Content MathML*

Sarah Coppin
Brent Hendricks

This work is produced by OpenStax-CNX and licensed under the
Creative Commons Attribution License 1.0†

Abstract

A short introduction to writing Content MathML by hand. It covers tokens, prefix notation, and applying functions and operators. In addition it introduces writing derivatives, integrals, vectors, and matrices.

The authoritative reference for Content MathML is Section 4 of the MathML 2.0 Specification[1] . The World Wide Web Consortium (W3C) is the body that wrote the specification for MathML. The text is very readable and it is easy to find what you are looking for. Look there for answers to questions that are not answered in this tutorial or when you need more elaboration. This tutorial is based on MathML 2.0.

In this document, the `m` prefix is used to denote tags in the MathML namespace. Thus the `<apply>` tag is referred to as `<m:apply>`. Remember all markup in the MathML namespace must be surrounded by `<m:math>` tags.

1 The Fundamentals of Content MathML: Applying Functions and Operators

The fundamental concept to grasp about Content MathML is that it consists of applying a series of functions and operators to other elements. To do this, Content MathML uses prefix notation. **Prefix notation** is when the operator comes first and is followed by the operands. Here is how to write "2 plus 3".

```
<m:math>
          <m:apply>
    <m:plus/>
    <m:cn>2</m:cn>
            <m:cn>3</m:cn>
  </m:apply>
</m:math>
```

This would display as $2 + 3$.

There are three types of elements in the Content MathML example shown above. First, there is the `apply` tag, which indicates that an operator (or function) is about to be applied to the operands. Second, there is the function or operator to be applied. In this case the operator, `plus`, is being applied. Third, the operands follow the operator. In this case the operands are the numbers being added. In summary, the apply tag applies the function (which could be sin or f, etc.) or operator (which could be plus or minus, etc.) to the elements that follow it.

*Version 2.15: Nov 2, 2005 5:13 pm -0600
†http://creativecommons.org/licenses/by/1.0
[1]http://www.w3.org/TR/MathML2/chapter4.html

1.1 Tokens

Content MathML has three tokens: `ci`, `cn`, and `csymbol`. A **token** is basically the lowest level element. The tokens denote what kind of element you are acting on. The **cn** tag indicates that the content of the tag is a number. The **ci** tag indicates that the content of the tag is an identifier. An **identifier** could be any variable or function; x, y, and f are examples of identifiers. In addition, `ci` elements can contain Presentation MathML. Tokens, especially `ci` and `cn`, are used profusely in Content MathML. Every number, variable, or function is marked by a token.

csymbol is a different type of token from `ci` and `cn`. It is used to create a new object whose semantics is defined externally. It can contain plain text or Presentation MathML. If you find that you need something, such as an operator or function, that is not defined in Content MathML, then you can use csymbol to create it.

Both `ci` and `csymbol` can use Presentation MathML to determine how an identifier or a new symbol will be rendered. To learn more about Presentation MathML see Section 3 of the MathML 2.0 Specification[2] . For example, to denote "x with a subscript 2", where the 2 does not have a more semantic meaning, you would use the following code.

```
<m:math>
  <m:ci>
    <m:msub>
      <m:mi>x</m:mi>
      <m:mn>2</m:mn>
    </m:msub>
  </m:ci>
</m:math>
```

This would display as x_2.

The `ci` elements have a type attribute which can be used to provide more information about the content of the element. For example, you can declare the contents of a `ci` tag to be a function (`type='fn'`), or a vector (`type='vector'`), or a complex number (`type='complex'`), as well as any number of other things. Using the type attribute helps encode the meaning of the math that you are writing.

1.2 Functions and Operators

In order to apply a function to a variable, make the function the first argument of an apply. The second argument will be the variable. For example, you would use the following code to encode the meaning, "the function f of x". (Note that you have to include the attribute `type='fn'` on the `ci` tag denoting f.)

```
<m:math>
  <m:apply>
    <m:ci type='fn'>f</m:ci>
    <m:ci>x</m:ci>
  </m:apply>
</m:math>
```

This will display as $f(x)$.

