

# Why shouldn't I prefix my fields?

## [closed]

Asked 15 years, 9 months ago   Modified 9 months ago

Viewed 20k times



50



As it currently stands, this question is not a good fit for our Q&A format. We expect answers to be supported by facts, references, or expertise, but this question will likely solicit debate, arguments, polling, or extended discussion. If you feel that this question can be improved and possibly reopened, [visit the help center](#) for guidance.

Closed 12 years ago.

I've never been a fan of Hungarian notation, I've always found it pretty useless unless you're doing some really low level programming, but in every C++ project I've worked on some kind of Hungarian notation policy was enforced, and with it the use of some 'not-really-Hungarian' prefixes as `m_` for fields, `s_` for statics, `g_` for globals and so on.

Soon I realized how much useless it was in C# and gradually started to drop all of my old habits... but the '`m_`' thing. I still use the `m_` prefix on private fields because I really find it very useful to being able to distinguish between parameters, locals and fields.

The [naming conventions for fields page at MSDN](#) says I shouldn't, but it does not say why (the way e.g. Google's conventions generally tend to rationalize their prescriptions).

Are there reasons why I shouldn't or is it only a matter of style. If it is the latter, are prefixes generally considered a bad style and can I expect negative reactions from other people working on the codebase?

c#

field

naming-conventions

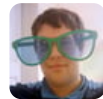
hungarian-notation

Share

Improve this question

Follow

edited Jul 3, 2015 at 22:07



user7610

28.5k ● 17 ● 142 ● 165

asked Mar 18, 2009 at 18:30




Trap

12.4k ● 15 ● 57 ● 69

- 
- 7 This has been discussed many times on this forum. It is a religious debate, and you will never find a 'correct' answer.  
– [John Kraft](#) Mar 18, 2009 at 18:32
- 

According to one of the answers, the MSDN articles about style were written by some Brad Adams. He explains the naming rules in a blog "The reasons to extend the public rules (no Hungarian, no prefix for member variables, etc.) is to produce a consistent source code appearance. In addition a goal is to have clean readable source. Code legibility should be a primary goal."

- 1 Microsoft's [naming\\_guidelines](#) for .NET (as enforced by Visual Studio's **Code Analysis** tool) do not require adherence by private types and members: *"Although adopting these naming conventions as general code development guidelines would result in more consistent naming throughout your code, you are required only to apply them to APIs that are publicly exposed (public or protected types and members, and explicitly implemented interfaces)."* – DavidRR Jul 12, 2019 at 16:04 

25 Answers

Sorted by:

