Converting math equations to C#

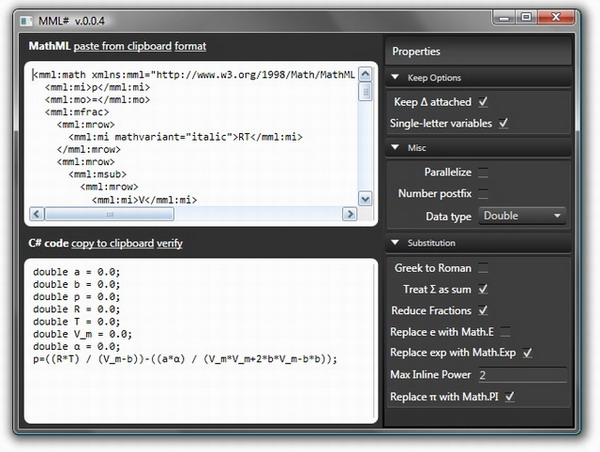


[**Dmitri Nеstеruk**](https://www.codeproject.com/script/Membership/View.aspx?mid=49722), 26 Oct 2012

|  |  |
| --- | --- |
| https://codeproject.global.ssl.fastly.net/script/Ratings/Images/stars-fill-md.png  https://codeproject.global.ssl.fastly.net/script/Ratings/Images/stars-empty-md.png | 4.93 (163 votes) |
|  | |  |  |  | | --- | --- | --- | | Rate this: | [vote 1vote 2vote 3vote 4vote 5](https://www.codeproject.com/Articles/30615/Converting-math-equations-to-C) |  | |

A magical tool to convert Word equations to C# - instantly!

* [**Download executable - 128 KB**](https://www.codeproject.com/KB/recipes/mmlsharp/mmlsharprelease.zip)
* [**Download source code - 173 KB**](https://www.codeproject.com/KB/recipes/mmlsharp/mmlsharpsource.zip)



Introduction

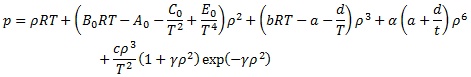
A while ago, I worked on a product where part of the effort involved turning math equations into code. At the time, I wasn't the person who was allocated the role, so my guess is the code was written by simply taking the equations from word and translating them by hand into C#. All well and good, but it got me thinking: is there a way to automate this process so that human error can be eliminated from this, admittedly boring, task? Well, turns out it is possible, and that's what this article is about.

Equations, eh?

I'd guess that barring any special math packages (such as Matlab), most of us developers get math requirements in Word format. For example, you might get something as simple as this:

mmlsharp/MmlSharp1.jpg

This equation is easy to program. Here, let me do it: y = a\*x\*x + b\*x + c;. However, sometimes, you end up getting *really* nasty equations, kind of like the following:

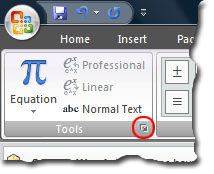


Got the above from [Wikipedia](http://en.wikipedia.org/wiki/Equation_of_state). Anyways, you should be getting the point by now: the above baby is a bit too painful to program. I mean, I'm sure if you have an infinite budget or access to very cheap labour, you could do it, but I guarantee you'd get errors, since getting it right every time (if you've got a hundred) is difficult.

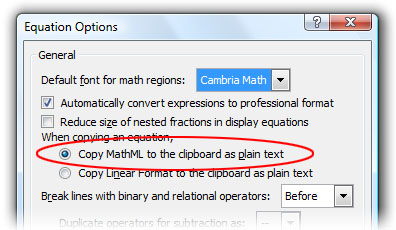
So, my thinking was: hey, there ought to be a way of getting the equation data structured somehow, and then you could restructure it for C#. That's where MathML entered the picture.

MathML

Okay, so you are probably wondering what this [MathML](http://en.wikipedia.org/wiki/MathML) beast is. Basically, it's an XML-like mark-up language for math. If all browsers supported it, you'd be seeing the equations above rendered using the browser's characters instead of bitmaps. But regardless, there's one tool that supports it: Word. Microsoft Word 2007, to be precise. There's a little-known trick to get Word to turn equations into MathML. You basically have to locate the equation options...



and choose the MathML option:



Okay, now copying our first equation onto the clipboard will result in something like the following:

|  |  |
| --- | --- |
| mmlsharp/MmlSharp3.jpg | Hide   Copy Code  <mml:math>  <mml:mi>y</mml:mi>  <mml:mo>=</mml:mo>  <mml:mi>a</mml:mi>  <mml:msup>  <mml:mrow>  <mml:mi>x</mml:mi>  </mml:mrow>  <mml:mrow>  <mml:mn>2</mml:mn>  </mml:mrow>  </mml:msup>  <mml:mo>+</mml:mo>  <mml:mi mathvariant="italic">bx</mml:mi>  <mml:mo>+</mml:mo>  <mml:mi>c</mml:mi>  </mml:math> |

You can probably guess what this all means by looking at the original equation. Hey, we just ripped out the structure of an equation! That's pretty cool, except for one problem: converting it to C#! (Otherwise, it's meaningless.)

Syntax tree

Keeping data the way we get it is no good. There's lots of extra information (like that italic statement near bx), and there's info missing (like the multiplication sign that ought to be between b and x). So, our take on the problem is turn this XML structure into a more OOP, XML-like structure. In fact, that's what the program does – it turns XML elements into corresponding C# classes. In most cases, XML and C# have a 1-to-1 correspondence, so that an <mi/> element turns into an Mi class. So woo-hoo, without too much effort, we turn XML into a syntax tree. Now, the tree is imperfect, but it's there. Let us instead discuss some of the thorny issues that we have to overcome.

