskov Pedersen
89.4k 23 165 212

87 Klaus, can you explain what will happen with this code? It could benefit from a more thorough explanation. – TylerH Nov 17 '14 at 18:53

3 So, just to be sure: it is like if I overloaded the `=` operator, but only for one particular element, right? – Hi-Angel Jan 12 '15 at 9:01

8 Why do we need the private var. :-/ shame. – Oliver Dixon May 29 '16 at 11:57

2 @TylerH The reason for the private variable is encapsulation, the get/set provides a "gate" to get or set the variable. Although there are many reasons not to use get/setters because the "gate" can break the encapsulation of the private variable. (it shouldn't be accessible) – Alexander Mar 9 '17 at 8:47

2 @mpag & dtgq 'value' is an implicit argument, see MS Using Properties. Simple, not poor design choice. Typechecking is preserved – stefano Sep 23 '18 at 17:00

410

So as I understand it `{ get; set; }` is an "auto property" which just like @Klaus and @Brandon said is shorthand for writing a property with a "backing field." So in this case:

```
    public string Name  // this is your property
    {
        get => name;
        set => name = value;
    }
}
```

However if you're like me - about an hour or so ago - you don't really understand what ***properties*** and ***accessors*** are, and you don't have the best understanding of some basic terminologies either. MSDN is a great tool for learning stuff like this but it's not always easy to understand for beginners. So I'm gonna try to explain this more in-depth here.

`get` and `set` are ***accessors***, meaning they're able to access data and info in **private** fields (usually from a *backing field*) and usually do so from **public *properties*** (as you can see in the above example).

There's no denying that the above statement is pretty confusing, so let's go into some examples. Let's say this code is referring to genres of music. So within the class Genre, we're going to want different genres of music. Let's say we want to have 3 genres: Hip Hop, Rock, and Country. To do this we would use the name of the **Class** to create new **instances** of that class.

```
Genre g1 = new Genre(); //Here we're creating a new instance of the class "Genre"
                        //called g1. We'll create as many as we need (3)
Genre g2 = new Genre();
Genre g3 = new Genre();

//Note the () following new Genre. I believe that's essential since we're creating a
//new instance of a class (Like I said, I'm a beginner so I can't tell you exactly why
//it's there but I do know it's essential)
```

Now that we've created the instances of the Genre class we can set the genre names using the 'Name' ***property*** that was set way up above.

```
public string Name //Again, this is the 'Name' property
{ get; set; } //And this is the shorthand version the process we're doing right now
```

We can set the name of 'g1' to Hip Hop by writing the following

```
g1.Name = "Hip Hop";
```

What's happening here is sort of complex. Like I said before, `get` and `set` access information from private fields that you otherwise

`set` uses an implicit variable called `value`. Basically what this means is any time you see "value" within `set`, it's referring to a variable; the "value" variable. When we write `g1.Name =` we're using the `=` to pass in the `value` variable which in this case is `"Hip Hop"`. So you can essentially think of it like this:

```
public class g1 //We've created an instance of the Genre Class called "g1"
{
    private string name;
    public string Name
    {
        get => name;
        set => name = "Hip Hop"; //instead of 'value', "Hip Hop" is written because
                                 //'value' in 'g1' was set to "Hip Hop" by previously
                                 //writing 'g1.Name = "Hip Hop"'
    }
}
```

It's Important to note that the above example isn't actually written in the code. It's more of a hypothetical code that represents what's going on in the background.

So now that we've **set** the Name of the g1 instance of *Genre*, I believe we can **get** the name by writing

```
console.WriteLine (g1.Name); //This uses the 'get' function from our 'Name' Property
                             //and returns the field 'name' which we just set to
                             //"Hip Hop"
```

and if we ran this we would get `"Hip Hop"` in our console.

So for the purpose of this explanation I'll complete the example with outputs as well

```
using System;
public class Genre
{
    public string Name { get; set; }
}

public class MainClass
{
    public static void Main()
    {
```

```
            g1.Name = "Hip Hop";
            g2.Name = "Rock";
            g3.Name = "Country";

            Console.WriteLine ("Genres: {0}, {1}, {2}", g1.Name, g2.Name, g3.Name);
        }
    }
```

Output:

```
"Genres: Hip Hop, Rock, Country"
```

edited Jun 30 '18 at 0:30
AustinWBryan
2,416 2 15 32

answered Apr 2 '15 at 21:42
Josie Thompson
4,310 1 8 21

17 Personally i would just comment it out as such `set{name = value;} // 'value' here is equal to "Hip Hop"` – maksymiuk Sep 4 '15 at 17:18

2 @iLoveUnicorns, it's there for the purpose of data abstraction. The backing field is what contains the actual data. The property definition actually defines how the data is accessed with the `get` and `set` methods. The link I provided has an excellent quote from John Guttag at the top of the page. I would recommend reading his book or even take this free online course – Josie Thompson Jun 3 '16 at 1:34

