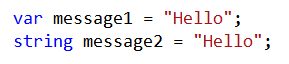
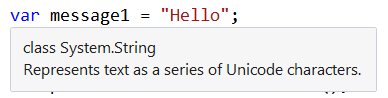
**Demystifying the "var" Keyword in C#**

There are some misconceptions about the "var" keyword in C#. The "var" keyword was added to the C# language to solve a specific problem, but developers use it in other parts of their code as well (and there's nothing wrong with that). We'll take a look at what "var" is (and what it isn't), look at the reason it was created, and then I'll show where I use it and my reasoning behind that.  
  
This whole discussion has been spurred by some questions that I've received from a few developers recently:  
  
***Does using "var" affect performance?***  
***Does "var" indicate a variant type that we have in other languages?***  
  
Let's answer these questions by looking at how "var" works.  
  
**"var" Is Just a Shortcut**  
***(The way that most developers use it)***  
Where we most often see "var", it is just a shortcut to writing out the type for a variable. The idea is that the compiler can figure out the actual type based on what we are assigning to the variable, so we don't have to be explicit about the type. Let's start with a simple example:

[](https://4.bp.blogspot.com/-ZhmP5eyHNjw/UwfI01W0U9I/AAAAAAAABSE/1G19DrLA53M/s1600/Code_strings.png)

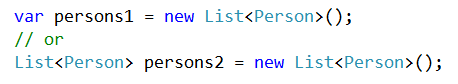
***These lines of code are 100% equivalent from a technical perspective.*** (We'll talk about the readability perspective a bit later.) Both the "message1" and "message2" variables of are type "string". And they are both strongly typed.  
  
We can see that "message1" is strongly typed by putting our mouse over the "var" keyword:

[](https://1.bp.blogspot.com/-JKhe4NcYbYY/UwfI0f3YCAI/AAAAAAAABRk/IX-dlgsvpns/s1600/Code_string_popup.png)

This clearly shows that "message1" is a string. Since the compiler can figure out the type based on the assignment, we can use "var" as a shortcut and let the compiler fill in the rest.  
  
***The variable is strongly typed.*** This is not a variant or dynamic variable that we might see in other languages. When "message1" is declared and initialized, it is a string, and it will always be a string. If we try to assign an integer to "message1" later in our code, we will get a compiler error.  
  
***There is no performance difference between these two lines of code.*** We can look at the IL (the intermediate language produced by the compiler) to verify this:

[https://3.bp.blogspot.com/-NLXzhVjdeHk/UwfI1vPYBEI/AAAAAAAABSA/5WberWEKMc8/s1600/IL_strings.png](https://3.bp.blogspot.com/-NLXzhVjdeHk/UwfI1vPYBEI/AAAAAAAABSA/5WberWEKMc8/s1600/IL_strings.png)

This shows the local variables for a particular method. We can see that "var message1" and "string message2" compile down to exactly the same code. This means that there is no performance difference between these 2 statements. All of the work is done at compile-time.  
  
Let's look at an example with a more complex type:

[](https://2.bp.blogspot.com/-B39NK0x-1Gg/UwfIzcuPA0I/AAAAAAAABRo/QpP9WherSwE/s1600/Code_lists1.png)

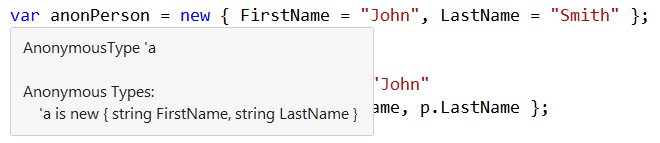
As above, these two lines of code are equivalent. both "persons1" and "persons2" are of type "List<Person>". If we look at the IL that is generated, we see that both variables have the same type:

[https://2.bp.blogspot.com/-TzmAxMd8jSY/UwfI1HJSQcI/AAAAAAAABR8/W2Yc-UuVlqE/s1600/IL_lists.png](https://2.bp.blogspot.com/-TzmAxMd8jSY/UwfI1HJSQcI/AAAAAAAABR8/W2Yc-UuVlqE/s1600/IL_lists.png)

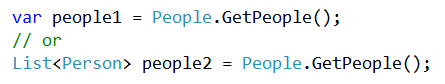
This type looks a little strange because it has a generic parameter and the assembly and namespaces are listed. But this just means "List<Person>".  
  
**Why "var" Exists**  
So, we've seen how most developers use "var" -- it's a way for us to create variables without having to write out the specific type. But, this isn't a good reason for adding a new keyword to a language; "var" was actually added for a completely different purpose.  
  
