

Re: Compiler Error CS0702

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2008-04/msg02574.html>

- *From:* "Peter Duniho" <NpOeStPeAdM@xxxxxxxxxxxxxxxxxxxx>
 - *Date:* Fri, 18 Apr 2008 12:43:30 -0700
-

On Fri, 18 Apr 2008 11:43:00 -0700, PIEBALD <PIEBALD@xxxxxxxxxxxxxxxxxxxxxxxxxxxx> wrote:

Enum and delegate could indeed be useful. (In particular, I'd like to write a generic version of Enum.Parse to avoid the casts.)

Yes, and I can only guess that support of them was left out because they don't buy as much as class and struct do. But I see no reason why they `_can't_` be supported by an implementation. So I see no reason for the spec to disallow them.

You are arguing the same reasoning that you say shouldn't be argued. You are using what's possible in the implementation as justification for what should exist in the specification. That's just as incorrect as arguing that the specification is a certain way because of a specific implementation.

You are right when you say that the language needs to be designed independently of an implementation. This includes hypothetical implementations. Just because something is possible does not make it desirable in a clean, user-friendly language design.

If, when generics were first added to the language, the spec had said that they `_are_` to be supported then they `_would_` be supported. The argument that the spec says they `_aren't_` because the current implementation doesn't support them is backward.

No one has argued that though.

I have pointed out reasons why it might make sense for System.Enum to not be a valid constraint, `_independent_` of the fact that the spec specifically disallows it (which you have for some time asserted wasn't the case...I assume you now understand your assertion was wrong, even though you haven't explicitly acknowledged that mistake).

I have also pointed out that the specification does specifically disallow the use of System.Enum as a constraint, contrary to your statements that it doesn't.

Re: Compiler Error CS0702

At no point have I said, or even suggested, that the specification disallows the use of `System.Enum` as a constraint `_because_` of the implementation.

A `_language_` exists separate from any particular `_implementation_`.

No one is saying it doesn't.

We can use enums as type parameters, I see no reason not to be allowed to constrain to them. It's not as if they weren't allowed in generics at all.

I would have some sympathy for you if you saw the reason but disagreed with it, or felt it wasn't significant enough to justify not supporting the behavior you want.

But at this point, for you to continue saying that you see `_no_` reason at all, that's just being hard-headed as far as I'm concerned.

I'm looking for a `_conceptual_` reason why they aren't, not an implementation-based reason and so far have seen none.

For me, the most obvious reason is that even though `System.Enum` is a class, actual enums don't really inherit it as a class, in the same way that value types don't really inherit `System.Object` as a class.

For example, if class B inherits class A, then when you write:

```
A a = new B();
```

The variable "a" contains exactly what was returned by the "new B()". But that's not how it works for `System.Enum`. As is the case for any value type, assigning an instance of the value type to a reference type variable requires boxing. So when you write:

```
System.Enum a = MyEnum.Value1;
```

a whole new instance of `System.Enum` is automatically created on your behalf, and a reference to that instance is assigned to the variable "a"..

Value types, in spite of some semantic sleight-of-hand, don't `_really_` inherit classes the same way that reference types do.

Additionally, generics treat value type parameters differently from reference type parameters. One particular difference is that for reference type parameters, a single implementation of the generic can be shared among all concrete uses of the generic. But for value types, a whole new instance is created for each different value type used with the generic.

For better or worse, the C# language specification treats value types differently than it treats reference (class) types. This difference in treatment extends to not allowing a value type to satisfy a reference type constraint in

Re: Compiler Error CS0702

a generic. Since no value type can satisfy the reference type constraint of System.Enum, that means no actual declared enum could do so (since they are always value types) and so it would be pointless for System.Enum to be allowed as a constraint.

Now, it's true: value types can implement `_interfaces_` and an interface is a valid constraint for a generic. So the language designers `_could_` have treated System.Enum as more of an interface, which would then allow for it to be a useful constraint.

But, they didn't. Nor is it clear to me that they easily could have, unless they had anticipated generics during the initial design of C# when enums were introduced. There are other reasons that it makes more sense to treat these special classes as actual classes than as interfaces, related to the fact that they bring an implementation with them.

Interfaces don't provide implementations, but classes do. Having special interfaces such as System.Object, System.Enum, and System.ValueType that include their own implementations would have been at least as awkward as not allowing System.Enum as a constraint for a generic.

So, instead they are special classes, not special interfaces, and with that definition carries the specific limitation within generics that they aren't useful as a constraint because a type that could satisfy them as a constraint is disallowed for other reasons (specifically, that value types aren't allowed to satisfy reference type constraints).

None of the above is unavoidable. It's true, the languages designers could have taken different routes to accomplish similar goals, and in doing so enabled what you're arguing for. But they didn't, and the fact is that design is always about compromise. This includes the design of a language. Inasmuch as the designers `_chose_` not to support this, I'm sure they did so in order to preserve some aspect of the language that was more important to them, such as those things I mention above.

If one regards `_only_` the language, not considering .net, the CLR, CTS etc.;

is there some reason why enum and delegate should `_not_` be supported as constraints?

Yes.

Pete

.