How do you give a C# Auto-Property a default value?

1672

179

How do you give a C# Auto-Property a default value? I either use the constructor, or revert to the old syntax.

Using the Constructor:

```
class Person
{
    public Person()
    {
        Name = "Default Name";
    }
    public string Name { get; set; }
}
```

Using normal property syntax (with a default value)

```
private string name = "Default Name";
public string Name
{
    get
    {
        return name;
    }
    set
    {
        name = value;
    }
}
```

Is there a better way?

c# automatic-properties

48 It's misleading, it even says in the IntelliSense, "Sets the default value for a property", but on MSDN it says, "A DefaultValueAttribute will not cause a member to be automatically initialized with the attribute's value. You must set the initial value in your code.", Urg! – Chris Feb 22 '10 at 5:11

20 This is one of the few places where vb.net has the clear upper hand, they can set default values for properties in the definition. – Blindy Aug 9 '10 at 8:35

4 Why do you not want to back the property with an initialized field? This is how the compiler implements automatic properties anyway. Also an explicit field has the advantage that you can declare it `readonly` . – finnw Nov 9 '10 at 12:52

9 @finnw IMO it's about improving code readability. – Keith Nov 17 '11 at 2:26

7 This is the only thing I love VB.NET over C#. `Public Property MyName As String = "Default Name"` – Alex Yeung Dec 3 '12 at 7:21

23 Answers

1986

In C# 5 and earlier, to give auto implemented properties a default value, you have to do it in a constructor.

The ability to have auto property initializers is included since C# 6.0. The syntax is:

```
public int X { get; set; } = x; // C# 6 or higher
```

edited Oct 23 '15 at 6:59
Shishir Gupta
983 2 11 30

answered Sep 2 '08 at 21:46

Darren Kopp
61.8k 9 67 87

103 you are incorrect. abstract classes can have constructors. – Darren Kopp Mar 1 '11 at 18:31

28 abstract classes usually have protected constructors, actually. – Ludovic Chabant Jul 29 '11 at 19:06

51 abstract classes usually have whatever accessibility you give to them. if you don't specify a constructor you always have a default public constructor – Darren Kopp Jul 29 '11 at 23:04

38 Constructors to abstract classes should never be given public access since they can never be instantiated. To do so would be a FxCop violation (hopefully you're using it) and a violation of .NET Framework Design Guidelines - amazon.com/Framework-Design-Guidelines-Conventions-Libraries/dp/ – Dave Black Feb 8 '12 at 14:16

271

Edited on 1/2/15

C# 6 :

With C# 6 you can initialize auto-properties directly (finally!), there are now other answers in the thread that describe that.

C# 5 and below:

Though the intended use of the attribute is not to actually set the values of the properties, you can use reflection to always set them anyway...

```
public class DefaultValuesTest
{
    public DefaultValuesTest()
    {
        foreach (PropertyDescriptor property in TypeDescriptor.GetProperties(this))
        {
            DefaultValueAttribute myAttribute =
(DefaultValueAttribute)property.Attributes[typeof(DefaultValueAttribute)];

            if (myAttribute != null)
            {
                property.SetValue(this, myAttribute.Value);
            }
        }
    }

    public void DoTest()
    {
        var db = DefaultValueBool;
        var ds = DefaultValueString;
        var di = DefaultValueInt;
    }


    [System.ComponentModel.DefaultValue(true)]
    public bool DefaultValueBool { get; set; }

    [System.ComponentModel.DefaultValue("Good")]
    public string DefaultValueString { get; set; }
```

edited Jan 31 at 6:21

Ajay2707
4,508 4 23 45

answered Jun 22 '11 at 18:14

Chuck Rostance
5,259 1 20 16

3 Yes,It's the answer:[System.ComponentModel.DefaultValue(GiveAnyTypeOfDefaultValueHere)] – WAP Guy Feb 10 '12 at 5:44

26 +1 for ingenuity, but this can be really slow. Imagine `new` ing a bunch of items in an inner loop without realizing the constructors use reflection... – Patrick M Nov 1 '12 at 20:04

2 If the base class uses a static constructor to load the attributes and values into a dictionary, there's no overhead on each class! – Gayot Fow Jul 4 '13 at 19:16

25 U're recommending really bad thing. It is quite slow and introduces reflection to quite simple class. If it wasn't so slow it would be okay to use some kind of il-weaving with such attribute (PostSharp, Fody, etc), but the performance... – Grigory Aug 21 '13 at 13:03

1 Only good for the VS designer and even than not in all cases (ie asp.net), anyway.. this will not do anything at runtime. – G.Y Aug 25 '13 at 8:08

149

When you inline an initial value for a variable it will be done implicitly in the constructor anyway.