With C# 3.0 (.NET 3.5), we got LINQ, Language INtegrated Query, and it's one of the most awesome things that's been added to C#. One thing that was added to make LINQ work well was the concept of anonymous types.  
  
An anonymous type is a new type that is created "on-the-fly". And just like an anonymous delegate, an anonymous type does not have a name (well, it actually does, we just can't see it). Here's an example:

[https://1.bp.blogspot.com/-h6aXIW7KAc8/UwfIykGbdcI/AAAAAAAABRE/Of06myIqnJI/s1600/Code_anonymous_type.png](https://1.bp.blogspot.com/-h6aXIW7KAc8/UwfIykGbdcI/AAAAAAAABRE/Of06myIqnJI/s1600/Code_anonymous_type.png)

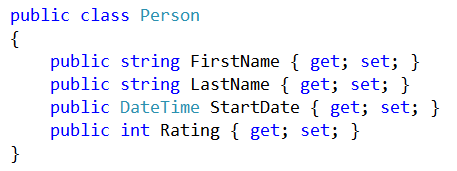
This declares a variable called "anonPerson" and assigns an anonymous type to it. We create an anonymous type by using the "new" keyword and then supplying property names and values inside curly braces. This example creates a type with 2 properties: FirstName and LastName.  
  
This is a situation where we **must** use "var". Why? Well, think about the type of the "anonPerson" variable. What is it? By definition, it is an anonymous type which doesn't have a name, so we really have no way to refer to it. So instead, we use "var" and let the compiler deal with it. We can see how the compiler deals with it by hovering the mouse over the "var" keyword:

[](https://3.bp.blogspot.com/-17eLpk5mk0E/UwfIy4WJXcI/AAAAAAAABRI/4fOITLqalkw/s1600/Code_anonymous_type_popup.png)

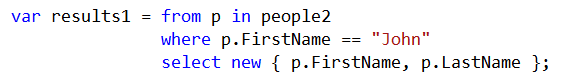
This shows us that "anonPerson" is an anonymous type, which has been designated 'a. That type consists of 2 properties: FirstName and LastName. The compiler has figured out these are both strings because we assigned strings to the properties.  
  
Now, most of the time, we don't go around creating anonymous types. But, if we use projections with LINQ, we can easily end up with anonymous types. Let's take a look at some more code. First, we have some variables:

[](https://3.bp.blogspot.com/-9Vh38qMkuzY/UwfIz_wO6YI/AAAAAAAABRs/nTtUwBp9Eq0/s1600/Code_lists2.png)

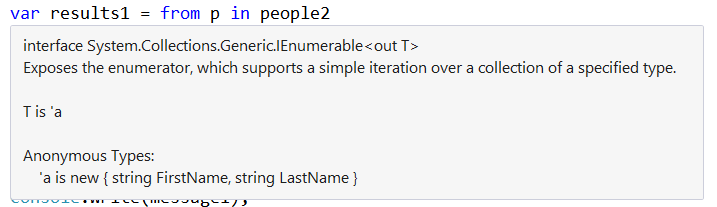
Again, these 2 lines of code are equivalent. The "People.GetPeople()" method returns a populated List consisting of 7 Person objects. The "Person" type is pretty simple; just 4 properties:

[](https://3.bp.blogspot.com/-jt-uBFM9j8Q/UwfIyRR_BoI/AAAAAAAABRQ/d7t9ZcUjCWc/s1600/Code_Person.png)

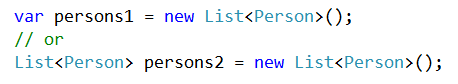
Next, let's query one of our variables using LINQ:

[](https://2.bp.blogspot.com/-kdU7eOmpr-o/UwfIx67lg-I/AAAAAAAABQ0/qmiDh-nQoa4/s1600/Code_LINQ_query.png)

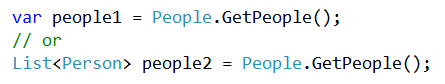
This will query the "people2" variable and find all of the items where the "FirstName" property is "John". Then in the "select" portion, it creates a new anonymous type based on 2 of the properties of the "Person" class. This is known as a projection, where we take the values of one type (or multiple types) and map them to a new type.  
  
If we hover the mouse over the "var", we can see the actual type:

[](https://3.bp.blogspot.com/-4ucCCj8WsWg/UwfIx1dR6NI/AAAAAAAABQw/TtFIc6PGWiU/s1600/Code_LINQ_query_popup.png)

We get an "IEnumerable<T>" back from this query (which is pretty normal). And if we look, we see that the "T" is actually an anonymous type. This properties for this type happen to match the anonymous type that we created manually above: FirstName and LastName.  
  
