

# Re: Function minimization and random numbers

---

*Source:*

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2008-02/msg00724.html>

---

- *From:* Jon Skeet [C# MVP] <[skeet@xxxxxxxx](mailto:skeet@xxxxxxxx)>
  - *Date:* Wed, 6 Feb 2008 23:12:53 -0000
- 

Jon Harrop <[usenet@xxxxxxxxxxxxxxxx](mailto:usenet@xxxxxxxxxxxxxxxx)> wrote:

It seems to me like *\*you're\** making the artificial distinction here.

C#'s type system makes the distinction, not me.

Well, .NET's type system makes the distinction, allowing conversions between them.

However, that seems irrelevant in terms of the question of whether C# lacks closures. You claim that it does, and I'll ask again: please provide some widely accepted definition of "closure" which C# doesn't satisfy.

That is why you must convert a member into a delegate manually in C# when, in fact, they are both just functions. There's a whole MSDN help page devoted to this no-op.

What exactly do you mean by "manually"? Do you mean there has to be an appropriate delegate type to convert to? That's certainly true. I don't see the issue though – unless you want to do away with static type safety, there has to be something available to encapsulate the signature of the function.

Now, you could well want to do away with the static type safety – Groovy supports overloaded method groups as closures, where the appropriate method to actually call is determined at execution time, for example. There are advantages and disadvantages to that – but I haven't seen anything in a definition of closures which *\*requires\** a dynamic interface.

You can build closures in C# just fine according to every definition of closure I've seen.

## Re: Function minimization and random numbers

That's not to say that you can use every member \*as\* a closure, but to me that's not the same thing at all.

Then you are retrofitting language features onto Algol with no regard for what makes them useful.

Yeah, right. Of course. Anonymous methods and lambda expressions were added with no thought whatsoever. Get real.

Provided you just want a list of checkboxes of language features, that's fine, but if you want the features to be useful then you're going to have to think a lot more about how they interact.

Are you claiming that the C# support for closures is useless? That would rule out most of LINQ to start with...

Claiming that C# lacks closures is pure FUD in my view.

Avoid closures if you want to call your F# libraries from C#.

So has F# eschewed the standard .NET use of delegates as effectively function pointers? If F# really doesn't allow the use of what \*the whole of the rest of .NET\* treats as functions to be treated as functions, that sounds like F#'s fault rather than C#'s. However, I seem to remember that actually you \*can\* use delegates in F# perfectly easily. I'd check with Robert's book, but it's at work at the moment.

To provide first-class closures, .NET (or C#) might replace all members with properties that return delegates. Then you could say "C# supports first-class closures".

No, then you could say "C# supports first-class closures in the same way that F# does".

That is not how F# supports closures.

Consider a language that support integers but imposes a compile time distinction between even and odd integers for no logical reason. If that

Re: Function minimization and random numbers

was the only language you knew, would you back it to the hilt because, technically, you can work around the type system to use all integers?

Perhaps you'd deign to explain to us mere peasants just why F#'s support of closures is so wonderful as to make C#'s support appear to be useless and to even count as a lack of support in your (previously stated) view.

--

Jon Skeet - <skeet@xxxxxxxx>

<http://www.pobox.com/~skeet> Blog: <http://www.ms MVPs.com/jon.skeet>

World class .NET training in the UK: <http://iterativetraining.co.uk>

.