How can I access the backing variable of an auto-implemented property?

Asked 16 years, 3 months ago Modified 6 months ago Viewed 266k times

78

In the past we declared properties like this:

```
public class MyClass
{
    private int _age;

    public int Age
    {
          get{ return _age;  }
          set{ _age = value; }
    }
}
```

Now we can do:

```
public class MyClass
{
    public int Age {get; set;}
}
```

My question is, how can I access the private variable that is created automatically using this notation?

I would rather access the private variable and not the public accessor 'Age'. Is there a default notation to access the private variable, or it is just not possible?

c# properties

Share

Improve this question

Follow

edited Aug 26, 2014 at 16:24

Jeroen Vannevel

44.4k • 24 • 111 • 174

asked Sep 14, 2008 at 17:19

public static

13k • 27 • 67 • 87

10 What is the difference in this case of accessing the private vs the public accessor? I think it is best practice to access the public accessor even from the logic in the declaring class. If you ever add some logic to the accessor you do not want to have to change all your code. – Eric Schoonover Sep 14, 2008 at 20:52

@spoon16 Can you give me an example of adding logic to the accessor and having to change all your code as a result. I didn't really understand this part. – Ogen Dec 21, 2014 at

3:17

7 Answers

Sorted by: Highest score (default)

92

The aim of the new automatic properties is to reduce the amount of boilerplate code you need to write when you just have a simple property that doesn't need any special logic in the get or the set.

If you want to access the private member that these properties use, that's usually for a few reasons:

- You need to more than just a simple get/set - in this case, you should just avoid using automatic properties for this member.
- You want to avoid the performance hit of going through the get or set and just use the member directly - in this case, I'd be surprised if there really was a performance hit. The simple get/set members are very very easy to inline, and in my (admittedly limited) testing I haven't found a difference between using the automatic properties and accessing the member directly.
- You only want to have public read access (i.e. just a 'get') and the class write to the member directly - in this case, you can use a private set in your automatic property. i.e.

```
public class MyClass
{
    public int Age {get; private set;}
}
```

This usually covers most the reasons for wanting to directly get to the backing field used by the automatic properties.

Share

Improve this answer

Follow

edited Feb 19, 2014 at 19:14

Ascendant

2,569 ● 3 ● 28 ● 35

answered Sep 14, 2008 at 17:47

Wilka

29.5k ● 15 ● 78 ● 98

True - they do reduce bowlerplate code and I will also ad that they reduce what you need to test. Some might argue that you need to test manual get/set but not with auto properties since you can trust the framework. – Daniel Auger Sep 14, 2008 at 18:07

3 The code sample is wrong here. It should be public class MyClass { public int Age {get; private set;} } I agree with this answer. You can't access the private field and if you need to, then you shouldn't be using Automatic Properties in the first place. – hwiechers Sep 14, 2008 at 18:21

hwiechers, thanks for that - I had that edit, but when trying to fix the formating I must have hit paste again and deleted the wrong block of code. Doh! – Wilka Sep 14, 2008 at 19:18

Commonly used in POCO objects – RobS Dec 9, 2008 at 22:33

The reason I ended up having to access the private field was because it was a struct and you can't modify a struct if it's not a variable (you'd need to create a new copy of it instead) - a private field provides a variable whereas a property (auto or not) does not. – Bilal Akil Mar 23, 2022 at 5:36

23

Your usage of automatic properties implies that you do not need any getting/setting logic for the property thus a private backing variable is unneccessary.

Don't use automatic properties if you have any complex logic in your class. Just go `private int _age` and normal getters/setters as you normally would.

IMO, automatic properties are more suited for quickly implementing throwaway objects or temporary data capsules like:

```
public class TempMessage {
    public int FromID { get; set; }
    public int ToID { get; set; }
    public string Message { get; set; }
}
```

Where you don't need much logic.

Share Improve this answer Follow

answered Sep 14, 2008 at 17:35

chakrit
61.5k ● 25 ● 136 ● 163

Why would adding complex logic to your class affect whether or not you used automatic properties within that class? – Eric Schoonover Sep 14, 2008 at 20:46

More of a consistency thing... if you have complex logic then you'll want to access it from the private backing variable per OO practices.. and if you use automatic properties then... there'll be inconsistencies in accessing those properties. since some will have underscore prefix and some will not. – chakrit Sep 18, 2008 at 22:31

12

This syntax is commonly called "syntax sugar", which means that the compiler takes that syntax and translates it into something else. In your example, the compiler would generate code that looks something like this:

```
[CompilerGenerated]
private int <Age>k_BackingField;

public int Age
{
   [CompilerGenerated]
```

```
get
{
   return this.<Age>k_BackingField;
}
[CompilerGenerated]
set
{
   this.<Age>k_BackingField = value;
}
```

Even knowing all of that, you could **probably** access the backing field directly but that sort of defeats the purpose of using automatic properties. I say probably here because you then depend on an implementation detail that could change at any point in a future release of the C# compiler.

Share Improve this answer Follow

answered Sep 14, 2008 at 17:34

Scott Dorman

42.5k ● 12 ● 81 ● 112

10

> Behind the scenes what happens is the injection of a private member variable, prefixed with <>k__AutomaticallyGeneratedPropertyField#

From C# 3.0 Automatic Properties explained

Although it may be possible to use that private member directly, it's very hacky and unnecessary.

Share Improve this answer Follow

answered Sep 14, 2008 at 17:27

macbirdie

16.2k ● 6 ● 49 ● 54

7

You shouldn't, and it's very unlikely you need to. If you need to access the property, just use the public property (e.g. this.Age). There's nothing special about the private field backing the public property, using it in preference to the property is just superstition.

Share Improve this answer Follow

answered Sep 14, 2008 at 20:39

Wedge

19.8k ● 7 ● 50 ● 71

1 I always wondered what the point of automatic properties were. If you have no special logic in your getters and setters why have them at all? – Matthew Lock Jan 22, 2013 at 1:35

4 @MatthewLock flexibility. With direct field access you are locked into that design unless you change all the client code at the same time. But with a property if you decide to change it from a "pseudo-field" into a computed property, or if you decide to add assertions and checks on the setter, or what-have-you, you can do so transparently. This is even more useful if you are writing library code, where you can't control all of the client code. – Wedge Jan 24, 2013 at 18:10

2

You can't, it's a language feature as opposed to a IDE feature. To be honest i'd prefer then IDE to add the private variable in for you. I agree that it is slightly weird for the class to internally have to use the public entry point to access its own variables. Hence I don't use this new feature that much myself.

Share Improve this answer Follow

answered Sep 14, 2008 at 17:22

Quibblesome
25.4k • 10 • 62 • 104

-1

Use C# version 13 or later :) and it will be possible.

Share Improve this answer Follow

answered May 30 at 12:06

David V. Corbin
397 • 1 • 12