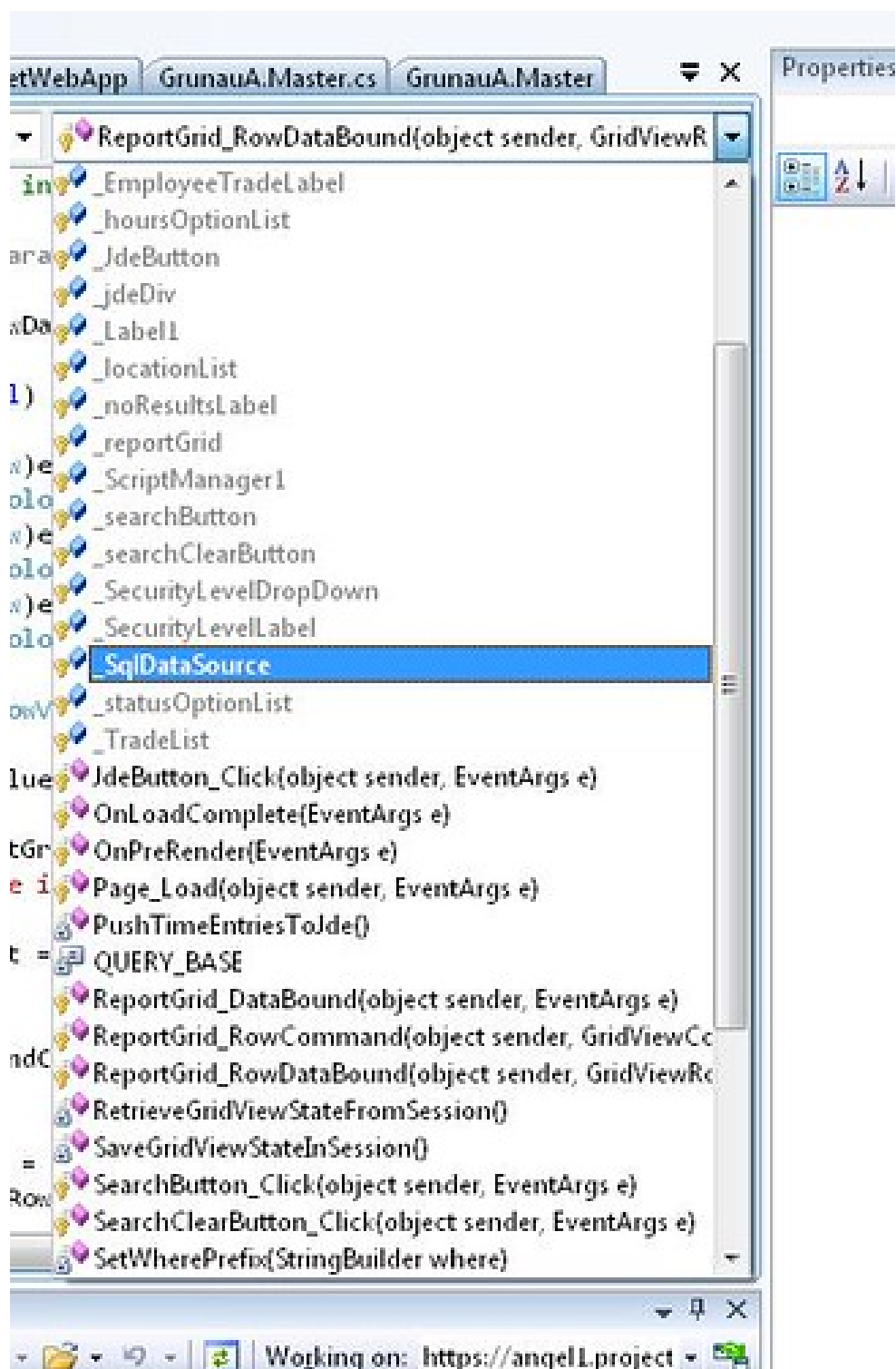
Highest score (default)



55

I like the underbar prefix for member fields. Mostly I like it because that way, all of my member fields are shown alphabetically before my methods in the wizard bar at the top of the screen.





Share Improve this answer

edited Feb 8, 2017 at 14:11

Follow



Community Bot

1 • 1

answered Mar 18, 2009 at 18:51



Matt Brunell

10.4k • 3 • 36 • 46

- 
- 1 That helps with intellisense, too. – [Brian](#) Mar 18, 2009 at 19:11
- 
- 2 I add my vote for prefixing member fields with underscore. It really helps. – [Sergio Acosta](#) Mar 18, 2009 at 22:32
- 
- 4 Not a good idea. Underscore prefixes can be reserved. If we use this as a habit we'll eventually bump into those collisions one day that force us to break the convention. – [kizzx2](#) Jul 22, 2010 at 12:40
- 
- 2 @kizzx2 Internally c# uses underscore prefixes for the backing fields on auto-implemented properties so using this for your own private fields would matching implementation. Where else are underscore prefixes reserved? – [Seph](#) May 2, 2012 at 13:56
- 
- 6 @Seph I have since started using the \_ prefix in C# myself. Forget about that comment 2 year ago :P – [kizzx2](#) May 2, 2012 at 15:57
- 



42

When you should:

- When your project coding guidelines say you should



When you shouldn't:

- When your project coding guidelines say you shouldn't



If you don't have any guidelines yet, you're free to choose whatever you or your team want and feel most comfortable with. Personally when coding C++ I tend to use `m_` for members, it does help. When coding in other

languages, particularly those without true classes (like Javascript, Lua) I don't.

In short I don't believe there is a "right" and a "wrong" way.

