Get name of property as a string

Ask Question

175

44

(See below solution I created using the answer I accepted)

I'm trying to improve the maintainability of some code involving reflection. The app has a .NET Remoting interface exposing (among other things) a method called Execute for accessing parts of the app not included in its published remote interface.

Here is how the app designates properties (a static one in this example) which are meant to be accessible via Execute:

```
RemoteMgr.ExposeProperty("SomeSecret", typeof(SomeClass), "SomeProperty");
```

So a remote user could call:

```
string response = remoteObject.Execute("SomeSecret");
```

and the app would use reflection to find SomeClass.SomeProperty and return its value as a string.

Unfortunately, if someone renames SomeProperty and forgets to change the 3rd parm of ExposeProperty(), it breaks this mechanism.

I need to the equivalent of:

```
SomeClass.SomeProperty.GetTheNameOfThisPropertyAsAString()
```

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

Is there a way to do this? Thanks in advance.

Okay, here's what I ended up creating (based upon the answer I selected and the question he referenced):

```
// <summary>
// Get the name of a static or instance property from a property access lambda.
// </summary>
// <typeparam name="T">Type of the property</typeparam>
// <param name="propertyLambda">lambda expression of the form: '() => Class.Property' or
'() => object.Property'</param>
// <returns>The name of the property</returns>
public string GetPropertyName<T>(Expression<Func<T>> propertyLambda)
{
    var me = propertyLambda.Body as MemberExpression;

    if (me == null)
    {
        throw new ArgumentException("You must pass a lambda of the form: '() =>
Class.Property' or '() => object.Property'");
    }

    return me.Member.Name;
 }
```

Usage:

```
// Static Property
string name = GetPropertyName(() => SomeClass.SomeProperty);

// Instance Property
string name = GetPropertyName(() => someObject.SomeProperty);
```

Now with this cool capability, it's time to simplify the ExposeProperty method. Polishing doorknobs is dangerous work...

Thanks everyone.

c# reflection properties

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

asked May 12 '10 at 16:12

Jim C

1,794 6 21 28

7 Its really appriciated that you added your solution and tied things up. – Simply G. Oct 3 '12 at 7:41

possible duplicate of Get property name and type using lambda expression – nawfal Apr 27 '13 at 14:16

You should add your solution as an answer – it's much more concise than the answer your accepted. – Kenny Evitt Jan 29 '16 at 17:46

1 @Kenny Evitt: Done :) – Jim C Jan 29 '16 at 22:51

@JimC Upvoted! And linked in a comment on the currently accepted answer. Thanks! – Kenny Evitt Jan 29 '16 at 23:04

13 Answers

60

Using GetMemberInfo from here: Retrieving Property name from lambda expression you can do something like this:

```
RemoteMgr.ExposeProperty(() => SomeClass.SomeProperty)
```

```
public class SomeClass
{
    public static string SomeProperty
    {
        get { return "Foo"; }
    }
}
```

Home

PUBLIC

Stack Overflow

Tags

Users

Jobs

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

Learn More

```
        {
            var expression = GetMemberInfo(property);
            string path = string.Concat(expression.Member.Decl
                ".", expression.Member.Name);
            // Do ExposeProperty work here...
        }
    }

    public class Program
    {
        public static void Main()
        {
            RemoteMgr.ExposeProperty("SomeSecret", () => SomeC
        }
    }
```

edited Apr 4 '18 at 3:25
Rob ♦
23.7k 12 57 77

answered May 12 '10 at 16:23
Daniel Renshaw
28.8k 6 65 87

That's totally cool. Looks like it would work on any property type, too. – Jim C May 12 '10 at 16:37

I just tried it with both instance and static properties. So far so good. – Jim C May 12 '10 at 16:50

Any idea where I can get the assembly or NuGet package that contains `GetMemberInfo` ? I can't find anything with the 'common utilities' package for the Microsoft Enterprise Library, which is what MSDN seems to indicate contains that method. There's an "unofficial" package but being unofficial is uninspiring. JimC's answer, which is based on this one, is much more concise and doesn't rely on a seemingly unavailable library. – Kenny Evitt Jan 29 '16 at 23:03

1 @KennyEvitt, the method he's referencing is one written by the author of the question he linked. Alternative to that methodyou can use this Type.GetMembers

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

359

With C# 6.0, this is now a non-issue as you can do:

```
nameof(SomeProperty)
```

This expression is resolved at compile-time to `"SomeProperty"` .