1 god bless you for clearing my day-long confusion. The first answer is hardly an answer (and has more votes) – LearnByReading Jun 15 '16 at 18:43

2 Can't we just use: `public class Genre{public string Name;}` instead of : `public class Genre{ public string Name { get; set; }}` . I mean, Why do we even need { get; set; }? – user2048204 Nov 7 '16 at 17:24

1 Seems like my concern has already been echoed. If you declare this way: "public string Name { get; set; }" and you access this way: g1.Name = "Hip Hop"; - then where is the Object Orientation? I don't even need the so-called "backing field". The backing field doesn't even exist, as far as I am concerned. Because I only access the public field. And if the public field is "public" then it is not OO compliant. Let's all go back to COBOL. – Baruch Atta Apr 29 at 12:21

96

Those are automatic properties

Basically another way of writing a property with a backing field.

```
    public string Name
    {
      get => _name;

      set => _name = value;
    }
}
```

edited Jun 30 '18 at 0:31
AustinWBryan
2,416 2 15 32

answered Feb 23 '11 at 20:51
Brandon
59.4k 28 179 212

7 What is called " backing field"? – kn3l Feb 23 '11 at 20:54

1 Succinct - a demonstration of use may go down well, too. – Grant Thomas Feb 23 '11 at 20:55

4 @stackunderflow: The backing field is where the data is stored. (what is returned when using `get` , and persisted using `set`). Like the cupboard to which `get` and `set` opens the door of. – Grant Thomas Feb 23 '11 at 20:57

5 @stackunderflow: In this answer, the backing field is `_name` . In the automatic property, the backing field is hidden. – Justin Feb 23 '11 at 21:00

36

This is the short way of doing this:

```
public class Genre
{
    private string _name;

    public string Name
    {
      get => _name;
      set => _name = value;
    }
}
```

edited Jun 30 '18 at 0:31
AustinWBryan
2,416 2 15 32

answered Feb 23 '11 at 20:54
froeschli
2,109 2 20 46

private data member for you.

30

You could just make your data members public without using this shortcut but then if you decided to change the implementation of the data member to have some logic then you would need to break the interface. So in short it is a shortcut to create more flexible code.

answered Feb 23 '11 at 20:55

Kelsey

41.6k 14 111 154

1 Kelsey - could you explain how this syntax makes it any more "flexible" code? I don't see it. If you would add any "logic" to the setter or getter, then in ether case (with or without private data) you would still break the interface, as it is, and need some coding. – Baruch Atta Apr 29 at 12:33

this is the best answer, simple and to the point. – shady sherif Jun 30 at 9:58

15

Basically, it's a shortcut of:

```
class Genre{
    private string genre;
    public string getGenre() {
        return this.genre;
    }
    public void setGenre(string theGenre) {
        this.genre = theGenre;
    }
}
//In Main method
genre g1 = new Genre();
g1.setGenre("Female");
g1.getGenre(); //Female
```

edited Dec 18 '18 at 17:11

learnAsWeGo

2,046 2 3 18

answered Sep 12 '15 at 1:07

Jirson Tavera

893 10 15

5 This doesn't answer the question. The OP was talking about properties. – theB Sep 12 '15 at 1:16

10

Its an auto-implemented property for C#.

answered Feb 23 '11 at 20:50

Daniel A. White
155k 40 306 384

1 Eh... Does this mean that you keep nil reference to the string and then load it's value from a standard location when `get; set;` is called? – Aurum Aquila Feb 23 '11 at 20:52

1 Yes it keeps `null` like any `string` variable until `someInstanceOfGenere.Name = "someValue"` – Daniel A. White Feb 23 '11 at 20:56

6

They are the accessors for the public property Name.

You would use them to get/set the value of that property in an instance of Genre.

answered Feb 23 '11 at 20:51

TimCodes.NET
3,706 20 35

6

That is an Auto-Implemented Property. It's basically a shorthand way of creating properties for a class in C#, without having to define private variables for them. They are normally used when no extra logic is required when getting or setting the value of a variable.