Share Improve this answer

edited Dec 13, 2019 at 14:48

Follow



Dan

10.9k ● 11 ● 67 ● 106

answered Mar 18, 2009 at 18:35



MattJ

7,924 ● 1 ● 30 ● 33

---

3 He's not asking for the right or wrong way, but opinions what we think is the right way. – Ben S Mar 18, 2009 at 18:41

---

2 As I said, the right way is his way ;) – MattJ Mar 18, 2009 at 18:50

---

7 I thought it was clear what I was asking. This is not about when I should follow guidelines or not. What I fail to understand is why people keeps voting this answer. Probably I failed to make myself clear enough. – Trap May 5, 2009 at 13:44

---

Presumably we agree about following guidelines. But I did also answer for the case when there are no guidelines. And like I said, there is no right/wrong, it is subjective. I also gave my personal preference too. What more did you want? :) – MattJ May 13, 2009 at 15:30

---



The auto-implemented property feature in C# 3.0 creates less of a need for this convention one way or the other.

21

Instead of writing



```
string name_;  
public string Name { get { return name_; } }
```



you can now write

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

Since you no longer need the explicit backing store variable, you no longer have to come up with a name for it, thus avoiding this entire discussion.

Obviously, this argument doesn't apply when you really need explicit backing store such as to perform validation.

Share Improve this answer

edited Mar 21 at 13:09

Follow

answered Mar 18, 2009 at 18:38



Dan

10.9k ● 11 ● 67 ● 106

---

Note: This will cause the value of the property to not be easily visible in the debugger due to optimizations. – Ben S Mar 18, 2009 at 18:39

---

6 This is fine for auto-properties, but properties that have logic in the get/set still require a backing store, so it doesn't answer the question. – mqp Mar 18, 2009 at 18:40

---

2 So what about private members that don't expose a public property, i.e. `m_ConnectionString`, which reads from the

- 1    ahem , I would use `_name` for a local variable. – [johnc](#) Mar 18, 2009 at 22:58
- 



11



As some have alluded to, the MS guidelines say:

Do not use a prefix for field names. For example, do not use `g_` or `s_` to distinguish static versus non-static fields.

I happen to agree with this. prefixes make your code look ugly and waste space with inconsequential characters. Having said that, it is often common to use fields to back properties where both the field and the property would have the same name (with the private field being camel case and the property being pascal case). In VB, this doesn't work, since VB isn't case-sensitive. In this scenario, I recommend the use of a single `_` prefix. No more, no less. It just looks cleaner, IMHO.

Share   Improve this answer

answered Mar 18, 2009 at 18:53

Follow



[Chris](#)

28k ● 26 ● 130 ● 227

- 4    This is quite funny considering that the .NET Framework is full of private fields prefixed by `m_` – [Tamas Czinege](#) Mar 18, 2009 at 19:31
-



I believe irony is the word you're looking for, and yes, the .NET BCL is full of violations of MS's own recommendations.  
:-P – [Chris](#) Mar 18, 2009 at 20:55

- 
- 4 Actually, the MS guidelines say that internal and private fields are not covered by these guidelines ([msdn.microsoft.com/en-us/library/ms229012.aspx](https://msdn.microsoft.com/en-us/library/ms229012.aspx)) – [Sean](#) Jul 26, 2013 at 16:03
- 



11



I have experimented with m\_, s\_, just \_, and no prefix at all. I have settled on using just \_ for all static and instance variables. I don't find it important to distinguish static variables from instance variables. In theory it sounds good, in practice it doesn't create a problem.



A coworker once made a convincing argument to eliminate all prefixes, we tried it on one project and it worked better than I expected. I carried it forward to my next project and became annoyed that it "interferes" with Intellisense. When you have the following situation

```
int foo;
public int Foo
{
    get { return foo; }
}
```

Starting to type foo will suggest both the instance variable and the property. Prefixing the variable with an underscore eliminates the annoying double suggestion, so I switched back to using just \_.

Follow



ScottS

8,543 ● 5 ● 31 ● 50



10



I try to follow the [MSDN .NET library guidelines](#). They include a [naming guidelines](#) section.

Obviously, these are secondary to your project guidelines.

