

not actually [to establish a blogging point where individuals can enrich their learns on facilitating and leveraging .NET-related activities most effectively](#)

The Old New Thing

The implementation of iterators in C# and its consequences (part 3)

I mentioned that [there was an exception](#) to the general statement that the conversion of an iterator into traditional C# code is something you could have done yourself. That's true, and it was also a pun, because the exception is exception handling.

If you have a `try ... finally` block in your iterator, the language executes the `finally` block under the following conditions:

- After the last statement of the `try` block is executed. (No surprise here.)
- When an exception propagates out of the `try` block. (No surprise here either.)
- When execution leaves the `try` block via `yield break`.
- When the iterator is `Disposed` and the iterator body was trapped inside a `try` block at the time.

That last case can occur if somebody decides to abandon the enumerator before it is finished.

```
IEnumerable<int> CountTo10()
{
    try {
        for (int i = 1; i <= 10; i++) {
            yield return i;
        }
    } finally {
        System.Console.WriteLine("finally");
    }
}

foreach (int i in CountTo10()) {
    System.Console.WriteLine(i);
    if (i == 5) break;
}
```

This code fragment prints "1 2 3 4 5 finally".

If you think about it, this behavior is completely natural. You want the `finally` block to execute when the `try` block is finished executing, either by normal or abnormal means. Although control leaves the `try` block during the `yield return`, it comes back when the caller asks for the next item from the enumerator, so execution of the `try` block isn't finished yet. The `try` is finished executing after the last statement completes, an exception is thrown past it, or execution is abandoned when the enumerator is prematurely destroyed.

And this is exactly what you want when you use the `finally` block to clean up resources used by the `try` block.

Now, technically, you *can* write this yourself without using iterators, but it's pretty ugly. You'll need more internal state variables to keep track of whether the `try` block is still active and whether the exit of the `try` block is temporary (due to `yield return`) or permanent. It's a real pain in the neck, however, so you probably are better off letting the compiler do the work for you.

Published Thursday, August 14, 2008 7:00 AM by [oldnewthing](#)

Filed under: [Code](#)

Comments

re: The implementation of iterators in C# and its consequences (part 3)

Thursday, August 14, 2008 10:14 AM by Professor Farnsworth

Good news everyone! CLR week is almost over!

re: The implementation of iterators in C# and its consequences (part 3)

Thursday, August 14, 2008 11:10 AM by Bob

I want more CLR week! GIME GIME GIME. :)

re: The implementation of iterators in C# and its consequences (part 3)

Thursday, August 14, 2008 12:29 PM by [Eric Lippert](#)

It is, as you note, a hard problem, so hard that we on the compiler team have gotten it wrong numerous times. I apologize for the inconvenience.

There are scenarios where you can have multiple nested try blocks and the released v3 compiler generates code that runs the finally blocks outside-to-inside instead of the correct order.

I believe we have fixed those bugs for the v3 service release, but we have found more since then.

(So far the only remaining bugs I know of involve bizarre cases of, say, a yield break which branches out to a finally which then executes a second yield break. The control flow can get messed up in complicated circumstances like that.)

If anyone finds bugs involving iterators blocks and finally execution, please email me via my blog.

re: The implementation of iterators in C# and its consequences (part 3)

Thursday, August 14, 2008 2:17 PM by Daniel Plaisted

Was support for try blocks with yield statements inside them added in VS2008? I tried to do this in VS2005 and the compiler told me that it wasn't supported, so I had to work around it. It was kind of a pain.

re: The implementation of iterators in C# and its consequences (part 3)

Thursday, August 14, 2008 2:23 PM by Daniel Plaisted

Regarding my previous comment, it looks like the limitation I was remembering was that you cannot yield a value in the body of a try block with a catch clause. You also can't yield a value within a catch clause.

I assume there are good reasons for this but for my situation it would have been nice if it worked :)

Today's Blogs & Roman's Blog

Thursday, August 14, 2008 4:11 PM by [Today's Blogs & Roman's Blog](#)

PingBack from <http://rhnatiuk.wordpress.com/2008/08/14/todays-blogs/>

re: The implementation of iterators in C# and its consequences (part 3)

Thursday, August 14, 2008 5:28 PM by Jack Mathews

If you don't Dispose the iterator, will the finalizer cause the code in the finally blocks to get executed?

Visual Studio Links #66

Monday, August 18, 2008 7:55 AM by [Visual Studio Hacks](#)

My latest in a series of the weekly, or more often, summary of interesting links I come across related to Visual Studio. Greg Duncan posted a link to the release announcement for Task Board for Team System Beta 2 . Raymond Chen discussed the implementation

Interesting Links - 8/21/2008

Thursday, August 21, 2008 3:21 PM by [Matt Johnson's Technical Adventures](#)

The Old New Thing : The implementation of iterators in C# and its consequences (part 3)
Notes from a

C# Yield Keyword & Vasu Balakrishnan's Blog

Wednesday, May 20, 2009 3:36 PM by [C# Yield Keyword & Vasu Balakrishnan's Blog](#)

PingBack from <http://vasubalakrishnan.wordpress.com/2009/05/20/c-yield-keyword/>

New Comments to this post are disabled

© 2009 Microsoft Corporation. All rights reserved. [Terms of Use](#) | [Trademarks](#) | [Privacy Statement](#)