

Re: any regex gurus out there?

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2004-09/5470.html>

From: Niki Estner (*niki.estner_at_cube.net*)

Date: 09/22/04

Date: Wed, 22 Sep 2004 14:38:29 +0200

"bill tie" <billtie@discussions.microsoft.com> wrote in
news:DDA274DF-88D1-47C6-9B36-61D328439DF5@microsoft.com...

> ...

> `myString = "foo & ! (\ * bar";`

> *// after we clean up we should have only "foo spaces bar"*

> `myString = Regex.Replace(myString, "[\\!&*(]", "");`

>

> *No matter what number or combination of backslashes I used it didn't work.*

I assume the first string constant really is `myString = "foo & ! (\ * bar";` – The compiler wouldn't take it otherwise.

The pattern you want to pass to the regex class looks like this: `[\\!&*(]`

You can either write it this way:

```
myString = Regex.Replace( myString, @"[\\!&*(]", "" );
```

or escape each backslash with a backslash, like this:

```
myString = Regex.Replace( myString, "[\\\\!&*(]", "" );
```

Note that this is exactly the same code, it's just a different way to write the string constant.

Also, this doesn't have to do anything with regular expressions: If you want to code a UNC-path in your code like `\\SomeMachine\\SomeFolder`, you'd either write `@ "\\SomeMachine\\SomeFolder"`, or `"\\\\SomeMachine\\SomeFolder"`.

> *The point of this exercise is I want to remove everything except:*

> *letters a through z, A through Z*

> *numerals 0 through 9*

> *hyphen –*

> *space " "*

>

> *I thought I could do it by negation as follows:*

>

> `myString = Regex.Replace(myString, "[^0-9a-zA-Z]", "");`

>

> *This would be fantastic. There seem to be two problems:*

>

microsoft.public.dotnet.languages.csharp: Re: any regex gurus out there?

> (a) *Regex considers some characters, e.g. parentheses, as letters.*

I don't think so. Maybe you have a sample?

> (b) *Letters the user enters may not be ordinary English letters. In the French language, the letter A may have three different accents. I'm not sure how the pattern [a-z] treats these letters.*

They're not included in this set. You'll have to use unicode character classes for that problem:

```
myString = Regex.Replace( myString, @"[\p{Ll}\p{Lu}\p{Lt}\p{Lo}\p{Nd} ]", "" );
```

(As usual, if you want a non-verbatim string, replace all \ characters with \\)

\p matches for a unicode character class:

\p{Ll} matches for "Letter lowercase", no matter what language

\p{Lu} matches for "Letter uppercase", no matter what language

\p{Lt} matches for "Letter titlecase", no matter what language

\p{Lo} matches for "other Letters", no matter what language

\p{Nd} matches for "Numeric decimal"

A list of all unicode character classes can be found here:

<http://msdn.microsoft.com/library/deu/default.asp?url=/library/DEU/cpref/html/frlrfssystemglobalizationunicodecategory>

Note that if you want to include underscores in this set you can use the shortcut 'w' resp. 'W'.

```
myString = Regex.Replace( myString, @"[\w ]", "" );
```

> [2]

>> *The problem with the MatchEvaluator-technique is that*

>> *calling delegates is quite slow, so 15 compiled regex's*

>> *might well be faster than one MatchEvaluator-Pass.*

>

> *I was wondering, too. Yet, some compilers optimize code. I'm terribly*

> *new*

> *to C#. I don't know whether it collapses my "calls" into some sort of*

> *loop*

> *at compile-time.*

There's a good article on .NET optimizations:

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/fastmanagedcode.asp>

As you can see there, delegates are a lot slower than usual function calls.

>> *I'd definitely favour readability over performance.*

>

> *I concur the readability of the matching algorithm is awful.*

Only if you don't know regular expressions. (And that probably applies to every kind of code)

Re: any regex gurus out there?

microsoft.public.dotnet.languages.csharp: Re: any regex gurus out there?

To use one of your previous examples:

```
myString = Regex.Replace( myString, "[^0-9a-zA-Z ]", "" );
```

I think readability of this line is great: Think about what the corresponding C# code would look like; I'm pretty sure it would be a lot more complex, and harder to read.

And you can always use tools like Espresso that make work with regex's really easy.

So, I would disagree in that point.

```
> Albeit I'm more inclined to apply the classic technique I started with,
> I'd
> like to complete this intellectual exercise. Understanding how matching
> and
> grouping work may be useful in other situations.
>
> Using my last example:
>
> Replace
>
> [abcd] with "z"
> [pqrs] with "y"
> [123] with "x"
>
> string myString = "abcd efg pqrs t 1 k 2 m 3 n";
> myString = Regex.Replace( myString, "([abcd])([pqrs])([123])",
> new MatchEvaluator(MyEvaluator));
```

Paranthesis is wrong in this pattern. Use this one:

```
myString = Regex.Replace( myString, "([abcd])([pqrs])([123])",
    new MatchEvaluator(MyEvaluator));
```

```
> ...
> My code looked something like this:
>
> static string MyEvaluator(Match m)
> {
> for ( int i = 0; i < m.Groups.Count; i++ )
> {
> if (m.Groups[i].Length != 0)
> {
> switch (i)
> {
> case 0:
> return "x";
> case 1:
> return "y";
> case 2:
> return "z";
> }
> }
> }
```

Re: any regex gurus out there?

microsoft.public.dotnet.languages.csharp: Re: any regex gurus out there?

```
> return null;
> }
> return null;
> }
>
> This didn't do the job I wanted.
```

Yes, there's a tweak here: Groups[0] always refers to the whole capture; Groups[1] returns the contents of the first parenthesis, and so on. Try this one:

```
static string MyEvaluator(Match m)
{
    for ( int i = 1; i < m.Groups.Count; i++ )
    {
        if (m.Groups[i].Length != 0)
        {
            switch (i)
            {
                case 1:
                    return "x";
                case 2:
                    return "y";
                case 3:
                    return "z";
            }
        }
    }
    return null;
}
```

Niki