So, we can see that when we do projections in LINQ that result in anonymous types, we **must** use the "var" keyword since the anonymous type does not have a name.  
  
**Restrictions to Using "var"**  
Because of how "var" is implemented, there are a few restrictions on where we can use "var". These will result in compiler errors if we don't adhere to them.  
  
***First, we can only use "var" in a local scope where the variable is declared and initialized in a single statement.*** This makes sense if we think about it. Since the variable is strongly typed, the compiler needs to know what type it is when it is declared. It figured out the type by looking at the value that is being assigned. If the variable is not immediately initialized, then the compiler cannot determine the type.  
  
***Second, "var" cannot be used for class-level fields.*** This is something that people often try to do. "var" is pretty much limited to variables that are locally scoped to a method.  
  
There are the restrictions that most people run into problems with. There are a few more restrictions, but we don't run across them as often. For more information, check out the MSDN article on [Implicitly Typed Local Variables](http://msdn.microsoft.com/en-us/library/bb384061.aspx).  
  
**How I Use "var"**  
What we've seen is that "var" compiles to exactly the same IL as if we are explicit with our types. And we've also seen that there is no performance difference when we use "var". So, the question becomes, when should we use "var" and when should we be explicit?  
  
*Note: We're talking about the non-required use of "var" in declaring variables like in our first example. We've already seen that we must use "var" with anonymous types.*  
  
There are many schools of thought on this. Since there is no technical difference, it really comes down to a matter of preference. I've heard of some developers who say that we should use "var" wherever we can (meaning, wherever the restrictions allow us to use "var"). The thinking behind this is that if the types need to change in the future, then the code that declares the variables does not need to be updated. I understand the thinking behind this, but I don't necessarily agree with it.  
  
***Focus on Readability***  
Whenever I use "var", I think about the readability of my code. I'm a big proponent of making code that is easy to approach, follow, maintain, and debug (you can check out my materials on [Clean Code](http://www.jeremybytes.com/Demos.aspx#CC) if you need convincing of that).  
  
So, let's go back to our examples from above, and I'll tell you where I do use "var" and where I don't.

[](https://2.bp.blogspot.com/-B39NK0x-1Gg/UwfIzcuPA0I/AAAAAAAABRo/QpP9WherSwE/s1600/Code_lists1.png)

For this example, I **do** use "var". For the "person1" variable, I can easily tell the type because it is stated in the assignment. This is a "List<Person>" variable.  
  
In my view, "person2" is a bit harder to read (this is my subjective viewpoint, of course). Since "List<Person>" is repeated on both sides of the assignment operator, it just gets in the way of me picking out the important bits of this statement. This is even more pronounced when we have longer types, like "Dictionary<int, Person>".  
  
Here's another example:

[](https://3.bp.blogspot.com/-9Vh38qMkuzY/UwfIz_wO6YI/AAAAAAAABRs/nTtUwBp9Eq0/s1600/Code_lists2.png)

In this situation, I do **not** use "var". For the "people1" variable, it is not immediately clear what the type is. I need to know the return type of the "GetPeople()" method. It is easy to get that value, by hovering the mouse over either "GetPeople" or "var", but that's an extra step.  
  
For the "people2" variable, I don't have to take any additional steps. I can see that the type is "List<Person>" right in the code.  
  
These are just my views on the subject. If you are in a team environment, it is more important that everyone on the team takes the same approach. If we all do the same thing, then other developers know what to expect when they open up our code.  
  
**Wrap Up**  
I use "var" pretty unconsciously at this point. And that's probably why I get asked about it so much. When we're new to "var", we might think that it creates a variant or dynamic variable. Or we might think that there is some technical reason why we should or should not use it.  
  
But the reality is that "var" does not have any effect on the compiled output. The variables are still strongly typed, and the IL that is generated is exactly the same as if we put in the explicit type. There are a few places where we **must** use "var" (such as with anonymous types), but most often "var" is simply used as a shortcut. Since the compiler can figure out what the actual type is, we don't have to put that in.  
  
The primary thing we need to keep in mind when we're writing code is that someone else will have to read it. So, we want to make sure that when we do use these types of shortcuts, we don't diminish the readability of our code.  
  
Happy Coding!

Posted by [Jeremy](https://www.blogger.com/profile/06749690234470413216)at [3:08 PM](https://jeremybytes.blogspot.com/2014/02/demystifying-var-keyword-in-c.html) [[https://resources.blogblog.com/img/icon18_email.gif](https://www.blogger.com/email-post.g?blogID=5359546512544809971&postID=6428632910810442547)](https://www.blogger.com/email-post.g?blogID=5359546512544809971&postID=6428632910810442547)