C# 3.0 auto-properties — useful or not? [closed]

Asked 16 years, 4 months ago Modified 6 years ago Viewed 51k times

158

Closed. This question is opinion-based. It is not currently accepting answers.

Want to improve this question? Update the question so it can be answered with facts and citations by editing this post.

Closed 10 years ago.

Improve this question

Note: This was posted when I was starting out C#. With 2014 knowledge, I can truly say that auto-properties are among the best things that ever happened to the C# language.

I am used to create my properties in C# using a private and a public field:

```
private string title;
public string Title
{
    get { return title;  }
    set { title = value;  }
}
```

Now, with .NET 3.0, we got auto-properties:

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

I know this is more a philosophical/subjective questions, but is there any reason to use these auto-properties except from saving five lines of code for each field? My personal gripe is that those properties are hiding stuff from me, and I am not a big fan of black magic.

In fact, the hidden private field does not even show up in the debugger, which is OK given the fact that the get/set functions do nothing. But when I want to actually implement some getter/setter logic, I have to use the private/public pair anyway.

I see the benefit that I save a lot of code (one vs six lines) without losing the ability to change the getter/setter logic later, but then again I can already do that by simply declaring a public field "Public string Title" without the need of the { get; set; } block, thus even saving more code.

So, what am I missing here? Why would anyone actually want to use auto-properties?

c# .net automatic-properties

Share
Improve this question
Follow

edited Oct 24, 2018 at 13:49
Uwe Keim
40.7k ● 61 ● 185 ● 301

asked Aug 12, 2008 at 23:06
Michael Stum
181k ● 119 ● 407 ● 541

90 "My personal gripe is that those properties are hiding stuff from me, and I am not a big fan of black magic." Huh? You ARE aware that the compiler hides a ton from you all the time, right? Unless you are writing assembly (or more accurately, the actual 1's and 0's for your code), EVERYTHING you write is hiding stuff from you. – Charles Boyung Jun 10, 2010 at 15:37

17 Answers

Sorted by: Highest score (default)

122

We use them all the time in Stack Overflow.

You may also be interested in a discussion of Properties vs. Public Variables. IMHO that's really what this is a reaction to, and for that purpose, it's great.

Share
Improve this answer
Follow

edited Apr 7, 2014 at 18:32
yoozer8
7,489 ● 7 ● 62 ● 96

answered Aug 12, 2008 at 23:13
Jeff Atwood
63.9k ● 48 ● 151 ● 153

64

Yes, it does *just* save code. It's miles easier to read when you have loads of them. They're quicker to write and easier to maintain. Saving code is always a good goal.

You can set different scopes:

```
public string PropertyName { get; private set; }
```

So that the property can only be changed inside the class. This isn't really immutable as you can still access the private setter through reflection.

As of C#6 you can also create true `readonly` properties - i.e. immutable properties that cannot be changed outside of the constructor:

```
public string PropertyName { get; }

public MyClass() { this.PropertyName = "whatever"; }
```

At compile time that will become:

```
readonly string pName;
public string PropertyName { get { return this.pName; } }

public MyClass() { this.pName = "whatever"; }
```

In immutable classes with a lot of members this saves a lot of excess code.

Share
Improve this answer
Follow

edited Oct 7, 2015 at 8:17

answered Aug 13, 2008 at 7:09
Keith
155k ● 82 ● 306 ● 446

"So you don't lose any functionality." how do you debug them? – wal Oct 25, 2010 at 3:43

2 @wal - what's there to debug? From that point of view you're basically dealing with member variables. – Keith Oct 26, 2010 at 11:17

6 @wal - You can put a breakpoint on them, just like you can access of a member variable, you just can't step into them. But why would you want to? What the auto-properties actually do is both trivial and auto-generated, if you've got bugs that's one place they're extremely unlikely to be. – Keith Oct 27, 2010 at 14:45

3 we may need to take this outside Keith. :) – wal Oct 28, 2010 at 13:10

1 But ok, assume you have many setter calls to myObj.Title...you want to see where the value changes from "text" to null, ie a conditional breakpoint. how do u acheive that? you cant even set a breakpoint on the setter – wal Oct 28, 2010 at 13:12

47

The three big downsides to using fields instead of properties are:

1. You can't databind to a field whereas you can to a property
2. If you start off using a field, you can't later (easily) change them to a property
3. There are some attributes that you can add to a property that you can't add to a field