I would argue that this syntax was best practice in C# up to 5:

```
class Person
{
    public Person()
    {
        //do anything before variable assignment

        //assign initial values
        Name = "Default Name";

        //do anything after variable assignment
    }
    public string Name { get; set; }
}
```

As this gives you clear control of the order values are assigned.

edited Sep 8 '15 at 16:23

answered Sep 4 '08 at 12:16

Keith
97.5k 59 240 358

11 It's only best practice if constructors need to initialize in different ways (across multiple constructors). If they all initialize the same way, isn't what you are suggesting bad practice if I also want Person(string name), and then later I add property int ID and I have yet another constructor Person(string name, int id)? Violation of open to extension, closed to modification. I argue that it's best practice to initialize in line so long as all constructors initialize the member the same way. – dbobrowski Sep 13 '12 at 19:12

2 @dbobrowski if I has multiple constructors I'd use `public Person(string name, int id) : this()` syntax, but you have a point. This gives you more fine control of exactly when properties are initialised at the cost of the inline option's simpler maintainability. – Keith Mar 8 '13 at 16:35

5 @Keith I'd prefer `public Person():this("Default Name")` , rather than setting Name twice. – ANeves Jan 31 '14 at 17:43

1 if you need to run the same code in several places, you don't resort to reflection and weird attributes that no one has ever heard of. you extract a helper method, or in the real complicated places, create a factory (or just redesign/break down your class alltogether since it's probably too complex) – sara Dec 14 '15 at 7:40

83

DefaultValueAttribute ONLY work in the vs designer. It will not initialize the property to that value.

See DefaultValue attribute is not working with my Auto Property

edited May 23 '17 at 11:55

Community ♦
1 1

answered Sep 2 '08 at 22:44

Darren Kopp
61.8k 9 67 87

7 +1: [DefaultValue] is used to tell the designed what the default value is. If, in the properties window, you set the value to this default, then the value is not written to the generated code. Similarly, non-default values are shown in bold in the properties window. Of course, this only works if [DefaultValue] actually has the correct value. It does nothing to enforce this. – Roger Lipscombe May 9 '10 at 19:08

8 `DefaultValueAttribute` is not just for the designer. However, a "default value" is not the same as an "initial value". – Ben Voigt Jun 3 '11 at 2:11

```
    {
        get
        {
            return string.IsNullOrEmpty(_name) ? "Default Name" : _name;
        }

        set { _name = value; }
    }
}
```

Obviously if it's not a string then I might make the object nullable (double?, int?) and check if it's null, return a default, or return the value it's set to.

Then I can make a check in my repository to see if it's my default and not persist, or make a backdoor check in to see the true status of the backing value, before saving.

Hope that helps!

answered Sep 2 '08 at 23:07

crucible
2,122 2 20 33

30 `return _name ?? "Default Name";` probably even is more clear that your – abatishchev Aug 9 '10 at 8:11

15 @abatishchev: though that is not the same. crucibles code would return "Default Name" if the string is "" or null, but using your approach would return "Default Name" only in case it is null. Also, it is discussible whether "??" or "IsNullOrEmpty" is more clear. – Sebastian Mach Dec 16 '10 at 13:28

10 @phresnel: If property value can be "" and it's the same as null then you're right. Otherwise "" is meaningful value and my code is better – abatishchev Dec 17 '10 at 6:51

1 `private string _name = "Default Name"; public string Name { get { return _name; } set { _name = value; } }` – Gilbert May 24 '16 at 13:40