**Single/multi-letter variables**

Does 'sin' mean *s* times *i* times *n*, or a variable called 'sin', or the Math.Sin function? When I looked at the equations I had, some of them used multiple letters, some were single-letter. There's no 'one size fits all' solution as to how to treat those. Basically, I made this an option.

**The times (×) sign**

If you write *ab*, it might mean *a* times *b*. If that's the case, you need to find all the locations where the multiplication has been omitted. On a funny note, there are also different Unicode symbols used by the times sign in different math editing packages (I was testing with MathML as well as Word). The end result is that finding where the multiplication sign is missing is very difficult.

**Greek to Roman**

Some people object to having Greek constants in C# code. Hey, I code in UTF-8, so I can include anything, including Japanese characters and those other funny Unicode symbols. It does mess up IntelliSense because your keyboard probably doesn't have Greek keys - unless you live in Greece, that is. Plus, it's a way to very quickly kill maintainability. So, one feature I had to add is turning Greek letters into Roman descriptions, so that Δ would become Delta and so on. Actually, Delta is a special case because we are so used to attaching it to our variables (e.g., writing ΔV). Consequently, I added a special rule for Δ to be kept attached even in cases where all other variables are single-letter.

**Correctly treating e, π, and exp**

Basically, the letter pi (π) can be just a variable, or it can mean Math.PI. Same goes for the letter e – it could be Math.E, and in most cases, it is. Another, more painful substitution is exp to Math.Exp. Support for all three of these had to be added.

**Power inlining**

Most people know that x\*x is faster than Math.Pow(x, 2.0), especially when dealing with integers. Inlining powers of X and above is an option in the program. I have seen articles (can't find the link) where people claim that you lose precision if you avoid doing it the Math.Pow way. I'm not sure though.

**Operation reduction**

I've been alerted to the fact that some expressions output are inefficient as far as their constituent operations go. For example, a\*x\*x+b\*x+c is not as efficient as x\*(a\*x+b)+c because it has more multiplications. Thus, one of the future goals of my solution is to attempt to optimize these scenarios. It will make them less readable though!

There were plenty of other problems in converting from XML to C#, but the main idea stayed the same: correctly implement the Visitor pattern over each possible MathML element, removing unnecessary information and supplying that information which is missing. Let's look at some examples.

Examples

Okay, I bet you can't wait to see an actual example. Let's start with what we had before:

mmlsharp/MmlSharp1.jpg

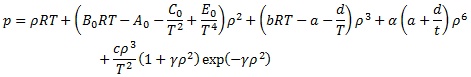
Here's the output we get:

Hide   Copy Code

y=a\*x\*x+b\*x+c;

I omitted the initialization steps for variables that the program also creates.

Let's look at the more complex equation. Here it is, in case you have forgotten:



Care to guess what the output of our tool is?

Hide   Copy Code

p = rho\*R\*T + (B\_0\*R\*T-A\_0-((C\_0) / (T\*T))+((E\_0) / (Math.Pow(T, 4))))\*rho\*rho +

(b\*R\*T-a-((d) / (T)))\*Math.Pow(rho, 3) +

alpha\*(a+((d) / (t)))\*Math.Pow(rho, 6) +

((c\*Math.Pow(rho, 3)) / (T\*T))\*(1+gamma\*rho\*rho)\*Math.Exp(-gamma\*rho\*rho);

I originally had the above output using Greek letters (reminder: C# is okay with them). However, due to coding, I've let my tool change them to Romanized versions, thus demonstrating yet another feature.

Okay, let's do another example just to be sure – this time with a square root. Here is the equation:

mmlsharp/MmlSharp5.jpg

I've turned power inlining off for this one - we don't want the expression with the root being evaluated twice. Here is the output:

Hide   Copy Code

a = 0.42748 \* ((Math.Pow((R\*T\_c), 2)) / (P\_c)) \*

Math.Pow((1 + m \* (1 - Math.Sqrt(T\_r))), 2);

Is this great or what? If you are ever handed a 100-page document full of formulae, well, you can surprise your client by coding them really quickly.

Conclusion

I hope you like the tool. Maybe you'll even find it useful. I have recently redesigned the tool from the ground up using F#, and you can find the latest version [here](http://activemesa.com/mathsharp).

License

This article, along with any associated source code and files, is licensed under [The Code Project Open License (CPOL)](http://www.codeproject.com/info/cpol10.aspx)

About the Author



[**Dmitri Nеstеruk**](https://www.codeproject.com/Members/nesteruk)

|  |  |  |
| --- | --- | --- |
| [Twitter](https://www.twitter.com/dnesteruk) | Founder ActiveMesa |  |
| United Kingdom United Kingdom |

I work primarily with the .NET technology stack, and specialize in accelerated code production via code generation (static or dynamic), aspect-oriented programming, MDA, domain-specific languages and anything else that gets products out the door faster. My languages of choice are C# and F#, though I'm open to suggestions.  
  
I'm a Microsoft MVP (Visual C#) since 2009. I run a collective tech blog at [DevTalk.net](http://devtalk.net/). I use my own editor called [TypograFix](http://bitbucket.org/nesteruk/typografix) to typeset articles and blog posts.  
  
Like the article and want this implemented in your product? Got a project that can benefit from Microsoft.Net goodness? Then get in touch!

<https://www.codeproject.com/Articles/30615/Converting-math-equations-to-C>