Hans Hagen
Pragma ADE
Hasselt
pragma@wxs.nl

ConTEXt MathML

Introduction

It is a well known fact that TEX can do a pretty good job on typesetting math. This is one reason why many scientific articles, papers and books are typeset using TEX. However, in these days of triumphing angle brackets, coding in TEX looks more and more out of place.

From the point of view of an author, coding in TEX is quite natural, given that some time is spent on reading the manuals. This is (I'm told) because not only the natural flow of the definition suits the way mathematicians think, but also because the author has quite some control over the way his thoughts end up on paper. It will be no surprise that switching to a more restricted way of coding, which also demands more keystrokes, is not on forehand considered to be better.

There are however circumstances that one wants to share formulas (or formula–like specifications) between several applications, one of which is a typesetting engine. In that case, a bit more work now, later saves you some headaches due to keeping the different source documents in sync.

As soon as coding math in angle brackets is discussed, those in favour stress that coding can be eased by using appropriate editors. Here we encounter a dilemma. For optimal usage, one should code in terms of content, that is, the principles that are expressed in a formula. Editors are not that strong in this area, and if they would be, editing would be not that much different from traditionally editing formulas: just keying in ideas using code that at first sight looks obscure. A more graphical oriented editor can help authors to compose formulas, but the underlaying coding will mainly be in terms of placing glyphs and boxes, and as a result the code will hardly be usable in other applications.

So either we code in terms of concepts, which permits sharing code among applications, and poses strong limitations on the influence of authors on the visual appearance. Or we use an interactive editor to fine tune the appearance of a formula and take for granted that reuse will be minimal or suboptimal.

In the following chapters we will discuss the mathematical language MATHML in the perspective of typography. As a typesetting vehicle, we have used CONTEXT. However, the principles introduced here and the examples that we provide are independent of CONTEXT. For a more formal exploration we recommend the MATHML specification.

This document is dedicated to all those CONTEXT users who like typesetting math. I'm sure that my father, who was a math teacher, would have liked proofreading this document. His absence was compensated by Tobias Burnus, Wang Lei, Ton Otten, and members of the CONTEXT mailing list who carefully read the text, corrected the errors in my math, tested the functionality, and made suggestions. Any remaining errors are mine.

What is MATHML

Backgrounds

MATHML showed up in the evolving vacuum between structural SGML markup and presentational HTML. Both SGML and HTML can be recognized by angle brackets. The disadvantage of SGML was that is was so open ended, that general tools could hardly be developed. HTML on the other hand was easy to use and became extremely popular and users as well as software vendors quickly spoiled the original ideas and created a mess. SGML never became really popular, but thanks to HTML people became accustomed to that kind of notation. So, when XML came around as a more restricted cousin of SGML, the world was kind of ready for it. It cannot be denied that by some clever marketing many of today's users think that they use something new and modern, while we are actually dealing with something from the early days of computing. A main benefit of XML is that it brought the ideas behind SGML (and medium neutral coding in general) to the users and at the same time made a major cleanup of HTML possible.

About the same time, MATHML was defined, both to bring math to the WWW, and to provide a way of coding math that will stimulate sharing the same code between different applications. At the end of 2000, the MATHML version 2 draft became a recommendation.

Now, imagine that we want to present a document on the internet using a format like HTML, either for viewing or for aural reproduction. Converting text and graphics is, given proper source coding, seldom a problem, but converting formulas into some angle bracket representation is more tricky. A way out of this is MATHML's presentational markup.

$$a + b = c$$

This simple formula, when coded in TEX, looks like:

```
$$ a + b = c $$
```

In presentational MATHML we get:

```
<math>
  <mrow>
    <mi> a </mi>
    <mo> + </mo>
    <mi> b </mi>
    <mo> = </mo>
    <mi> c </mi>
  </mrow>
</math>
```

In presentational MATHML, we use mostly begintags (`<mi>`) and end tags (`</mi>`). The *row* element is the basic building block of a formula. The *mi* element specifies a math identifier and *mo* is used for operators. In the process of typesetting, both are subjected to interpretation in order to get the best visualization.

Converting TEX code directly or indirectly, using the DVI output or even in–memory produced math lists, has been one of the driving forces behind presentational MATHML and other math related DTD's like EUROMATH. One may wonder if there are sound and valid reasons for going the opposite way. You can imagine that a converter from TEX to MATHML produces *menclose*, *mspace*, *mstyle* and other elements that can have many spacing related attributes, but I wonder if any author is willing to think in those quantities. Visual editors of course are good candidates for producing presentational MATHML.