There are also pre-defined functions and operators in Content MathML. For example, sine and cosine are predefined. These predefined functions and operators are all **empty tags** and they directly follow the apply tag. "The sine of x" is similar to the example above.

[2] http://www.w3.org/TR/MathML2/chapter3.html

```
<m:math>
  <m:apply>
    <m:sin/>
    <m:ci>x</m:ci>
  </m:apply>
</m:math>
```

This will display as $\sin(x)$.

You can find a more thorough description of the different predefined functions in Chapter 4 of the MathML specification.

In addition to the predefined functions, there are also many predefined operators. A few of these are `plus` (for addition), `minus` (for subtraction), `times` (for multiplication), `divide` (for division), `power` (for taking the nth-power of something), and root (for taking the nth-root of something).

Most operators expect a specific number of child tags. For example, the power operator expects two children. The first child is the base and the second is the value in the exponent. However, there are other tags which can take many children. For example, the plus operator merely expects one or more children. It will add together all of its children whether there are two or five. This is referred to as an **n-ary operator**.

Representing "the negative of a variable" and explicitly representing "the positive of a variable or number" has slightly unusual syntax. In this case you apply the plus or minus operator to the variable or number, etc., in question. The following is the code for "negative x."

```
<m:math>
  <m:apply>
    <m:minus/>
    <m:ci>x</m:ci>
  </m:apply>
</m:math>
```

This will display as $-x$.

In contrast to representing the negative of a variable, the negative of a number may be coded as follows:

```
<m:math><m:cn>-1</m:cn></m:math>
```

This will display as -1.

To create more complicated expressions, you can nest these bits of apply code within each other. You can create arbitrarily complex expressions this way. "a times the quantity b plus c" would be written as follows.

```
<m:math>
  <m:apply>
    <m:times/>
    <m:ci>a</m:ci>
    <m:apply>
      <m:plus/>
      <m:ci>b</m:ci>
      <m:ci>c</m:ci>
```

```
      </m:apply>
    </m:apply>
  </m:math>
```

This will display as $a\left(b+c\right)$.

The `eq` operator is used to write equations. It is used in the same way as any other operator. That is, it is the first child of an apply. It takes two (or more) children which are the two quantities that are equal to each other. For example, "a times b plus a times c equals a times the quantity b plus c" would be written as shown.

```
  <m:math>
    <m:apply>
      <m:eq/>
      <m:apply>
        <m:plus/>
        <m:apply>
  <m:times/>
  <m:ci>a</m:ci>
  <m:ci>b</m:ci>
        </m:apply>
        <m:apply>
  <m:times/>
  <m:ci>a</m:ci>
  <m:ci>c</m:ci>
        </m:apply>
      </m:apply>
      <m:apply>
        <m:times/>
        <m:ci>a</m:ci>
        <m:apply>
  <m:plus/>
  <m:ci>b</m:ci>
  <m:ci>c</m:ci>
        </m:apply>
      </m:apply>
    </m:apply>
  </m:math>
```

This will display as $ab+ac=a\left(b+c\right)$.

2 Integrals

The operator for an integral is `int`. However, unlike the operators and functions discussed above, it has children that define the independent variable that you integrate with respect to (`bvar`) and the interval over which the integral is taken (use either `lowlimit` and `uplimit`, or `interval`, or `condition`). `lowlimit` and `uplimit` (which go together), `interval`, and `condition` are just three different ways of denoting the integrands. Don't forget that the bvar, `lowlimit`, `uplimit`, `interval`, and `condition` children take token elements as well. The following is "the integral of f of x with respect to x from 0 to b."

```
<m:math>
          <m:apply>
    <m:int/>
    <m:bvar><m:ci>x</m:ci></m:bvar>
    <m:lowlimit><m:cn>0</m:cn></m:lowlimit>
    <m:uplimit><m:ci>b</m:ci></m:uplimit>
    <m:apply>
      <m:ci type='fn'>f</m:ci>
      <m:ci>x</m:ci>
    </m:apply>
  </m:apply>
</m:math>
```

This will display as $\int_0^b f(x)\,dx$.