Share Improve this answer

answered Mar 18, 2009 at 18:37

Follow



Ben S

69.3k ● 31 ● 173 ● 214

1 I do for the most part, but I am not a fan of how they would like you to name private members. Their way causes conflicts with method parameters. – [Ed Swangren](#) Mar 18, 2009 at 18:42

If I have private properties that can't be auto-implemented I often cheat on these guidelines and add an underscore to the private field. Old habit from when I started to learn Java.  
– [Ben S](#) Mar 18, 2009 at 18:46

2 there is no confusion with method parameters. C# already has a way to identify a variable as a class member, the "this." prefix. – [Joseph Kingry](#) Mar 18, 2009 at 19:47

1 That's the issue. Disambiguation is required, thus causing confusion. – [Ben S](#) Mar 18, 2009 at 20:37

2 Exactly. I don't like typing 'this' over and over just because the variable names are the same when they don't need to be.  
– [Ed Swangren](#) Mar 18, 2009 at 21:04



10



I prefer to mark property backing fields (although as already mentioned .NET 3.0+ reduces the need thanks to Automatic Properties) with underscores but not the "m". For one it puts them at the top of the IntelliSense list when I come to use them.

I will admit that I need to brush-up on the guidelines on MSDN, things can change so quickly these days.

Share Improve this answer

answered Mar 18, 2009 at 18:48

Follow



[argibson](#)

256 ● 2 ● 10



10



With tools like resharper there's really no reason for prefixes. Also if you write short methods, you should be able to tell really quickly where the var is coming from. Finally, I guess I wouldn't really see the need to tell a difference between a static or not because again resharper is going to red line it if you try to do something you're not able to. Even without resharper you're probably saved by the compiler.

Share Improve this answer

answered Mar 18, 2009 at 18:50

Follow



[rball](#)

6,955 ● 7 ● 50 ● 78

---

1 I dunno why this is downmodded, it's a good answer. +1  
– [Randolpho](#) Mar 18, 2009 at 19:20

---

My problem is a class may have a private field 'firstName' and in its constructor it will often have a parameter also

called 'firstName'. It seems to be a common pattern, regardless of how decoupled the classes are (this.firstName is too dangerous with the risk of leaving off the 'this.').

– [LegendLength](#) Mar 19, 2009 at 10:19

---

I still prefer to distinguish members from local variables and parameters, prefix is excellent for that. (Not that I -1'd this answer - but I'm tempted ;)) – [peterchen](#) Mar 19, 2009 at 14:30

---

@legendlength: I'm just spoiled then I guess. Resharper handles instantiation of the field, and the this.sameNameAsCtorVar for ya. @peterchen I guess we just code differently. I prefer a ton of short methods and don't need a \_ to tell me what type it is. – [rball](#) Mar 19, 2009 at 19:49

---

- 1 You are spoiled. We can't afford Resharper :( (wish there was a cheaper alternative that was just as good). – [LegendLength](#) Jun 9, 2009 at 11:34
- 



6



I always prefix member variables with **m\_** and static variables with **s\_** for the same reasons that you state.

Some people prefix member variables with an underscore, but I've always found this a bit odd looking (but that's just a personal preference).



Most people I work with use the m\_/s\_ prefix. I don't really think it matters too much what you use, as long as you're consistent.

Share Improve this answer

Follow

answered Mar 18, 2009 at 18:36



[Sean](#)

62.4k ● 11 ● 99 ● 138

---

Ah that's where that strange convention comes from... ^^  
Always thought it was a VB-thing (because of `_` being used  
as the new line escape character). – [Christian Klauser](#) Mar  
18, 2009 at 19:08

---

- 1 Additionally, I use a `_` for function arguments.  
– [CW Holeman II](#) Jun 24, 2009 at 0:52
-



5



I never use them. It encourages sloppy coding. The MSDN coding guidelines, that's where it's at.

Share Improve this answer

answered Mar 18, 2009 at 18:41

Follow



Inferis

4,642 ● 6 ● 41 ● 47

---

4 Why? How should this encourage sloppy coding???

– [marc\\_s](#) Mar 18, 2009 at 18:59

---

Because it allows you to use the same names for fields and locals or parameters. It's a good way to keep your naming clear. – [Inferis](#) Mar 18, 2009 at 19:33

---

9 What is sloppy about naming a constructor parameter 'name' and the field that will hold the value '\_name'? – [Sergio Acosta](#) Mar 18, 2009 at 22:34

---

1 @SergioAcosta There is nothing sloppy about it. It is idiotic to dream up another name for a parameter just to make it different from a member variable. Such behaviour is likely to lead to more verbose and potentially unclear code.