Share
Improve this answer
Follow

edited Jan 20, 2010 at 9:16
Dimitri C.
22.4k ● 21 ● 88 ● 103

answered Aug 12, 2008 at 23:13
lomaxx
116k ● 58 ● 147 ● 180

7 "If you start off using a field, you can't later (easily) change them to a property", sorry but why ? – Homam Oct 11, 2010 at 13:26

6 @Homam Mainly, any consumer code that uses reflection on your fields would break, since they would have to switch from using FieldInfo to PropertyInfo. – Isabelle Wedin Oct 21, 2010 at 16:10

9 @Homam Also, changing a field to a property breaks binary compatability, requiring all consumers of the field to recompile. – Odrade Jan 20, 2011 at 18:20

1 recompilation & reflection issues aside, it's very easy to encapsulate fields using Visual Studio: Ctrl-R+E will allow you to turn a field into a property with appropriate getters/setters. (or right-click the field, re-factor, encapsulate field). – JoeBrockhaus Nov 7, 2012 at 18:30

@Hommam fields are lvalues (they're variables), and properties aren't. Stuff that might have compiled when it was a field might not when it's a property. – Mark May 26, 2015 at 21:07

30

I personally love auto-properties. What's wrong with saving the lines of code? If you want to do stuff in getters or setters, there's no problem to convert them to normal properties later on.

As you said you could use fields, and if you wanted to add logic to them later you'd convert them to properties. But this might present problems with any use of reflection (and possibly elsewhere?).

Also the properties allow you to set different access levels for the getter and setter which you can't do with a field.

I guess it's the same as the var keyword. A matter of personal preference.

Share
Improve this answer
Follow

edited Dec 12, 2018 at 8:34
Sнаđошƒаӽ
17.5k ● 13 ● 76 ● 93

answered Aug 12, 2008 at 23:13
Ray
46.5k ● 29 ● 127 ● 170

30

From Bjarne Stroustrup, creator of C++:

> I particularly dislike classes with a lot of get and set functions. That is often an indication that it shouldn't have been a class in the first place. It's just a data structure. And if it really is a data structure, make it a data structure.

And you know what? He's right. How often are you simply wrapping private fields in a get and set, without actually doing anything within the get/set, simply because it's the "object oriented" thing to do. This is Microsoft's solution to the problem; they're basically public fields that you can bind to.

Share
Improve this answer

edited Dec 12, 2018 at 8:35

answered Oct 8, 2008 at 11:21

Follow

Sнаđошƒаӽ
17.5k ● 13 ● 76 ● 93

Giovanni Galbo
13.1k ● 13 ● 61 ● 79

2 I really think this should have more points. Far too many people see auto properties as a green-light for writing horribly encapsulated (or not encapsulated at all) classes that are little more than a glorified public field. Of course this is more of an issue with how people use the tool rather than with the tool itself, but I think it's important to mention when discussing properties in general. – sara Sep 3, 2015 at 14:56

18

One thing nobody seems to have mentioned is how auto-properties are unfortunately not useful for immutable objects (usually immutable structs). Because for that you really should do:

```
private readonly string title;
public string Title
{
    get { return this.title; }
}
```

(where the field is initialized in the constructor via a passed parameter, and then is read only.)

So this has advantages over a simple `get` / `private set` autoproperty.

Share Improve this answer Follow

answered Aug 27, 2008 at 18:04

Domenic
113k ● 42 ● 226 ● 273

If you have a struct changing any property on it results in a new struct. This would only be an issue if you wanted an internally immutable reference type - I can't see a reason why you ever would need one. – Keith Feb 13, 2009 at 12:55

@Keith: Your first sentence seems factually incorrect. – Domenic Feb 18, 2009 at 21:27

3 Wouldn't `public string Title { get; private set; }` kind of result in exactly the same thing? You would be able to change it from inside the class then of course, but if you do that you have other problems... :p – Svish Aug 31, 2009 at 8:14

2 @Svish - by that argument the readonly keyword in C# should never be used because its use would mean that we were hiding these "different problems" – Zaid Masud Sep 12, 2011 at 15:55

2 I just mean that from an outside API kind of view, it wouldn't make much difference. And so, auto-properties could be used if wanted. Best thing would of course be if you could do something like `public string Title { get; private readonly set; }` – Svish Sep 13, 2011 at 11:01

12

I always create properties instead of public fields because you can use properties in an interface definition, you can't use public fields in an interface definition.

Share Improve this answer Follow

answered Dec 5, 2008 at 10:59
Theo
452 ●3 ●3

8

Auto-properties are as much a black magic as anything else in C#. Once you think about it in terms of compiling down to IL rather than it being expanded to a normal C# property first it's a lot less black magic than a lot of other language constructs.