3 Derivatives

The derivative operator is `diff`. The derivative is done in much the same way as the integral. That is, you need to define a base variable (using `bvar`). The following is "the derivative of the function f of x, with respect to x."

```
<m:math>
          <m:apply>
    <m:diff/>
    <m:bvar>
      <m:ci>x</m:ci>
    </m:bvar>
    <m:apply>
      <m:ci type="fn">f</m:ci>
      <m:ci>x</m:ci>
    </m:apply>
  </m:apply>
</m:math>
```

This will display as $\frac{d}{dx} f(x)$.

To apply a higher level derivative to a function, add a `degree` tag inside of the `bvar` tag. The degree tag will contain the order of the derivative. The following shows "the second derivative of the function f of x, with respect to x."

```
<m:math>
          <m:apply>
            <m:diff/>
    <m:bvar>
      <m:ci>x</m:ci>
      <m:degree><m:cn>2</m:cn></m:degree>
    </m:bvar>
    <m:apply><m:ci type="fn">f</m:ci>
      <m:ci>x</m:ci>
    </m:apply>
  </m:apply>
</m:math>
```

This will display as $\frac{d^2}{dx^2} f(x)$.

4 Vector and Matrices

Vectors are created as a combination of other elements using the `vector` tag.

```
<m:math>
          <m:vector>
    <m:apply>
      <m:plus/>
      <m:ci>x</m:ci>
      <m:ci>y</m:ci>
    </m:apply>
    <m:ci>z</m:ci>
    <m:cn>0</m:cn>
  </m:vector>
</m:math>
```

This will display as $\begin{pmatrix} x+y \\ z \\ 0 \end{pmatrix}$.

Matrices are done in a similar manner. Each `matrix` element contains several `matrixrow` elements. Then each `matrixrow` element contains several other elements.

```
<m:math>
          <m:matrix>
    <m:matrixrow>
      <m:ci>a</m:ci>
      <m:ci>b</m:ci>
      <m:ci>c</m:ci>
    </m:matrixrow>
    <m:matrixrow>
      <m:ci>d</m:ci>
      <m:ci>e</m:ci>
      <m:ci>f</m:ci>
    </m:matrixrow>
    <m:matrixrow>
      <m:ci>g</m:ci>
      <m:ci>h</m:ci>
      <m:ci>j</m:ci>
    </m:matrixrow>
  </m:matrix>
</m:math>
```

This will display as $\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & j \end{pmatrix}$.

There are also operators to take the determinant and the transpose of a matrix as well as to select elements from within the matrix.

5 Entities

NOTE: The use of MathML character entity references in Connexions content is **deprecated**.

MathML defines its own entities for many special characters used in mathematical notation. While the entity references have the advantage of being mnemonic with respect to the characters they stand for, they also entail some technical limitations, and so their use in Connexions content is deprecated. Please use the UTF-8-encoded Unicode characters themselves where possible, or, failing that, the XML Unicode character references for the characters. At some time in the future, the Connexions repository system will likely convert entity references and character references silently to the UTF-8-encoded Unicode characters they stand for. See 6.2.1 Unicode Character Data[3] from the XML Specification for more information. The MathML specification contains a list of character entities with their corresponding Unicode code points[4] .

There are character picker utilities available to help you select and paste UTF-8 characters into applications like Connexions. If you are running Microsoft Windows, the Windows accessory Character Map can help you. The "Lucida Sans Unicode" font seems to have a good selection of mathematical operators and special characters. Under Linux, the charmap utility and GNOME applet provide access to all Unicode characters.

6 Other Resources

There is a lot more that can be done with Content MathML. Especially if you are planning on writing a lot of Content MathML, it is well worth your time to take a look at the MathML specification[5] .

[3] http://www.w3.org/TR/2003/REC-MathML2-20031021/chapter6.html#chars.unicodechars
[4] http://www.w3.org/TR/MathML2/bycodes.html
[5] http://www.w3.org/TR/MathML2/