[MSDN documentation of nameof](#).

edited Feb 22 '18 at 19:57

answered Jun 28 '15 at 18:47

James Ko
12.5k 14 56 130

13 This is badass and very useful for ModelState.AddModelError calls. – Michael Silver Dec 12 '15 at 19:09

It works also on VB (from Visual Studio 2015) – Max Dec 18 '15 at 11:45

7 And this is a `const string` ! Amazing – Jack Jun 5 '16 at 4:33

2 @RaidenCore sure, if you're writing for a microcontroller you should use a low-level language like C, and if you need to squeeze every bit of performance like image and video processing, you should use C or C++. but for the other 95% of applications, a managed code framework will be fast enough. Eventually C# is also compiled to machine code, and you can even pre-compile it to native if you want. – Tsahi Asher Jan 18 '17 at 7:54

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

of RaisePropertyChanged("property") – Pierre Mar 2 '17 at 9:17

17

There's a well-known hack to extract it from lambda expression (this is from the PropertyObserver class, by Josh Smith, in his MVVM foundation):

```
private static string GetPropertyName<TPropertySource>
    (Expression<Func<TPropertySource, object>> express
{
    var lambda = expression as LambdaExpression;
    MemberExpression memberExpression;
    if (lambda.Body is UnaryExpression)
    {
        var unaryExpression = lambda.Body as UnaryExpr
        memberExpression = unaryExpression.Operand as 
    }
    else
    {
        memberExpression = lambda.Body as MemberExpres
    }

    Debug.Assert(memberExpression != null,
       "Please provide a lambda expression like 'n => 

    if (memberExpression != null)
    {
        var propertyInfo = memberExpression.Member as 

        return propertyInfo.Name;
    }

    return null;
}
```

Sorry, this was missing some context. This was part of a

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

TPropertySource to extract it from the class. I recommend taking a look at the full code from the MVVM Foundation.

edited Nov 17 '15 at 4:40
Rikin Patel
5,944 6 55 69

answered May 12 '10 at 16:15
Dan Bryant
24.3k 3 45 89

With an example of how to call the function, this is certainly a +1. Oops, didn't see that there is one in the debug assertion - that's why making a developer horizontally scroll to get to the important portion of a line is evil ;) – OregonGhost May 12 '10 at 16:18

Hmmm...I need to dissect this one to understand it. – Jim C May 12 '10 at 16:26

Visual Studio 2008 flags "TPropertySource" as error ("cannot be found"). – Jim C May 12 '10 at 16:53

I just realized its a type name not just a symbol <T> as in C++. What does TPropertySource represent? – Jim C May 12 '10 at 16:58

2 To make this compile you can just change the method signature to read `public static string GetPropertyName<TPropertySource>(Expression<Func<TPropertySource, object>> expression)` then call like so: `var name = GetPropertyName<TestClass>(x => x.Foo);` – dav_i Feb 3 '14 at 10:48

Okay, here's what I ended up creating (based upon the answer I selected and the question he referenced):

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

```
// <summary>
// Get the name of a static or instance property from a pr
// </summary>
// <typeparam name="T">Type of the property</typeparam>
// <param name="propertyLambda">lambda expression of the f
'() => object.Property'</param>
// <returns>The name of the property</returns>

public string GetPropertyName<T>(Expression<Func<T>> prope
{
    var me = propertyLambda.Body as MemberExpression;

    if (me == null)
    {
        throw new ArgumentException("You must pass a lambd
Class.Property' or '() => object.Property'");
    }

    return me.Member.Name;
 }
```

Usage:

```
// Static Property
string name = GetPropertyName(() => SomeClass.SomeProperty

// Instance Property
string name = GetPropertyName(() => someObject.SomePropert
```

answered Jan 29 '16 at 22:51

Jim C
1,794 6 21 28

8

The PropertyInfo class should help you achieve this, if I understand correctly.

