What is the purpose of nameof?

Asked 4 years, 3 months ago Active 7 months ago Viewed 124k times

238

35

Version 6.0 got a new feature of `nameof`, but I can't understand the purpose of it, as it just takes the variable name and changes it to a string on compilation.

I thought it might have some purpose when using `<T>` but when I try to `nameof(T)` it just prints me a `T` instead of the used type.

Any idea on the purpose?

c# .net c#-6.0 nameof

edited Aug 3 '15 at 18:40
Patrick Hofman
134k 18 194 259

asked Jul 29 '15 at 9:04
atikot
1,715 3 17 27

2 See also msdn.microsoft.com/library/dn986596.aspx – Corak Jul 29 '15 at 9:08

25 There wasn't a way to get that `T` before. There was a way to get the used type before. – Jon Hanna Jul 29 '15 at 9:10

12 Definitely useful when refactory/renaming the name within `nameof`. Also helps to prevent typos. – bvj May 11 '17 at 18:17

4 The official documentation of *nameof* has moved to here: docs.microsoft.com/en-us/dotnet/csharp/language-reference/… - It also lists key use cases which serve as a pretty good answer to the question. – markus s Apr 20 '18 at 6:40

1 Possible duplicate of What is a NullReferenceException, and how do I fix it? – Matt J Oct 22 '18 at 12:09

15 Answers

295

What about cases where you want to reuse the name of a property, for example when throwing exception based on a property name, or handling a `PropertyChanged` event. There are numerous cases where you would want to have the name of the property.

Take this example:

```
switch (e.PropertyName)
{
    case nameof(SomeProperty):
    { break; }

    // opposed to
    case "SomeOtherProperty":
    { break; }
}
```

In the first case, renaming `SomeProperty` will change the name of the property too, or it will break compilation. The last case doesn't.

This is a very useful way to keep your code compiling and bug free (sort-of).

(A very nice article from Eric Lippert why `infoof` didn't make it, while `nameof` did)

edited Mar 9 '18 at 14:28

answered Jul 29 '15 at 9:07

Patrick Hofman

134k 18 194 259

1 I understand the point, just adding that resharper changes the strings when refactoring names, not sure if VS has similar functionality. – Ash Burlaczenko Jul 29 '15 at 11:26

6 It has. But both Resharper and VS don't work over projects for example. This does. In fact, this is the better solution. – Patrick Hofman Jul 29 '15 at 11:28

43 Another common use case is routing in MVC using `nameof` and the action's name instead of a hard-coded string. – RJ Cuthbertson Jul 29 '15 at 14:21

2 @sotn I'm not sure I understand what you're asking. There's nothing stopping you from using it like `public class MyController { public ActionResult Index() { return View(nameof(Index)); } }` - and you can use `nameof` on non-static members (for instance you can call `nameof(MyController.Index)` using the class above and it will emit "Index"). Check out the examples at msdn.microsoft.com/en-us/library/… – RJ Cuthbertson Oct 24 '16 at 19:56

2 I don't see why that is special. Variable names are always the same, right? Whether you have an instance or not, the variable names won't change @sotn – Patrick Hofman Oct 25 '16 at 7:37

It's really useful for `ArgumentException` and its derivatives:

165

```
public string DoSomething(string input)
{
    if(input == null)
    {
        throw new ArgumentNullException(nameof(input));
    }
    ...
```

Now if someone refactors the name of the `input` parameter the exception will be kept up to date too.

It is also useful in some places where previously reflection had to be used to get the names of properties or parameters.

In your example `nameof(T)` gets the name of the type parameter - this can be useful too:

```
throw new ArgumentException(nameof(T), $"Type {typeof(T)} does not support this
method.");
```

Another use of `nameof` is for enums - usually if you want the string name of an enum you use `.ToString()`:

```
enum MyEnum { ... FooBar = 7 ... }

Console.WriteLine(MyEnum.FooBar.ToString());

> "FooBar"
```

This is actually relatively slow as .Net holds the enum value (i.e. `7`) and finds the name at run time.

Instead use `nameof`:

```
Console.WriteLine(nameof(MyEnum.FooBar))

> "FooBar"
```

Now .Net replaces the enum name with a string at compile time.

