**yield (C# Reference)**

**Visual Studio 2013**

[Other Versions](javascript:;)

Description: http://i.msdn.microsoft.com/Areas/Epx/Content/Images/ImageSprite.png

* [Visual Studio 2012](http://msdn.microsoft.com/en-us/library/9k7k7cf0(d=printer,v=vs.110).aspx)
* [Visual Studio 2010](http://msdn.microsoft.com/en-us/library/9k7k7cf0(d=printer,v=vs.100).aspx)
* [Visual Studio 2008](http://msdn.microsoft.com/en-us/library/9k7k7cf0(d=printer,v=vs.90).aspx)
* [Visual Studio 2005](http://msdn.microsoft.com/en-us/library/9k7k7cf0(d=printer,v=vs.80).aspx)

When you use the yield keyword in a statement, you indicate that the method, operator, or get accessor in which it appears is an iterator. Using yield to define an iterator removes the need for an explicit extra class (the class that holds the state for an enumeration, see [IEnumerator<T>](http://msdn.microsoft.com/en-us/library/78dfe2yb.aspx) for an example) when you implement the [IEnumerable](http://msdn.microsoft.com/en-us/library/system.collections.ienumerable.aspx) and [IEnumerator](http://msdn.microsoft.com/en-us/library/system.collections.ienumerator.aspx) pattern for a custom collection type.

The following example shows the two forms of the yield statement.

yield return <expression>;

yield break;

[Remarks](javascript:void(0))

You use a yield return statement to return each element one at a time.

You consume an iterator method by using a [foreach](http://msdn.microsoft.com/en-us/library/ttw7t8t6.aspx) statement or LINQ query. Each iteration of the foreach loop calls the iterator method. When a yield return statement is reached in the iterator method, expression is returned, and the current location in code is retained. Execution is restarted from that location the next time that the iterator function is called.

You can use a yield break statement to end the iteration.

For more information about iterators, see [Iterators (C# and Visual Basic)](http://msdn.microsoft.com/en-us/library/dscyy5s0.aspx).

**Iterator Methods and get Accessors**

The declaration of an iterator must meet the following requirements:

* The return type must be [IEnumerable](http://msdn.microsoft.com/en-us/library/system.collections.ienumerable.aspx), [IEnumerable<T>](http://msdn.microsoft.com/en-us/library/9eekhta0.aspx), [IEnumerator](http://msdn.microsoft.com/en-us/library/system.collections.ienumerator.aspx), or [IEnumerator<T>](http://msdn.microsoft.com/en-us/library/78dfe2yb.aspx).
* The declaration can't have any [ref](http://msdn.microsoft.com/en-us/library/14akc2c7.aspx) or [out](http://msdn.microsoft.com/en-us/library/t3c3bfhx.aspx) parameters.

The yield type of an iterator that returns [IEnumerable](http://msdn.microsoft.com/en-us/library/system.collections.ienumerable.aspx) or [IEnumerator](http://msdn.microsoft.com/en-us/library/system.collections.ienumerator.aspx) is object. If the iterator returns [IEnumerable<T>](http://msdn.microsoft.com/en-us/library/9eekhta0.aspx) or [IEnumerator<T>](http://msdn.microsoft.com/en-us/library/78dfe2yb.aspx), there must be an implicit conversion from the type of the expression in the yield return statement to the generic type parameter .

You can't include a yield return or yield break statement in methods that have the following characteristics:

* Anonymous methods. For more information, see [Anonymous Methods (C# Programming Guide)](http://msdn.microsoft.com/en-us/library/0yw3tz5k.aspx).
* Methods that contain unsafe blocks. For more information, see [unsafe (C# Reference)](http://msdn.microsoft.com/en-us/library/chfa2zb8.aspx).

**Exception Handling**

A yield return statement can't be located in a try-catch block. A yield return statement can be located in the try block of a try-finally statement.

A yield break statement can be located in a try block or a catch block but not a finally block.

If the foreach body (outside of the iterator method) throws an exception, a finally block in the iterator method is executed.

**Technical Implementation**

The following code returns an **IEnumerable<string>** from an iterator method and then iterates through its elements.

C#

IEnumerable<string> elements = MyIteratorMethod();

foreach (string element in elements)

{

…

}

The call to MyIteratorMethod doesn't execute the body of the method. Instead the call returns an **IEnumerable<string>** into the elements variable.

On an iteration of the foreach loop, the [MoveNext](http://msdn.microsoft.com/en-us/library/system.collections.ienumerator.movenext.aspx) method is called for elements. This call executes the body of MyIteratorMethod until the next yield return statement is reached. The expression returned by the yield return statement determines not only the value of the element variable for consumption by the loop body but also the [Current](http://msdn.microsoft.com/en-us/library/bb778561.aspx) property of elements, which is an **IEnumerable<string>**.

On each subsequent iteration of the foreach loop, the execution of the iterator body continues from where it left off, again stopping when it reaches a yield return statement. The foreach loop completes when the end of the iterator method or a yield break statement is reached.

[Example](javascript:void(0))

The following example has a yield return statement that's inside a for loop. Each iteration of the foreach statement body in Process creates a call to the Power iterator function. Each call to the iterator function proceeds to the next execution of the yield return statement, which occurs during the next iteration of the for loop.

The return type of the iterator method is [IEnumerable](http://msdn.microsoft.com/en-us/library/system.collections.ienumerable.aspx), which is an iterator interface type. When the iterator method is called, it returns an enumerable object that contains the powers of a number.

C#

using System;

using System.Collections.Generic;

public class PowersOf2

{

static void Main()

{

// Display powers of 2 up to the exponent of 8:

foreach (int i in Power(2, 8))

{

Console.Write("{0} ", i);

}

}

public static System.Collections.Generic.IEnumerable<int> Power(int number, int exponent)

{

int result = 1;

for (int i = 0; i < exponent; i++)

{

result = result \* number;

yield return result;

}

}

// Output: 2 4 8 16 32 64 128 256

}

The following example demonstrates a get accessor that is an iterator. In the example, each yield return statement returns an instance of a user-defined class.

C#

using System;

public static class GalaxyClass

{

public static void ShowGalaxies()

{

var theGalaxies = new Galaxies();

foreach (Galaxy theGalaxy in theGalaxies.NextGalaxy)

{

Console.WriteLine(theGalaxy.Name + " " + theGalaxy.MegaLightYears.ToString());

}

}

public class Galaxies

{

public System.Collections.Generic.IEnumerable<Galaxy> NextGalaxy

{

get

{

yield return new Galaxy { Name = "Tadpole", MegaLightYears = 400 };

yield return new Galaxy { Name = "Pinwheel", MegaLightYears = 25 };

yield return new Galaxy { Name = "Milky Way", MegaLightYears = 0 };

yield return new Galaxy { Name = "Andromeda", MegaLightYears = 3 };

}

}

}

public class Galaxy

{

public string Name { get; set; }

public int MegaLightYears { get; set; }

}

static void Main()

{

ShowGalaxies();

Console.ReadKey();

}

}