– [Ob101010](#) Jun 25, 2015 at 16:54 ✎



4



Here are a few reasons to use `_` (and not `m_`).

(1) Many BCL guys do it despite MS's naming guide. (Check out their [blog](#).) Those guys write the framework, so they have some good habits worth copying. Some of the most helpful example code on MSDN is written by



them, and so uses the underscore convention. It's a de-facto industry standard.

(2) A single underscore is a noticeable yet unobtrusive way to disambiguate method and class-level variables by simply reading the source. It helps people understand new (or old) code ***at-a-glance*** when reading it. Yes, you can mouse-over to see this in an IDE, but we shouldn't be forced to. You may want to read it in a text editor, or dare I say it, on paper.

(3) Some say you don't need any prefix as methods will be short, and later if needed you can change the field to an auto-implemented property. But in the real world methods are as long as they need to be, and there are important differences between fields and properties (e.g. serialization and initialization).

Footnote: The "m" for member in `m_` is redundant in our usage here, but it was lower case because one of the ideas in many of these old naming conventions was that type names started with upper case and instance names started with lower case. That doesn't apply in .NET so it's doubly redundant. Also Hungarian notation was sometimes useful with old C compilers (e.g. integer or pointer casting and arithmetic) but even in C++ its usefulness was diminished when dealing with classes.

Share Improve this answer

edited Jan 6, 2016 at 5:54

Follow

answered May 3, 2010 at 23:37



Nick Westgate

3,263 ● 2 ● 36 ● 43

---

Despite what I've said, I think your intention are good (as in, wise people are using it, so pause should be given when questioning it). I also understand this comment I'm replying to is getting close to being a decade old and things change through time. There are a lot more languages now. – [Water](#)  
Nov 8, 2019 at 4:08

---



4

As @John Kraft mentions, there is no "correct" answer. MattJ is the closest—you should always follow your company's style guidelines. When in Rome, and all that.



As for my personal opinion, since it's called for here, I vote that you drop `m_` entirely.



I believe the best style is one where all members are `PascalCased`, regardless of visibility (that means even `private` members), and all arguments are `camelCased`. I do not break this style.

I can understand the desire to prefix property backing store field; after all you must differentiate between the field and the property, right? I agree, you must. But use a post-fix.

Instead of `m_MyProperty` (or even `_MyProperty`, which I've seen and even promoted once upon a time), use `MyPropertyValue`. It's easier to read and understand and



-- more importantly -- it's close to your original property name in intellisense.

Ultimately, that's the reason I prefer a postfix. If I want to access `MyPropertyValue` using intellisense you (typically) type "`My` `<down-arrow>` `<tab>`", since by then you're close enough that only `MyProperty` and `MyPropertyValue` are on the list. If you want to access `m_MyProperty` using intellisense, you'll have to type "`m_My` `<tab>`".

It's about keystroke economy, in my opinion.

Share Improve this answer

Follow

edited Mar 25, 2020 at 20:21



Dan

10.9k ● 11 ● 67 ● 106

answered Mar 18, 2009 at 19:08



Randolpho

56.4k ● 18 ● 151 ● 180

---



3



There is one important difference between C++ and C#: Tool support. When you follow the established guidelines (or common variations), you will get a deep level of tool support that C++ never had. Following the standards allows tools to do deeper refactoring/rename operations than you'd otherwise be capable of. Resharper does this. So stick with one of the established standards.

Share Improve this answer

answered Mar 18, 2009 at 18:47

Follow



[krosenvold](#)

77k ● 33 ● 156 ● 209



3



I never do this and the reason why is that I [try to] keep my methods short. If I can see the whole method on the screen, I can see the params, I can see the locals and so I can tell what is owned by the class and what is a param or a local.