Yet another use is for things like `INotifyPropertyChanged` and logging - in both cases you want the name of the member that you're calling to be passed to another method:

```
// Property with notify of change
public int Foo
{
    get { return this.foo; }
    set
    {
        this.foo = value;
        PropertyChanged(this, new PropertyChangedEventArgs(nameof(this.Foo));
    }
}
```

Or...

```
// Write a log, audit or trace for the method called
void DoSomething(... params ...)
{
    Log(nameof(DoSomething), "Message....");
}
```

edited Jul 31 '15 at 7:52

answered Jul 29 '15 at 9:07

Keith

101k 63 249 366

9 And you've put another cool feature in: string interpolation! – Patrick Hofman Jul 29 '15 at 9:14

1 @PatrickHofman and `typeof(T)` , which is another piece of compile-time sugar that's useful in similar circumstances :-) – Keith Jul 29 '15 at 9:18

One thing that i'm missing is something like `nameofthismethod` . You can use `Log.Error($"Error in {nameof(DoSomething)}...")` but if you copy-paste this to other methods you won't notice that it's still referring to `DoSomething` . So while it's working perfectly with local variables or parameters the method-name is a problem. – Tim Schmelter Jul 21 '16 at 8:48

3 Do you know if `nameOf` will use the `[DisplayName]` attribute if present? For the `enum` example I use `[DisplayName]` often with MVC projects – Luke T O'Brien Jun 15 '17 at 15:20

2 @AaronLS Yes, it's fairly specialised and not something that you would use often. However that `throw new` is a whole other anti-pattern - I find over-using `catch` to be a common problem with junior devs because it feels like fixing the problem (when most of the time it's just hiding it). – Keith Jul 24 at 14:27

Another use-case where `nameof` feature of C# 6.0 becomes handy - Consider a library like Dapper which makes DB retrievals much easier. Albeit this is a great library, you need to hardcode property/field names within query. What this means is that if you decide to

25

rename your property/field, there are high chances that you will forget to update query to use new field names. With string interpolation and `nameof` features, code becomes much easier to maintain and typesafe.

From the example given in link

without nameof

```
var dog = connection.Query<Dog>("select Age = @Age, Id = @Id", new { Age = (int?)null,
Id = guid });
```

with nameof

```
var dog = connection.Query<Dog>($"select {nameof(Dog.Age)} = @Age, {nameof(Dog.Id)} =
@Id", new { Age = (int?)null, Id = guid });
```

edited Oct 25 '18 at 15:02

afk

100 7

answered Nov 16 '16 at 1:57

sateesh

7,518 2 17 39

21

Your question already expresses the purpose. You must see this might be useful for logging or throwing exceptions.

for example.

```
public void DoStuff(object input)
{
    if (input == null)
    {
        throw new ArgumentNullException(nameof(input));
    }
}
```

this is good, **if I change the name of the variable the code will break instead or returning an exception with an incorrect message**.

Of course, the uses are not limited to this simple situation. You can use `nameof` whenever it would be useful to code the name of a variable or property.

The uses are manifold when you consider various binding and reflection situations. Its an excellent way to bring what were run time errors to compile time.

edited Aug 10 '15 at 13:31

answered Jul 29 '15 at 9:07

Jodrell
28k 3 61 107

6 @atikot: But then, if you rename the variable, the compiler won't notice that the string doesn't match any more. – O. R. Mapper Jul 29 '15 at 9:09

1 I actually use resharper which takes it to account, but i see your point. – atikot Jul 29 '15 at 9:11

4 @atikot, so do I, but Resharper only generates a warning, not a compiler error. There's a difference between a certainty and good advice. – Jodrell Jul 29 '15 at 9:12

1 @atikot, and, Resharper doesn't check logging messages – Jodrell Jul 29 '15 at 9:14

2 @Jodrell: And, I suspect, it doesn't check various other uses, either - how about WPF bindings created in code-behind, custom `OnPropertyChanged` methods (that directly accept property names rather than `PropertyChangedEventArgs`), or calls to reflection to look for a particular member or type? – O. R. Mapper Jul 29 '15 at 9:19

14

The most common use case I can think of is when working with the `INotifyPropertyChanged` interface. (Basically everything related to WPF and bindings uses this interface)

Take a look at this example:

```
public class Model : INotifyPropertyChanged
{
    // From the INotifyPropertyChanged interface
    public event PropertyChangedEventHandler PropertyChanged;

    private string foo;
    public String Foo
    {
        get { return this.foo; }
        set
        {
            this.foo = value;
            // Old code:
            PropertyChanged(this, new PropertyChangedEventArgs("Foo"));

            // New Code:
            PropertyChanged(this, new PropertyChangedEventArgs(nameof(Foo)));
```

```
        }
    }
}
```

As you can see in the old way we have to pass a string to indicate which property has changed. With `nameof` we can use the name of the property directly. This might not seem like a big deal. But image what happens when somebody changes the name of the property `Foo`. When using a string the binding will stop working, but the compiler will not warn you. When using nameof you get a compiler error that there is no property/argument with the name `Foo`.