1 @Gilbert - you have repeated one of the code options shown in the question. Crucible's answer is specifically for the case "I don't want the default value persisted in my db". That is, he wants to be able to determine elsewhere whether _name has been written to, or is still null. [His write or persist code checks `_name` for null, skips writing it.] – ToolmakerSteve Jan 28 '18 at 21:31

```
public class Coordinate
{
    public int X { get; set; } = 34; // get or set auto-property with initializer

    public int Y { get; } = 89;      // read-only auto-property with initializer

    public int Z { get; }            // read-only auto-property with no initializer
                                     // so it has to be initialized from constructor

    public Coordinate()              // .ctor()
    {
        Z = 42;
    }
}
```

edited Sep 29 '17 at 16:34 — Graham — 4,053 14 42 63

answered Jul 25 '16 at 19:35 — Shiva — 14.7k 10 65 96

1 I don't have C#6.0 yet, and was checking to see what version I needed for default values on auto-properties. Does C# 6.0 also remove the need to have `{ get; set; }` or `{ get; private set; }` as otherwise setting the value would be blocked by the compiler? – freefaller Apr 20 '18 at 11:20

38

Starting with C# 6.0, We can assign default value to auto-implemented properties.

```
public string Name { get; set; } = "Some Name";
```

We can also create read-only auto implemented property like:

```
public string Name { get; } = "Some Name";
```

See: C# 6: First reactions , Initializers for automatically implemented properties - By Jon Skeet

edited Jul 24 '15 at 17:22 — answered Apr 29 '14 at 14:40

26

In Version of *C# (6.0) & greater*, you can do :

For Readonly properties

```
public int ReadOnlyProp => 2;
```

For both Writable & Readable properties

```
public string PropTest { get; set; } = "test";
```

In current Version of *C# (7.0)*, you can do : (The snippet rather displays how you can use expression bodied get/set accessors to make is more compact when using with backing fields)

```
private string label = "Default Value";

// Expression-bodied get / set accessors.
public string Label
{
   get => label;
   set => this.label = value;
 }
```

edited Sep 26 '17 at 5:39

answered Mar 15 '17 at 1:07

ANewGuyInTown

2,244 1 14 17

3 The is fine but only the second of the three examples is an auto-property. – Aluan Haddad Feb 22 '18 at 8:50

26

In addition to the answer already accepted, for the scenario when you want to define a default property as a *function* of other properties you can use ***expression body notation*** on C#6.0 (and higher) for even more elegant and concise constructs like:

```
    public string Last { get; set; } = "Last";
}
```

You can use the above in the following fashion

```
var p = new Person();

p.FullName; // First Last

p.First = "Jon";
p.Last = "Snow";

p.FullName; // Jon Snow
```

In order to be able to use the above "=>" notation, the property must be read only, and you do not use the get accessor keyword.

Details on MSDN

answered Oct 30 '16 at 2:16

brakeroo
754 6 18

13

little complete sample:

```
using System.ComponentModel;

private bool bShowGroup ;
[Description("Show the group table"), Category("Sea"),DefaultValue(true)]
public bool ShowGroup
{
    get { return bShowGroup; }
    set { bShowGroup = value; }
}
```

`false` . – Boris B. Jul 29 '11 at 13:05

10

My solution is to use a custom attribute that provides default value property initialization by constant or using property type initializer.

```
[AttributeUsage(AttributeTargets.Property, AllowMultiple = false, Inherited = true)]
public class InstanceAttribute : Attribute
{
    public bool IsConstructorCall { get; private set; }
    public object[] Values { get; private set; }
    public InstanceAttribute() : this(true) { }
    public InstanceAttribute(object value) : this(false, value) { }
    public InstanceAttribute(bool isConstructorCall, params object[] values)
    {
        IsConstructorCall = isConstructorCall;
        Values = values ?? new object[0];
    }
}
```

To use this attribute it's necessary to inherit a class from special base class-initializer or use a static helper method:

```
public abstract class DefaultValueInitializer
{
    protected DefaultValueInitializer()
    {
        InitializeDefaultValues(this);
    }

    public static void InitializeDefaultValues(object obj)
    {
        var props = from prop in obj.GetType().GetProperties()
                    let attrs = prop.GetCustomAttributes(typeof(InstanceAttribute),
false)
                    where attrs.Any()
                    select new { Property = prop, Attr =
((InstanceAttribute)attrs.First()) };
        foreach (var pair in props)
        {
```

```
    }
}
```