But wouldn't it be more efficient if we could express ideas and concepts in such a way that they could be handled by a broad range of applications, including a typesetting engine? This is why, in addition to presentational MATHML, there is also content MATHML. The previous formula, when coded in such a way, looks like:

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

This way of defining a formula resembles the so called polish (or stackwise) notation. Opposite to presentational markup, here a typesetting engine has to find out in what order and what way the content has to be presented. This may seem a disadvantage, but in practice implementing content markup is not that complicated. The big advantage is that, once we know how to typeset a concept, TEX can do a good job, while in presentational markup much hard coded spacing can spoil everything. One can of course ignore specific elements, but it is more safe to start from less and enhance, than to leave away something with unknown quantities.

Instead of using hard coded operators as in presentational MATHML, content markup uses empty elements like `<plus/>`. Many operators and functions are predefined but one can also define his own, which is not entirely en par with the concept.

Of course the main question to be answered now is to what extent the author can influence the appearance of a formula defined in content markup. Content markup has the advantage that the results can be more consistent, but taking away all control is counterproductive. The MATHML level 2 draft mentions that this level covers most of the pre university math. If so, that is a proper starting point, but especially educational math often has to be typeset in such ways that it serves its purpose. Also, (re)using the formulas in other applications (simulators and alike) is useful in an educational setting, so content markup is quite suitable.

How do we combine the advantages of content markup with the wish of an author to control the visual output and at the same time get an as high as possible typeset result. There are several ways to accomplish this. One is to include in the document source both the content markup and the TEX specific code.

```
<math>
  <semantics>
    <apply> <eq/>
      <apply> <plus/>
        <ci> a </ci>
        <ci> b </ci>
      </apply>
    </apply>
    <ci> c </ci>
    <annotation encoding="TeX">a+b=c</annotation>
  </semantics>
</math>
```

The *annotation* element is one of the few that is permitted inside the *math* element. In this example, we embed pure TEX code, which, when enabled is typeset in math mode. It will be clear that for a simple formula like this one, such redundant coding is not needed, but one can imagine more complicated formulas. Because we want to limit the amount of work, we prefer just content markup.

Two methods

The best way to learn MATHML is to key in formulas, so that is what we did as soon as we started adding MATHML support to CONTEXT. In some areas, MATHML provides much detail (many functions are represented by elements) while in other areas one has to fall back on the more generic function element or a full description. Compare the following definitions:

```
<math> <apply> <sin/> <ci> x </ci> </apply> </math>
<math> <mrow> <mo> sin </mo> <mi> x </mi> </mrow> </math>
```

We prefer the first definition because it is more structured and gives more control over the result. There is only one ‘unknown’ quantity, x, and from the encapsulating element *ci* we know that it is an identifier.

$$\sin x$$

$$\sin x$$

In the content example, from the *apply sin* we can deduce that the following argument is an operand, either an *apply*, or an *ci* or *cn*. In the presentational alternative, the following elements can be braces, a math identifier, a row, a sequence of identifiers and operators, etc. There, the look and feel is hard coded.

```
<?context-mathml-directive function reduction no ?>
```

This directive, either issued in the XML file, or set in the style file, changes the appearance of the function, but only in content markup. It is because of this feature, that we favour content markup.

$$\sin(x)$$

$$\sin x$$

Does this mean that we can cover everything with content markup? The answer to this is still unclear. Consider the following definition.

Here we combine several cases in one formula by using $\pm$ and $\mp$ symbols. Because we only have *plus* and *minus* elements, we have to revert to the generic function element *fn*. We show the complete definition of this formula.

```
[file pc-i-380.xml does not exist]
```

The MATHML parser and typesetting engine have to know how to handle these special cases, because the visualization depends on the function (or operator). Here both composed signs are treated like the plus and minus signs, but in other cases an embraced argument may be needed. Each special case needs a specific handler.

Presentational markup

If a document contains presentational MATHML, there is a good chance that the code is output by an editor. Here we will discuss the presentation elements that make sense for users when they want to manually code presentational MATHML. In this chapter we show the default rendering, later we will discuss options.

Although much is permitted, we advise to keep the code as simple as possible, because then TEX can do a rather good job on interpreting and typesetting it. Just let TEX take care of the spacing.

mi, mn, mo

Presentational markup comes down to pasting boxes together in math specific ways. The basic building blocks are these three character elements.

$$x = 5$$

```
<math>
  <mrow>
    <mi> x </mi> <mo> = </mo> <mn> 5 </mn>
  </mrow>
</math>
```

mi	identifier	normally typeset in an italic font
mn	number	normally typeset in a normal font
mo	operator	surrounded by specific spacing

mrow

The previous example demonstrated the use of *mrow*, the element that is used to communicate the larger building blocks. Although this element from the perspective of typesetting is not always needed, by using it, the structure of the formula in the document source is more clear.

msup, msub, msubsup

Where in content markup super and subscript are absent and derived from the context, in presentational markup they are quite present.

$${x_1}^2$$

```
<math>
  <msup>
    <msub> <mi> x </mi> <mn> 1 </mn> </msub>
    <mn> 2 </mn>
  </msup>
</math>
```

$$x_1^2$$

```
<math>
  <msubsup>
    <mi> x </mi>
    <mn> 1 </mn>
    <mn> 2 </mn>
  </msubsup>
</math>
```