Note that some frameworks use some reflection magic to get the name of the property, but now we have nameof this is no longer neccesary.

edited Nov 24 '15 at 22:33

Patrick Hofman
134k 18 194 259

answered Jul 29 '15 at 19:03

Roy T.
7,305 2 38 58

5 Whilst this is a valid approach, a more convenient (and DRY) approach is to use the `[CallerMemberName]` attribute on a new method's param to raise this event. – Drew Noakes Jul 29 '15 at 22:18

1 I agree CallerMemberName is also nice, but its a separate use case, since (as you said) you can only use it in methods. As for DRY, I'm not sure if `[CallerMemberName]string x = null` is better than `nameof(Property)`. You could say the property name is used twice, but thats basically the argument passed to a function. Not really what is meant with DRY I think :). – Roy T. Jul 30 '15 at 6:46

Actually you can use it in properties. They are members too. The benefit over `nameof` is that the property setter needn't specify the property name at all, eliminating the possibility of copy/paste bugs. – Drew Noakes Jul 30 '15 at 7:44

4 It's a 'better together' situation for `INotifyPropertyChanged`, using the `[CallerMemberNameAttribute]` allows the change notification to be cleanly raised from a property setter, while the `nameof` syntax allows a change notification to be cleanly raised from a different location in your code. – Andrew Hanlon Jul 30 '15 at 13:16

8

Most common usage will be in input validation, such as

```
//Currently
void Foo(string par) {
   if (par == null) throw new ArgumentNullException("par");
}

//C# 6 nameof
void Foo(string par) {
```

```
    if (par == null) throw new ArgumentNullException(nameof(par));
}
```

In first case, if you refactor the method changing *par* parameter's name, you'll probably forget to change that in the *ArgumentNullException*. With *nameof* you don't have to worry about that.

See also: nameof (C# and Visual Basic Reference)

answered Jul 29 '15 at 9:09

Massimo Prota
1,081 6 11

7

The ASP.NET Core MVC project uses `nameof` in the `AccountController.cs` and `ManageController.cs` with the `RedirectToAction` method to reference an action in the controller.

Example:

```
return RedirectToAction(nameof(HomeController.Index), "Home");
```

This translates to:

```
return RedirectToAction("Index", "Home");
```

and takes takes the user to the 'Index' action in the 'Home' controller, i.e. `/Home/Index` .

edited May 4 '16 at 22:38

answered Apr 22 '16 at 13:55

Fred
7,170 2 39 45

Why not go whole-hog and use `return RedirectToAction(nameof(HomeController.Index), nameof(HomeController).Substring(nameof(HomeController),0,nameof(HomeController).Length-"Controller".Length));` ? – Suncat2000 Nov 3 '17 at 19:02

4 @Suncat2000 Oh boy is that ugly. – Fred Nov 6 '17 at 12:43

@Suncat2000 because one of those things is done in compilation and the other isn't? :) – Dinerdo Aug 16 '18 at 16:42

6

As others have already pointed out, the `nameof` operator does insert the name that the element was given in the sourcecode.

I would like to add that this is a really good idea in terms of refactoring since it makes this string refactoring safe. Previously, I used a static method which utilized reflection for the same purpose, but that has a runtime performance impact. The `nameof` operator has no runtime performance impact; it does its work at compile time. If you take a look at the `MSIL` code you will find the string embedded. See the following method and its disassembled code.

```
static void Main(string[] args)
{
    Console.WriteLine(nameof(args));
    Console.WriteLine("regular text");
}

// striped nops from the listing
IL_0001 ldstr args
IL_0006 call System.Void System.Console::WriteLine(System.String)
IL_000C ldstr regular text
IL_0011 call System.Void System.Console::WriteLine(System.String)
IL_0017 ret
```

However, that can be a drawback if you plan to obfuscate your software. After obfuscation the embedded string may no longer match the name of the element. Mechanisms that rely on this text will break. Examples for that, including but not limited to are: Reflection, NotifyPropertyChanged ...

Determining the name during runtime costs some performance, but is safe for obfuscation. If obfuscation is neither required nor planned, I would recommend using the `nameof` operator.

edited Sep 27 '18 at 19:45
Sildoreth
1,558 20 35

answered Aug 1 '15 at 12:36
cel sharp
130 9

5

Consider that you use a variable in your code and need to get the name of the variable and lets say print it, you should have to use

```
int myVar = 10;
print("myVar" + " value is " + myVar.toString());
```

And if then someone refactors the code and uses another name for "myVar", he/she would have to watch for the string value in your code and chenge it accordingly.