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

```
    PropertyInfo[] propInfos = typeof(ReflectedType).GetPr
    propInfos.ToList().ForEach(p =>
        Console.WriteLine(string.Format("Property name: {0
```

Is this what you need?

answered May 12 '10 at 16:22

Will Marcouiller
18.2k 15 77 131

Nope, although I do use GetProperties when the app receives the request for "SomeSecret". The app looks up "SomeSecret" in a map to discover it needs to find a property called "SomeProperty" in a class called "SomeClass". – Jim C May 12 '10 at 16:38

6

You can use Reflection to obtain the actual names of the properties.

http://www.csharp-examples.net/reflection-property-names/

If you need a way to assign a "String Name" to a property, why don't you write an attribute that you can reflect over to get the string name?

```
[StringName("MyStringName")]
private string MyProperty
{
    get { ... }
}
```

edited May 12 '10 at 16:25

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

answered May 12 '10 at 16:14

Robert Harvey ♦
150k 35 279 422

1 Ya, that's how the app handles incoming requests for "SomeSecret", but it doesn't give me a tool for the ExposeProperty problem. – Jim C May 12 '10 at 16:19

@Jim: See my edit – Robert Harvey ♦ May 12 '10 at 16:25

Interesting...then you could rename MyProperty to your hearts content as long as you don't mess with MyStringName, and if for some reason you do want to change it then you need to modify the ExposeProperty parm. At least I could add a comment next to the attribute with such a warning since you have to be looking at it to change the attribute's value (unlike renaming a property, which can be done from any reference location). – Jim C May 12 '10 at 16:34

5

I modified your solution to chain over multiple properties:

```
public static string GetPropertyName<T>(Expression<Func<T>
{
    MemberExpression me = propertyLambda.Body as MemberExp
    if (me == null)
    {
        throw new ArgumentException("You must pass a lambd
Class.Property' or '() => object.Property'");
    }

    string result = string.Empty;
    do
    {
        result = me.Member.Name + "." + result;
        me = me.Expression as MemberExpression;
    } while (me != null);

    result = result.Remove(result.Length - 1); // remove t
```

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

Usage:

```
string name = GetPropertyName(() => someObject.SomePropert
// returns "SomeProperty.SomeOtherProperty"
```

answered Jan 23 '14 at 0:13

hypehuman
793 1 11 33

4

Based on the answer which is already in the question and on this article: https://handcraftsman.wordpress.com/2008/11/11/how-to-get-c-property-names-without-magic-strings/ I am presenting my solution to this problem:

```
public static class PropertyNameHelper
{
    /// <summary>
    /// A static method to get the Propertyname String of 
    /// It eliminates the need for "Magic Strings" and ass
properties.
    /// See: http://stackoverflow.com/questions/2820660/ge
    /// </summary>
    /// <example>
    /// // Static Property
    /// string name = PropertyNameHelper.GetPropertyName((
    /// // Instance Property
    /// string name = PropertyNameHelper.GetPropertyName((
    /// </example>
    /// <typeparam name="T"></typeparam>
    /// <param name="propertyLambda"></param>
    /// <returns></returns>
    public static string GetPropertyName<T>(Expression<Fun
    {
        var me = propertyLambda.Body as MemberExpression;
```

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

```
Class.Property' or '() => object.Property'");
        }

        return me.Member.Name;
    }
    /// <summary>
    /// Another way to get Instance Property names as stri
    /// With this method you don't need to create a instan
    /// See the example.
    /// See: https://handcraftsman.wordpress.com/2008/11/1
without-magic-strings/
    /// </summary>
    /// <example>
    /// string name = PropertyNameHelper((Firma f) => f.Fi
    /// </example>
    /// <typeparam name="T"></typeparam>
    /// <typeparam name="TReturn"></typeparam>
    /// <param name="expression"></param>
    /// <returns></returns>
    public static string GetPropertyName<T, TReturn>(Expre
expression)
    {
        MemberExpression body = (MemberExpression)expressi
        return body.Member.Name;
    }
}
```

And a Test which also shows the usage for instance and static properties:

```
[TestClass]
public class PropertyNameHelperTest
{
    private class TestClass
    {
        public static string StaticString { get; set; }
        public string InstanceString { get; set; }
    }

    [TestMethod]
    public void TestGetPropertyName()
    {
        Assert.AreEqual("StaticString", PropertyNameHelper
TestClass.StaticString));
```