I do typically name my params and locals using a particular notation, but not always. I'm nothing if not inconsistent. I rely on the fact that my methods are short and try to keep them from doing X, Y and Z when they should be only doing X.

Anyhow, that's my two cents.

Share Improve this answer

answered Mar 18, 2009 at 19:09

Follow



[itsmatt](#)

31.4k ● 11 ● 102 ● 165



3



Unless I'm stuck with vi or Emacs for editing code, my IDE takes care of differential display of members for me so I rarely uses any special conventions. That also goes for prefixing interfaces with I or classes with C.

Someone, please, explain the .NET style of I-prefix on interfaces. :)

Share Improve this answer

answered Mar 18, 2009 at 20:17

Follow



Magnus

129 ● 4

---

The I stands for...ready..."Interface" – [Ed Swangren](#) Mar 18, 2009 at 21:02

- 
- 1 Yes, and that is quite obvious. The real question is, why would anyone care if they're using interfaces, abstract classes or concrete classes? Does it matter to you if you accept an `ICustomerService` or a `CCustomerService` in the CTOR of your object? – [Magnus](#) Mar 19, 2009 at 6:56

---

It's just convention. Makes it easy to find the interface as well. I have tried removing it and I didn't really like it as some concrete classes were named the same `ICustomerService` and `CustomerService`, if you drop the I then they are the same. I don't want to have to think about naming it more cleverly, I just want to move on with the code. – [rball](#) Nov 20, 2009 at 23:51

---

For the reason behind the I-prefix, see (for example) [Interface naming convention on StackOverflow](#): which quotes Brad Abrams: "...the 'I' prefix on interfaces is a clear recognition of the influence of COM (and Java) on the .NET Framework" – [Wai Ha Lee](#) Jan 21, 2015 at 15:35

---

@Magnus "Does it matter to you if you accept an *ICustomerService* or a *CCustomerService* in the CTOR of your object?" Absolutely it does! In OOP you should "program to an interface, not an implementation." The "I" prefix easily lets you distinguish between interfaces and implementations, so you can ensure you're not accidentally specifying concrete parameters in your class constructors. – Tagc Aug 9, 2017 at 13:42 ✎

---



3



what i am used to is that private properties got small underscore f.ex "string \_name". the public one got "Name". and the input variables in methods got small letter "void MyMethod(string name)".



```
static const MYCONST = "hmpf" .
```



Share Improve this answer

edited Jun 28, 2018 at 21:58

Follow



Dan

10.9k ● 11 ● 67 ● 106

answered Mar 18, 2009 at 18:47



ThorHalvor

627 ● 5 ● 17

---



3



I am sure that I will get flamed for this but so be it.

It's called Microsoft's .NET library guidelines but it's really [Brad Abrams's](#) views ([document here](#)) - there are other views with valid reasons.



People tend to go with the majority view rather than having good solid reasons for a specific style.



The important point is to evaluate why a specific style is used and why it's preferred over another style - in other words, have a reason for choosing a style not just because everyone says it's the thing to do - think for yourself.

The basic reason for not using old style Hungarian was the use of abbreviations which was different for every team and difficult to learn - this is easily solved by not abbreviating.

As the available development tools change the style should change to what makes the most sense - but have a solid reason for each style item.

Below are my style guidelines with my reasons - I am always looking for ways to improve my style to create more reliable and easier to maintain code.

## Variable Naming Convention

We all have our view on variable naming conventions. There are many different styles that will help produce easily maintainable quality code - any style which supports the basic essential information about a variable are okay. The criteria for a specific naming convention should be that it aids in producing code that is reliable and easily maintainable. Criteria that should not be used are: It's ugly Microsoft (i.e. Brad Abrams) says don't use

that style - Microsoft does not always produce the most reliable code just look at the bugs in Expression Blend. It is very important when reading code that a variable name should instantly convey three essential facts about the variable: it's scope it's type a clearly understand about what it is used for Scope: Microsoft recommends relying totally on IntelliSense . IntelliSense is awesome; however, one simply does not mouse over every variable to see it's scope and type. Assuming a variable is in a scope that it is not can cause significant errors. For example, if a reference variable is passed in as a parameter and it is altered in local scope that change will remain after the method returns which may not be desired. If a field or a static variable is modified in local scope but one thinks that it is a local variable unexpected behavior could result. Therefore it is extremely important to be able to just look at a variable (not mouse over) and instantly know it's scope.