Instead if you had

```
print(nameof(myVar) + " value is " + myVar.toString());
```

It would help to refactor automatically!

edited Jul 29 '15 at 9:16

Patrick Hofman
134k 18 194 259

answered Jul 29 '15 at 9:12

cnom
1,260 2 12 41

I wish there had been a special variable-parameter syntax which would pass an array of tuples, one for each parameter, containing the source code representation, the `Type` , and the value. That would make it possible for code invoking logging methods to eliminate a lot of redundancy. – supercat Jul 29 '15 at 16:07

4

[The MSDN article] lists MVC routing (the example that really clicked the concept for me) among several others. The (formatted) description paragraph reads:

> - **When reporting errors in code,**
> - **hooking up model-view-controller (MVC) links,**
> - **firing property changed events, etc.,**
>
> you often want to **capture the string name of a method**. Using nameof helps **keep your code valid when renaming definitions.**
>
> ***Before you had to use string literals** to refer to definitions, **which is brittle** when renaming code elements **because tools do not know to check these string literals.***

The accepted / top rated answers already give several excellent concrete examples.

answered Aug 5 '15 at 16:16

brichins
2,682 1 31 50

The purpose of the `nameof` operator is to provide the source name of the artifacts.

3

Usually the source name is the same name as the metadata name:

```
public void M(string p)
{
    if (p == null)
    {
        throw new ArgumentNullException(nameof(p));
    }
    ...
}

public int P
{
    get
    {
        return p;
    }
    set
    {
        p = value;
        NotifyPropertyChanged(nameof(P));
    }
}
```

But this may not always be the case:

```
using i = System.Int32;
...
Console.WriteLine(nameof(i)); // prints "i"
```

Or:

```
public static string Extension<T>(this T t)
{
    return nameof(T); returns "T"
}
```

One use I've been giving to it is for naming resources:

```
[Display(
    ResourceType = typeof(Resources),
    Name = nameof(Resources.Title_Name),
    ShortName = nameof(Resources.Title_ShortName),
```

```
Description = nameof(Resources.Title_Description),
Prompt = nameof(Resources.Title_Prompt))]
```

The fact is that, in this case, I didn't even need the generated properties to access the resources, but now I have a compile time check that the resources exist.

edited Sep 4 '15 at 11:13

cel sharp
130 9

answered Jul 31 '15 at 0:06

Paulo Morgado
7,881 1 18 35

0

One of the usage of `nameof` keyword is for setting `Binding` in wpf **programmatically**.

to set `Binding` you have to set `Path` with string, and with `nameof` keyword, it's possible to use Refactor option.

For example, if you have `IsEnable` dependency property in your `UserControl` and you want to bind it to `IsEnable` of some `CheckBox` in your `UserControl` , you can use these two codes:

```
CheckBox chk = new CheckBox();
Binding bnd = new Binding ("IsEnable") { Source = this };
chk.SetBinding(IsEnabledProperty, bnd);
```

and

```
CheckBox chk = new CheckBox();
Binding bnd = new Binding (nameof (IsEnable)) { Source = this };
chk.SetBinding(IsEnabledProperty, bnd);
```

It's obvious the first code can't refactor but the secend one...

answered Jul 17 '16 at 23:58

Mehdi Khademloo
1,693 1 11 30

Previously we were using something like that:

0

```
// Some form.
SetFieldReadOnly( () => Entity.UserName );
...
// Base form.
private void SetFieldReadOnly(Expression<Func<object>> property)
{
    var propName = GetPropNameFromExpr(property);
    SetFieldsReadOnly(propName);
}

private void SetFieldReadOnly(string propertyName)
{
    ...
}
```

Reason - compile time safety. No one can silently rename property and break code logic. Now we can use nameof().

answered Jul 26 '16 at 22:09

QtRoS
786 1 12 18

But nameof works also for local variables, fields, methods... – Massimiliano Kraus Oct 5 '16 at 22:46

0

Another use case of `nameof` is to check tab pages, instead of checking the index you can check the `Name` property of the tabpages as follow:

```
if(tabControl.SelectedTab.Name == nameof(tabSettings))
{
    // Do something
}
```

Less messy :)

answered Apr 1 at 8:20

diedrop
6 4

0

It has advantage when you use ASP.Net MVC. When you use HTML helper to build some control in view it uses property names in name attribure of html input:

```
@Html.TextBoxFor(m => m.CanBeRenamed)
```

It makes something like that:

```
<input type="text" name="CanBeRenamed" />
```

So now, if you need to validate your property in Validate method you can do this:

```
public IEnumerable<ValidationResult> Validate(ValidationContext validationContext) {
  if (IsNotValid(CanBeRenamed)) {
    yield return new ValidationResult(
      $"Property {nameof(CanBeRenamed)} is not valid",
      new [] { $"{nameof(CanBeRenamed)}" })
  }
}
```

In case if you rename you property using refactoring tools, your validation will not be broken.

answered Mar 20 '18 at 13:43

dgtlfx
31 4