**Implicitly Typed Local Variables (C# Programming Guide)**

Updated: July 2008

Local variables can be given an inferred "type" of **var** instead of an explicit type. The **var** keyword instructs the compiler to infer the type of the variable from the expression on the right side of the initialization statement. The inferred type may be a built-in type, an anonymous type, a user-defined type, or a type defined in the .NET Framework class library. For more information about how to initialize arrays with **var**, see [Implicitly Typed Arrays (C# Programming Guide)](http://msdn.microsoft.com/en-us/library/bb384090.aspx).

The following examples show various ways in which local variables can be declared with **var**:

C#

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl02CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl02CSharp');)

// i is compiled as an int

var i = 5;

// s is compiled as a string

var s = "Hello";

// a is compiled as int[]

var a = new[] { 0, 1, 2 };

// expr is compiled as IEnumerable<Customer>

// or perhaps IQueryable<Customer>

var expr =

from c in customers

where c.City == "London"

select c;

// anon is compiled as an anonymous type

var anon = new { Name = "Terry", Age = 34 };

// list is compiled as List<int>

var list = new List<int>();

It is important to understand that the **var** keyword does not mean “variant” and does not indicate that the variable is loosely typed, or late-bound. It just means that the compiler determines and assigns the most appropriate type.

The **var** keyword may be used in the following contexts:

* On local variables (variables declared at method scope) as shown in the previous example.
* In a [for](http://msdn.microsoft.com/en-us/library/ch45axte.aspx) initialization statement.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl04other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl04other');)

for(var x = 1; x < 10; x++)

* In a [foreach](http://msdn.microsoft.com/en-us/library/ttw7t8t6.aspx) initialization statement.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl06other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl06other');)

foreach(var item in list){...}

* In a [using Statement](http://msdn.microsoft.com/en-us/library/yh598w02.aspx)

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl08other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl08other');)

using (var file = new StreamReader("C:\\myfile.txt")) {...}

For more information, see [How to: Use Implicitly Typed Local Variables and Arrays in a Query Expression (C# Programming Guide)](http://msdn.microsoft.com/en-us/library/bb310799.aspx).

http://i.msdn.microsoft.com/Global/Images/clear.gif var and Anonymous Types

In many cases the use of **var** is optional and is just a syntactic convenience. However, when a variable is initialized with an anonymous type you must declare the variable as **var** if you need to access the properties of the object at a later point. This is a common scenario in LINQ query expressions. For more information, see [Anonymous Types (C# Programming Guide)](http://msdn.microsoft.com/en-us/library/bb397696.aspx).

From the perspective of your source code, an anonymous type has no name. Therefore, if a query variable has been initialized with **var**, then the only way to access the properties in the returned sequence of objects is to use **var** as the type of the iteration variable in the **foreach** statement.

C#

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl15CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl15CSharp');)

class ImplicitlyTypedLocals2

{

static void Main()

{

string[] words = { "aPPLE", "BlUeBeRrY", "cHeRry" };

// If a query produces a sequence of anonymous types,

// then use var in the foreach statement to access the properties.

var upperLowerWords =

from w in words

select new { Upper = w.ToUpper(), Lower = w.ToLower() };

// Execute the query

foreach (var ul in upperLowerWords)

{

Console.WriteLine("Uppercase: {0}, Lowercase: {1}", ul.Upper, ul.Lower);

}

}

}

/\* Outputs:

Uppercase: APPLE, Lowercase: apple

Uppercase: BLUEBERRY, Lowercase: blueberry

Uppercase: CHERRY, Lowercase: cherry

\*/

http://i.msdn.microsoft.com/Global/Images/clear.gif Remarks

The following restrictions apply to implicitly-typed variable declarations:

* **var** can only be used when a local variable is declared and initialized in the same statement; the variable cannot be initialized to null, or to a method group or an anonymous function.
* **var** cannot be used on fields at class scope.
* Variables declared by using **var** cannot be used in the initialization expression. In other words, this expression is legal: int i = (i = 20); but this expression produces a compile-time error: var i = (i = 20);
* Multiple implicitly-typed variables cannot be initialized in the same statement.
* If a type named var is in scope, then the **var** keyword will resolve to that type name and will not be treated as part of an implicitly typed local variable declaration.

You may find that **var** can also be useful with query expressions in which the exact constructed type of the query variable is difficult to determine. This can occur with grouping and ordering operations.

The **var** keyword can also be useful when the specific type of the variable is tedious to type on the keyboard, or is obvious, or does not add to the readability of the code. One example where **var** is helpful in this manner is with nested generic types such as those used with group operations. In the following query, the type of the query variable is IEnumerable<IGrouping<string, Student>>. As long as you and others who must maintain your code understand this, there is no problem with using implicit typing for convenience and brevity.

C#

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl19CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl19CSharp');)

// Same as previous example except we use the entire last name as a key.

// Query variable is an IEnumerable<IGrouping<string, Student>>

var studentQuery3 =

from student in students

group student by student.Last;

However, the use of **var** does have at least the potential to make your code more difficult to understand for other developers. For that reason, the C# documentation generally uses **var** only when it is required.

**Implicitly Typed Arrays (C# Programming Guide)**

You can create an implicitly-typed array in which the type of the array instance is inferred from the elements specified in the array initializer. The rules for any implicitly-typed variable also apply to implicitly-typed arrays. For more information, see [Implicitly Typed Local Variables (C# Programming Guide)](http://msdn.microsoft.com/en-us/library/bb384061.aspx).

Implicitly-typed arrays are usually used in query expressions together with anonymous types and object and collection initializers.

The following examples show how to create an implicitly-typed array:

C#

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl02CSharp');)

class ImplicitlyTypedArraySample

{

static void Main()

{

var a = new[] { 1, 10, 100, 1000 }; // int[]

var b = new[] { "hello", null, "world" }; // string[]

// single-dimension jagged array

var c = new[]

{

new[]{1,2,3,4},

new[]{5,6,7,8}

};

// jagged array of strings

var d = new[]

{

new[]{"Luca", "Mads", "Luke", "Dinesh"},

new[]{"Karen", "Suma", "Frances"}

};

}

}

In the previous example, notice that with implicitly-typed arrays, no square brackets are used on the left side of the initialization statement. Note also that jagged arrays are initialized by using **new []** just like single-dimension arrays. Multidimensional implicitly-typed arrays are not supported.

 Implicitly-typed Arrays in Object Initializers

When you create an anonymous type that contains an array, the array must be implicitly typed in the type's object initializer. In the following example, contacts is an implicitly-typed array of anonymous types, each of which contains an array named PhoneNumbers. Note that the **var** keyword is not used inside the object initializers.

C#

[Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl06CSharp');)

var contacts = new[]

{

new {

Name = " Eugene Zabokritski",

PhoneNumbers = new[] { "206-555-0108", "425-555-0001" }

},

new {

Name = " Hanying Feng",

PhoneNumbers = new[] { "650-555-0199" }

}

};