The following style for indicating scope is suggested; however, any style is perfectly okay as long as it clearly and consistently indicates the variable's scope: m\_ field variable p\_ parameter passed to a method s\_ static variable local variable Type: Serious errors can occur if one believes they are working with a specific type when they are actually working with a different type - again, we simply do not mouse over ever variable to determine its type, we just assume that we know what its type is and that is how errors are created.

Abbreviations: Abbreviations are evil because they can mean different things to different developers. One developer may think a leading lower case "s" means string while another may think it means signed integer. Abbreviations are a sign of lazy coding - take a little extra time and type the full name to make it clear to the developer that has to maintain the code. For example, the difference between "str" and "string" is only three characters - it does not take much more effort to make code easy to maintain.

Common and clear abbreviations for built-in data types only are acceptable but must be standardized within the team.

Self Documenting Code: Adding a clear description to a variable name makes it very easy for another developer to read and understand the code - make the name so understandable that the team manager can read and understand the code without being a developer.

Order of Variable Name Parts: The recommended order is scope-type-description because: IntelliSense will group all similar scopes and within each scope IntelliSense will group all similar types which makes lookups easy - try finding a variable the other way It makes it very easy to see and understand the scope and to see and understand the type It's a fairly common style and easy to understand It will pass FxCop

Examples: Here are a few examples:  
m\_stringCustomerName

p\_stringCustomerDatabaseConnectionString  
intNumberOfCustomerRecords or  
iNumberOfCustomerRecords or  
integerNumberOfCustomerRecords These simple rules  
will significantly improve code reliability and  
maintainability.

**Control Structure Single Line Statements** All control structures (if, while, for, etc.) single line statements should always be wrapped with braces because it is very easy to add a new statement not realizing that a given statement belongs to a control structure which will break the code logic without generating any compile time errors.

**Method Exception Wrapping** All methods should be wrapped with an outer try-catch which trap, provide a place to recover, identify, locate, log, and make a decision to throw or not. It is the unexpected exception that cause our applications to crash - by wrapping every method trapping all unhandled exceptions we guarantee identifying and logging all exceptions and we prevent our application from ever crashing. It takes a little more work but the results is well worth the effort.

**Indentation** Indentation is not a major issue; however, four spaces and not using tabs is suggested. If code is printed, the first printer tab usually defaults to 8 spaces. Different developer tend to use different tab sizes. Microsoft's code is usually indented 4 space so if one uses any Microsoft code and uses other than 4 spaces,



then the code will need to be reformatted. Four spaces makes it easy and consistent.

Share Improve this answer

edited Jul 12, 2019 at 16:16

Follow



DavidRR

19.3k ● 27 ● 111 ● 196

answered Mar 18, 2009 at 19:39



David Roh

- 
- 2 "People tend to go with the majority view rather than having good solid reasons for a specific style." In the absence of a good personal reason to do otherwise, following the majority consensus is *usually* a good bet. – [Chris](#) Mar 18, 2009 at 20:58
- 



2



I never use any hungarian warts whenever I'm given the choice. It's extra typing and doesn't convey any meaningful information. Any good IDE (and I define "good" based on the presence of this feature, among others) will allow you to have different syntax highlighting for static members, instance members, member functions, types, etc. There is no reason to clutter your code with information that can be provided by the IDE. This is a corollary to not cluttering your code with commented-out old code because your versioning system should be responsible for that stuff.

Share Improve this answer

answered Mar 18, 2009 at 18:49

Follow



rmeador



2



The best way is to agree on a standard with your colleagues, and stick to it. It doesn't absolutely have to be the method that would work best for everyone, just agreeing on one method is more important than which method you actually agree on.

What we chose for our code standard is to use `_` as prefix for member variables. One of the reasons was that it makes it easy to find the local variables in the intellisense.

Before we agreed on that standard I used another one. I didn't use any prefix at all, and wrote

`this.memberVariable` in the code to show that I was using a member variable.