Usage example:

```
public class Simple : DefaultValueInitializer
{
    [Instance("StringValue")]
    public string StringValue { get; set; }
    [Instance]
    public List<string> Items { get; set; }
    [Instance(true, 3,4)]
    public Point Point { get; set; }
}

public static void Main(string[] args)
{
    var obj = new Simple
        {
            Items = {"Item1"}
        };
    Console.WriteLine(obj.Items[0]);
    Console.WriteLine(obj.Point);
    Console.WriteLine(obj.StringValue);
}
```

Output:

```
Item1
(X=3,Y=4)
StringValue
```

edited Jan 18 '13 at 7:52

answered Jan 17 '13 at 22:04

introspected

139 1 5

1 As stated above, using reflection to initialize default values is both slow and overkill. Initialize on the constructor, use a non auto property or on c# 6 and

9

```
public object Foo { get; set; } = bar;
```

Note that to have a `readonly` property simply omit the set, as so:

```
public object Foo { get; } = bar;
```

You can also assign `readonly` auto-properties from the constructor.

Prior to this I responded as below.

I'd avoid adding a default to the constructor; leave that for dynamic assignments and avoid having two points at which the variable is assigned (i.e. the type default and in the constructor). Typically I'd simply write a normal property in such cases.

One other option is to do what ASP.Net does and define defaults via an attribute:

http://msdn.microsoft.com/en-us/library/system.componentmodel.defaultvalueattribute.aspx

edited Jun 17 '17 at 15:43

answered Sep 4 '08 at 17:08

Lex

548 3 15

1 This is wrong; there is no "double assignment" problem w/using the constructor.... – Mark Brackett Feb 25 '16 at 19:19

Wow, this is a blast from the past. I seem to recall this was based on the reading of the spec (partial excerpt here: msdn.microsoft.com/en-us/library/aa645756(v=vs.71).aspx). Given the time and number of versions (and Roslyn) this could well not the case anymore. Although a counter reference would be appreciated. – Lex Feb 27 '16 at 21:22

Default assignment happens automatically whether or not you use an initial value or assign in constructor. There is a slight semantic difference - field assignments happen prior to constructor calls - but the null assignment will still happen. See 10.4.5 "all instance fields...are first initialized to their default values, and then the instance field initializers are executed" msdn.microsoft.com/en-us/library/aa645757(VS.71).aspx – Mark Brackett Feb 27 '16 at 23:47

In the constructor. The constructor's purpose is to initialized it's data members.

3

Have you tried using the DefaultValueAttribute or ShouldSerialize and Reset methods in conjunction with the constructor? I feel like one of these two methods is necessary if you're making a class that might show up on the designer surface or in a property grid.

edited Sep 2 '08 at 21:38

answered Sep 2 '08 at 21:32

OwenP

18k 13 60 84

3 No, those only work in the designer, not in "real life". – Hans Kesting Jul 26 '10 at 14:21

2 @Hans: They do work "in real life", but only for setting a default value (this affects xml serialization, for example). What's wanted here is not a "default value" but an "initial value". – Ben Voigt Jun 3 '11 at 2:09

3

```
public Class ClassName{
    public int PropName{get;set;}
    public ClassName{
        PropName=0;  //Default Value
    }
}
```

answered Sep 4 '14 at 20:44

FloodMoo

31 3

3

```
private string name;
public string Name
{
    get
    {
        if(name == null)
        {
```

```
            name = value;
        }
    }
```

answered Nov 1 '18 at 14:51

ComeIn
1,004 11 10

1 I think the asker wanted an auto-property, i.e. a non-abstract property in a class or struct where you use just `get;` with a semicolon (often combined with `set;`) to indicate that the compiler should generate the body of the `get` accessor automatically. – Jeppe Stig Nielsen Nov 2 '18 at 8:59

1 Also, this code has the side effect of reverting to the default value when explicitly set to null. – MarkO Dec 5 '18 at 12:27

3

You can simple put like this

```
    public sealed  class Employee
{
    public int Id { get; set; } = 101;
}
```

answered Feb 18 at 15:10

Sunil Dhappadhule
1,030 3 20

2

Personally, I don't see the point of making it a property at all if you're not going to do anything at all beyond the auto-property. Just leave it as a field. The encapsulation benefit for these item are just red herrings, because there's nothing behind them to encapsulate. If you ever need to change the underlying implementation you're still free to refactor them as properties without breaking any dependent code.