Share Improve this answer Follow

answered Aug 18, 2008 at 0:27
ICR
14.2k ●4 ●51 ●78

5

I use auto-properties all the time. Before C#3 I couldn't be bothered with all the typing and just used public variables instead.

The only thing I miss is being able to do this:

```
public string Name = "DefaultName";
```

You have to shift the defaults into your constructors with properties. tedious :-(

Share Improve this answer Follow

answered Aug 12, 2008 at 23:22
Orion Edwards
123k ●66 ●245 ●339

5 With Auto-property initializers from C# 6 you will soon be able to do this: `public string Name { get; set; } = "DefaultName";` blogs.msdn.com/b/csharpfaq/archive/2014/11/20/… – Carlos Muñoz Mar 26, 2015 at 17:01

5

I think any construct that is intuitive AND reduces the lines of code is a big plus.

Those kinds of features are what makes languages like Ruby so powerful (that and dynamic features, which also help reduce excess code).

Ruby has had this all along as:

```
attr_accessor :my_property
attr_reader :my_getter
attr_writer :my_setter
```

Share Improve this answer Follow

answered Aug 13, 2008 at 0:14
Mike Stone
44.6k ● 30 ● 114 ● 140

2

The only problem I have with them is that they don't go far enough. The same release of the compiler that added automatic properties, added partial methods. Why they didnt put the two together is beyond me. A simple "partial On<PropertyName>Changed" would have made these things really really useful.

Share Improve this answer Follow

answered Aug 13, 2008 at 1:36
David Wengier
10.2k ● 5 ● 40 ● 43

You can put multiple partial methods inside of another method. Creating an auto-pattern of some sort for them would be confusing. – Matthew Whited Sep 12, 2010 at 12:23

2

It's simple, it's short and if you want to create a real implementation inside the property's body somewhere down the line, it won't break your type's external interface.

As simple as that.

Share Improve this answer Follow

answered Sep 7, 2008 at 8:06
Omer van Kloeten
12k ● 9 ● 44 ● 54

1

One thing to note here is that, to my understanding, this is *just* syntactic sugar on the C# 3.0 end, meaning that the IL generated by the compiler is the same. I agree about avoiding black magic, but all the same, fewer lines for the same thing is usually a good thing.

Share Improve this answer Follow

answered Aug 17, 2008 at 22:40
pbh101
10.4k ● 9 ● 33 ● 31

1

In my opinion, you should always use auto-properties instead of public fields. That said, here's a compromise:

Start off with an internal field using the naming convention you'd use for a property. When you first either

- need access to the field from outside its assembly, or
- need to attach logic to a getter/setter

Do this:

1. rename the field
2. make it private
3. add a public property

Your client code won't need to change.

Someday, though, your system will grow and you'll decompose it into separate assemblies and multiple solutions. When that happens, any exposed fields will come back to haunt you because, as Jeff mentioned, changing a public field to a public property is a breaking API change.

Share Improve this answer Follow

answered Aug 17, 2008 at 23:19
ESV
7,730 ● 4 ● 40 ● 29

0

I use CodeRush, it's faster than auto-properties.

To do this:

```
 private string title;
public string Title
{
    get { return title;  }
    set { title = value;  }
}
```

Requires eight keystrokes total.

Share Improve this answer Follow

answered Aug 21, 2008 at 11:13
Brian Leahy
35.4k ● 12 ● 46 ● 60

6 If I hold down CTRL and V, I can paste lots and lots of stuff, /really quickly/, but that doesn't make it 'better'. How does this answer the original question? – JBRWilkinson Feb 18, 2010 at 14:59

0

@Domenic : I don't get it.. can't you do this with auto-properties?:

```
public string Title { get; }
```

or

```
public string Title { get; private set; }
```

Is this what you are referring to?

Share Improve this answer Follow

answered Aug 30, 2008 at 11:59

Andrei Rînea
20.7k ● 18 ● 121 ● 169

You can (the latter; the former will not compile), but then the field is not immutable inside your object. – Domenic Sep 19, 2008 at 22:21

Word of caution, only structs are immutable when flagged readonly, classes are just unassignable. – Guvante Oct 8, 2008 at 11:21

0

My biggest gripe with auto-properties is that they are designed to save time but I often find I have to expand them into full blown properties later.

What VS2008 is missing is an **Explode Auto-Property** refactor.

The fact we have an **encapsulate field** refactor makes the way I work quicker to just use public fields.

Share Improve this answer Follow

answered Dec 5, 2008 at 10:24

Johnno Nolan
29.6k ● 19 ● 113 ● 160