You can read more on MSDN's Auto-Implemented Properties Programming Guide.

answered Feb 23 '11 at 20:52

Dusda
2,279 5 32 56

This mean that if you create a variable of type Genre, you will be able to access the variable as a property

```
oG.Name = "Test";
```

edited Feb 23 '11 at 21:02

abatishchev
72.4k 70 270 406

answered Feb 23 '11 at 20:51

David Brunelle
4,098 9 54 98

5 When you don't use auto-implemented properties, you still able to access it in this manner. i.e. AIP is not about access from outside, but about declaration inside a class. – abatishchev Feb 23 '11 at 21:02

5

- The get/set pattern provides a structure that allows logic to be added during the setting ('set') or retrieval ('get') of a property instance of an instantiated class, which can be useful when some instantiation logic is required for the property.
- A property can have a 'get' accessor only, which is done in order to make that property read-only
- When implementing a get/set pattern, an intermediate variable is used as a container into which a value can be placed and a value extracted. The intermediate variable is usually prefixed with an underscore. this intermediate variable is private in order to ensure that it can only be accessed via its get/set calls. See the answer from Brandon, as his answer demonstrates the most commonly used syntax conventions for implementing get/set.

answered Jun 15 '17 at 1:22

Chris Halcrow
13.1k 6 86 104

4

Such `{ get; set; }` syntax is called automatic properties, C# 3.0 syntax

You must use Visual C# 2008 / csc v3.5 or above to compile. But you can compile output that targets as low as .NET Framework 2.0 (no runtime or classes required to support this feature).

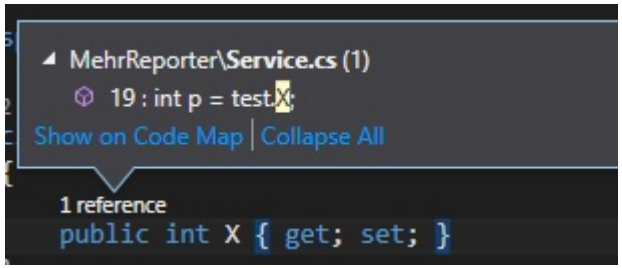
answered Feb 28 '13 at 5:39

linquize
15.3k 8 50 74

4

By adding a `{ get; set; }` to `X` property, you will not get this warning.

In addition in Visual Studio 2013 and upper versions, by adding `{ get; set; }` you are able to see all references to that property.



edited Aug 29 '16 at 11:22

answered Aug 29 '16 at 11:09

Omid Ebrahimi
591 1 12 35

2

Get set are access modifiers to property. Get reads the property field. Set sets the property value. Get is like Read-only access. Set is like Write-only access. To use the property as read write both get and set must be used.

answered Nov 25 '14 at 21:34

community wiki
Ashraf Abusada

1 i think get set are not access modifiers, infact they are accessors. Access modifiers are like : public, private, internal etc. – Rohit Arora Nov 17 '15 at 9:45

2

Its basically a shorthand. You can write `public string Name { get; set; }` like in many examples, but you can also write it:

```
private string _name;

public string Name
{
    get { return _name; }
    set { _name = value ; } // value is a special keyword here
```

Let me give you an example:

```
private class Person {
    private int _age;  // Person._age = 25; will throw an error
    public int Age{
        get { return _age; }  // example: Console.WriteLine(Person.Age);
        set {
            if ( value >= 0) {
                _age = value; }  // valid example: Person.Age = 25;
        }
    }
}
```

Officially its called Auto-Implemented Properties and its good habit to read the (programming guide). I would also recommend tutorial video C# Properties: Why use "get" and "set".

edited Jul 11 at 18:18

toobladink
54 5

answered Dec 12 '18 at 14:30

Randel
304 2 8

1

Get is invoked when the property appears on the right-hand side (RHS) Set is invoked when the property appears on the left-hand side (LHS) of '=' symbol

For an auto-implemented property, the backing field works behind the scene and not visible.

Example:

```
public string Log { get; set; }
```

Whereas for a non auto-implemented property the backing field is upfront, visible as a private scoped variable.

Example:

```
private string log;
```

```
    set => log = value;
}
```

Also, it is worth noted here is the 'getter' and 'setter' can use the different 'backing field'

edited Jun 30 '18 at 0:32

AustinWBryan
2,416 2 15 32

answered May 24 '17 at 21:47

Bala
37 3

This does not seem to answer the question asked. – TylerH May 24 '17 at 22:37

Provided hint on when the get & set is invoked. All the answers mentioned above, makes an impression that the backing field for get & set is the same. But it is not the case. So my answer is very much relevant to the main question. Hope you agree with me. – Bala May 30 '17 at 16:58

For an auto-generated property, which is what the question asks about, there can't be different backing fields used for the getter and the setter; there is only one backing field. For a non-auto property (which the question doesn't ask about) there may not even conceptually be a backing field at all. Additionally, you can write a program with a getter on the left hand side of an assignment operator and one with a setter on the right hand side of an assignment operator. So not only is all of this information not answering the question asked, but it's all also wrong. – Servy May 30 '17 at 17:00

protected by Community ♦ Jul 12 '16 at 12:54

Thank you for your interest in this question. Because it has attracted low-quality or spam answers that had to be removed, posting an answer now requires 10 reputation on this site (the association bonus does not count).

Would you like to answer one of these unanswered questions instead?