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

```
        }
    }
```

answered Apr 2 '15 at 11:30

Thomas
1,894 23 35

3

Old question, but another answer to this question is to create a static function in a helper class that uses the CallerMemberNameAttribute.

```
public static string GetPropertyName([CallerMemberName] St
  return propertyName;
}
```

And then use it like:

```
public string MyProperty {
  get { Console.WriteLine("{0} was called", GetPropertyNam
}
```

edited Jan 23 '17 at 11:07

Jesper Matthiesen
138 1 10

answered Apr 6 '15 at 9:17

Jim Pedid
1,436 1 17 23

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

0

step down a level and get the calling function).

See http://msdn.microsoft.com/en-us/library/system.diagnostics.stacktrace(VS.71).aspx

answered May 12 '10 at 16:15
Sprotty
3,136 1 23 33

I don't know where you had in mind to capture the stack trace, but I can't think of one that would contain the name of the property. – Jim C May 12 '10 at 16:25

You can do this, but this can lead to unexpected results (including exceptions) due to compiler inlining optimizations. smelser.net/blog/post/2008/11/27/… – JoeGeeky May 12 '10 at 18:34

0

I've been using this answer to great effect: Get the property, as a string, from an Expression<Func<TModel,TProperty>>

I realize I already answered this question a while back. The only advantage my other answer has is that it works for static properties. I find the syntax in this answer much more useful because you don't have to create a variable of the type you want to reflect.

edited May 23 '17 at 10:31
Community ♦
1 1

answered Aug 27 '15 at 15:08
hypehuman
793 1 11 33

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

0

I had some difficulty using the solutions already suggested for my specific use case, but figured it out eventually. I don't think my specific case is worthy of a new question, so I am posting my solution here for reference. (This is very closely related to the question and provides a solution for anyone else with a similar case to mine).

The code I ended up with looks like this:

```
public class HideableControl<T>: Control where T: class
{
    private string _propertyName;
    private PropertyInfo _propertyInfo;

    public string PropertyName
    {
        get { return _propertyName; }
        set
        {
            _propertyName = value;
            _propertyInfo = typeof(T).GetProperty(value);
        }
    }

    protected override bool GetIsVisible(IRenderContext co
    {
        if (_propertyInfo == null)
            return false;

        var model = context.Get<T>();

        if (model == null)
            return false;

        return (bool)_propertyInfo.GetValue(model, null);
    }

    protected void SetIsVisibleProperty(Expression<Func<T,
    {
        var expression = propertyLambda.Body as MemberExpr
        if (expression == null)
            throw new ArgumentException("You must pass a la
vm.Property'");
```

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

```
}

public interface ICompanyViewModel
{
    string CompanyName { get; }
    bool IsVisible { get; }
}

public class CompanyControl: HideableControl<ICompanyViewM
{
    public CompanyControl()
    {
        SetIsVisibleProperty(vm => vm.IsVisible);
    }
}
```

The important part for me is that in the `CompanyControl` class the compiler will only allow me to choose a boolean property of `ICompanyViewModel` which makes it easier for other developers to get it right.

The main difference between my solution and the accepted answer is that my class is generic and I only want to match properties from the generic type that are boolean.

edited Dec 23 '16 at 15:36
halfer
14.8k 7 59 119

answered Dec 18 '16 at 22:11
bikeman868
651 7 18

it's how I implemented it , the reason behind is if the class that you want to get the name from it's member is not

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.

```
public static string GetName<TClass>(Expression<Func<TClas
{
    MemberExpression body = exp.Body as MemberExpression;

    if (body == null)
    {
         UnaryExpression ubody = (UnaryExpression)exp.Body
         body = ubody.Operand as MemberExpression;
    }

     return body.Member.Name;
}
```

the usage is like this

```
var label = ClassExtension.GetName<SomeClass>(x => x.Label
'SomeClass'
```

answered May 29 '17 at 4:20

Mo Hrad A
667 1 8 26

By using our site, you acknowledge that you have read and understand our Cookie Policy, Privacy Policy, and our Terms of Service.