Hmm... maybe this will be the subject of it's own question later

answered Sep 3 '08 at 3:43

10 You cannot refactor a field into an auto property without breaking the calling code. It might look the same same but the generated code is different. With auto properties the calling code calls get_propname and set_propname behind the covers, whereas it just access the field directly if it's a field. – David Reis Jul 28 '09 at 23:37

2 Yes, this is very old- I've since revised my position: stackoverflow.com/questions/205568/… – Joel Coehoorn Jul 28 '09 at 23:53

You cannot access a field across AppDomain boundaries, either -- only a property or method. – Jacob Krall Nov 6 '09 at 18:00

And you cannot declare a field in an interface only a property – yoel halb Aug 7 '13 at 2:22

2

To clarify, yes, you need to set default values in the constructor for class derived objects. You will need to ensure the constructor exists with the proper access modifier for construction where used. If the object is not instantiated, e.g. it has no constructor (e.g. static methods) then the default value can be set by the field. The reasoning here is that the object itself will be created only once and you do not instantiate it.

@Darren Kopp - good answer, clean, and correct. And to reiterate, you CAN write constructors for Abstract methods. You just need to access them from the base class when writing the constructor:

Constructor at Base Class:

```
public BaseClassAbstract()
{
    this.PropertyName = "Default Name";
}
```

Constructor at Derived / Concrete / Sub-Class:

```
public SubClass() : base() { }
```

The point here is that the instance variable drawn from the base class may bury your base field name. Setting the current instantiated object value using "this." will allow you to correctly form your object with respect to the current instance and required permission levels (access modifiers) where you are instantiating it.

edited Jun 7 '13 at 14:26 answered Jul 31 '12 at 11:02

2

Use the constructor because "When the constructor is finished, Construction should be finished". properties are like states your classes hold, if you had to initialize a default state, you would do that in your constructor.

answered Jan 11 '16 at 11:33

Preet Singh
1,281 9 15

1

```
class Person
{
    /// Gets/sets a value indicating whether auto
    /// save of review layer is enabled or not
    [System.ComponentModel.DefaultValue(true)]
    public bool AutoSaveReviewLayer { get; set; }
}
```

edited Jul 31 '14 at 13:55

Bridge
24.8k 8 49 71

answered May 26 '11 at 19:29

Nag
109 1 1

26 Welcome to Stack Overflow! Just so you know, bumping up an old question like this is generally frowned upon unless you have some good new information. However, in this case, several others have already posted about the DefaultValue attribute. If someone else has already posted what you were going to say, it's more appropriate to upvote them by clicking on the up arrow above the number next to their answer. – fire.eagle May 26 '11 at 19:34

10 @fire: Commenting requires 50 reputation. Voting also requires reputation, IIRC. – Ben Voigt Jun 3 '11 at 2:12

3 -1 as this does not initializes the property to the default value. – Gyum Fox Aug 7 '14 at 8:03

only works on the designer as stated in all the previous answers – KinSlayerUY Nov 1 '17 at 15:47

I think this would do it for ya givng SomeFlag a default of false.

```
            SomeFlag = false;

        return SomeFlag;
    }
    set
    {
        if (!_SomeFlagSet)
            _SomeFlagSet = true;

        SomeFlag = value;
    }
}
```

answered Dec 6 '13 at 21:50

user3076134
13 1

1 This is not an auto-property. – Aluan Haddad Feb 22 '18 at 8:55

-5

initialize in line, using constructors to initialize is bad practice and will lead to more breaking changes later.

answered Sep 13 '12 at 19:13

dbobrowski
509 1 6 17

-1 Are you referring to declaring the value of a variable in the same line as its definition? Since we're talking about properties here, that doesn't apply. – doppelgreener Apr 26 '13 at 0:25

Just be consistent. -1 for not being an answer. – Aluan Haddad Feb 22 '18 at 8:56

protected by Brad Larson ♦ Oct 23 '15 at 19:27