

Re: Design: Custom Exception Usage

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2008-04/msg01185.html>

- *From:* "Peter Duniho" <NpOeStPeAdM@xxxxxxxxxxxxxxxxxxxx>
 - *Date:* Tue, 08 Apr 2008 14:39:58 -0700
-

On Tue, 08 Apr 2008 13:55:55 -0700, Anthony Jones <Ant@xxxxxxxxxxxxxxxxxxxx> wrote:

Hmm... yes perhaps 'plague' is too strong. It is however a bad smell and one would really have to question a choice to use it.

We'll have to agree to disagree.

Taking TryParse as an example I would have preferred to have:–

```
bool CanParse(string)
```

I wouldn't want that at all. If I'm calling CanParse(), 99 times out of a 100 it's because I want to parse the string. It'd be silly to call a method that basically has to do the work of parsing the string, but which only returns a flag telling me if actually parsing the string will work.

If anything the "Try..." pattern is a great example of the usefulness of by-reference parameters.

but TryParse would clearly be faster. Not having CanParse I'm forced to use TryParse even in code where performance is not an issue (which frankly is true of most code).

You write "forced" as though there's something about calling TryParse() that is harmful.

Beyond that, however, since sometimes performance *is* an issue, it's not like TryParse() could have been left out of the API.

```
int x;  
string y = "Brass Monkeys";  
  
if (Int32.TryParse(y, out x))  
{
```

Re: Design: Custom Exception Usage

```
// do stuff with x but how did x get assigned a value its not that
obvious
}
```

Why is it the case that "how did x get assigned a value" is "not that obvious"? The "out" in the parameter list for the call to TryParse() makes it very clear where x is getting assigned.

Especially in the context of C# where the compiler is very strict about dealing with uninitialized variables, I never find myself wondering how something got assigned. Because an "out" parameter is required by the compiler to be assigned in the method being called, a call like this is no exception.

Where I would have preferred:-

```
if (Int32.TryParse(y))
{
    x = Int32.Parse(y);
    // do stuff with x and its clear where x got its value.
}
```

If performance is not a concern (and you already wrote that it almost never is), you can always just wrap a call to Parse() with a try/catch block. That provides the exact assignment pattern you're asking for.

OR alternatively

```
int? x;
string y = "Jon 'Google' Skeets idea"

x = Int32.ParseOrNull(y)
if(x != null)
{
    //do stuff with x
}
```

I would not be against the inclusion of a method like that as an `_optional_` pattern. However, I certainly would resent being forced to use it in situations where I don't already have a nullable type.

Of course its easy enough to build round but it means doing so out of principle rather than actual need.

Indeed. And IMHO it's the principle that's flawed. There's nothing inherently wrong with passing by reference. Why go out of your way to avoid it?

Pete

.