With the property shorthand in C# 3, I find that I use a lot less explicit member variables.

Share Improve this answer

answered Mar 18, 2009 at 18:59

Follow



Guffa

700k ● 110 ● 750 ● 1k

One problem with the underscore prefix is that it makes variables that are not "CLS compliant" and you get a lot of compiler warnings. – JohnFx Mar 18, 2009 at 19:00

I'm surprised nobody mentioned the 'this.' approach before.  
– Trap Mar 19, 2009 at 12:48



2



The closest thing to official guidelines is [StyleCop](#), a tool from Microsoft which can automatically analyse your source files and detect violations from the recommended coding style, and can be run from within Visual Studio and/or automated builds such as MSBuild.

We use it on our projects and it does help to make code style and layout more consistent between developers, although be warned it does take *quite* a bit of getting used to!

To answer your question - it doesn't allow any Hungarian notation, nor any prefixes like `m_` (in fact, it doesn't allow the use of underscores at all).

Share Improve this answer

answered Mar 18, 2009 at 20:33

Follow



[Greg Beech](#)

136k ● 45 ● 209 ● 250

---

What does the tool say about a `m` prefix without the underscore, as in `mLocalVariable` ? (This is what we used in a C++ course back in college.) – [user7610](#) Jul 3, 2015 at 22:19

---



2



I don't use that style any longer. It was developed to help you see quickly how variables were being used. The newer dev environments let you see that information by hovering your mouse over the variable. The need for it has gone away if you use those newer tools.



Share Improve this answer

answered Mar 18, 2009 at 22:18



Follow



Jay

14.4k ● 5 ● 46 ● 74



2



There might also be some insight to be gleaned from C++ *Coding Standards* (**Sutter, Herb** and **Alexandrescu Andrei**, 2004). Item #0 is entitled "Don't sweat the small stuff. (Or: Know what not to standardize.)".



They touch on this specific question a little bit by saying "If you can't decide on your own naming convention, try ... private member variables **likeThis\_** ..." (Remember use of leading underscore is subject to very specific rules in C++).

However, before getting there, they emphasize a certain level of consistency "...the important thing is not to set a rule but just to be consistent with the style already in use within the file..."

Share Improve this answer

answered Mar 19, 2009 at 14:12

Follow



Dan

10.9k ● 11 ● 67 ● 106



2



The benefit of that notation in C/C++ was to make it easier to see what a symbol's type was without having to go search for the declaration. These styles appeared before the arrival of Intellisense and "Go to Definition" - we often had to go on a goose chase looking for the declaration in who knows how many header files. On a



large project this could be a significant annoyance which was bad enough when looking at C source code, but even worse when doing forensics using mixed assembly+source code and a raw call stack.

When faced with these realities, using `m_` and all the other hungarian rules starts to make some sense even with the maintenance overhead because of how much time it would save just in looking up a symbol's type when looking at unfamiliar code. Now of course we have Intellisense and "Go to Definition", so the main time saving motivation of that naming convention is no longer there. I don't think there's much point in doing that any more, and I generally try to go with the .NET library guidelines just to be consistent and possibly gain a little bit more tool support.

Share Improve this answer

answered Mar 20, 2009 at 14:44

Follow



[Eric Cosky](#)

574 ● 3 ● 11

- 
- 1 `m_` has nothing to do with type and everything to do with lifetime. Some editors use color coding to differentiate locals from class members, but Intellisense doesn't actually give much help in this regard. – [Ben Voigt](#) May 3, 2010 at 23:41
- 



1

If you are not coding under a particular guideline, you should keep using your actual `m_` notation and change it if the project coding guidelines says so.



Share Improve this answer

Follow



edited Mar 25, 2020 at 20:21



Dan

10.9k ● 11 ● 67 ● 106

answered Mar 18, 2009 at 19:34



jose Ernest Davila

397 ● 1 ● 5 ● 15



0



Be functional.

- Do not use global variables.
- Do not use static variables.
- Do not use member variables.



If you really have to, but only if you really have to, use one and only one variable to access your application / environment.

Share Improve this answer

Follow

answered Mar 18, 2009 at 23:02



Mark Stock

1,733 ● 2 ● 13 ● 23