Watch the difference between both definitions and appearances. You can influence the default behaviour with processing instructions.

mfrac

Addition, subtraction and multiplication is hard coded using the *mo* element with +, −, and × (or nothing). You can use / for division, but for more complicated formulas you have to fall back on fraction building. This is why MATHML provides the *mfrac*.

$$\frac{x+1}{y+1}$$

```
<math>
  <mfrac>
    <mrow> <mi> x </mi> <mo> + </mo> <mn> 1 </mn> </mrow>
    <mrow> <mi> y </mi> <mo> + </mo> <mn> 1 </mn> </mrow>
  </mfrac>
</math>
```

You can change the width of the rule, but this is generally a bad idea. For special purposes you can set the line thickness to zero.

$$\genfrac{}{}{0pt}{}{x \geq 2}{y \leq 4}$$

```
<math>
  <mfrac linethickness="0">
    <mrow> <mi> x </mi> <mo> &geq; </mo> <mn> 2 </mn> </mrow>
    <mrow> <mi> y </mi> <mo> &leq; </mo> <mn> 4 </mn> </mrow>
  </mfrac>
</math>
```

mfenced
Braces are used to visually group sub–expressions. In presentational MATHML you can either hard code braces, or use the *mfenced* element to generate delimiters automatically.

```
<mo>(</mo> <mi> x </mi> <mo> + </mo> <mn> 1 </mn> <mo>)</mo>
<mfenced>  <mi> x </mi> <mo> + </mo> <mn> 1 </mn> </mfenced>
```

In CONTEXT, as much as possible, the operators and identifiers are interpreted, and when recognized treated according to their nature.

$$\frac{(x+1)\,(x-1)}{(y+1)\,(y-1)}$$

```
<math>
  <mfrac>
    <mrow>
      <mfenced> <mi> x </mi> <mo> + </mo> <mn> 1 </mn> </mfenced>
      <mfenced> <mi> x </mi> <mo> - </mo> <mn> 1 </mn> </mfenced>
    </mrow>
    <mrow>
      <mfenced> <mi> y </mi> <mo> + </mo> <mn> 1 </mn> </mfenced>
      <mfenced> <mi> y </mi> <mo> - </mo> <mn> 1 </mn> </mfenced>
    </mrow>
  </mfrac>
</math>
```

The braces adapt their size to the content. Their dimensions also depend on the way math fonts are defined. The standard TEX fonts will give the same height of braces around x and y, but in other fonts the y may invoke slightly larger ones.

$$(x,y,z)$$

```
<math>
  <mfenced open="(" close=")" separators=",">
    <mi> x </mi> <mi> y </mi> <mi> z </mi>
  </mfenced>
</math>
```

The separators will adapt their size to the fenced content.

$$\left[\frac{1}{x}\middle|\frac{1}{y}\middle|\frac{1}{z}\right]$$

```
<math>
  <mfenced open="[" close="]" separators="|">
    <mfrac> <mn> 1 </mn> <mi> x </mi> </mfrac>
    <mfrac> <mn> 1 </mn> <mi> y </mi> </mfrac>
    <mfrac> <mn> 1 </mn> <mi> z </mi> </mfrac>
  </mfenced>
</math>
```

msqrt, mroot

The shape and size of roots, integrals, sums and products can depend on the size of the content.

$$\sqrt{b}$$

```
<math>
  <msqrt>
    <mi> b </mi>
  </msqrt>
</math>
```

$$\sqrt[2]{b}$$

```
<math>
  <mroot>
    <mi> b </mi>
    <mn> 2 </mn>
  </mroot>
</math>
```

$$\sqrt[2]{\frac{1}{b}}$$

```
<math>
  <mroot>
    <mfrac> <mn> 1 </mn> <mi> b </mi> </mfrac>
    <mn> 2 </mn>
  </mroot>
</math>
```

$$\sqrt[3]{\frac{1}{a+b}}$$

```
<math>
  <mroot>
    <mfrac>
      <mn> 1 </mn>
      <mrow> <mi> a </mi> <mo> + </mo> <mi> b </mi> </mrow>
    </mfrac>
    <mn> 3 </mn>
  </mroot>
</math>
```

mtext

If you put text in a *mi* element, it will come out rather ugly. This is due to the fact that identifiers are (at least in TEX) not subjected to the kerning that is normally used in text. Therefore, when you want to add some text to a formula, you should use the *mtext* element.

$$\frac{Some Text}{\text{Some Text}}$$

```
<math>
  <mfrac>
    <mi> Some Text </mi>
    <mtext> Some Text </mtext>
  </mfrac>
</math>
```

mover, munder, munderover

Not all formulas are math and spacing and font rules may differ per discipline. The following formula reflects a chemical reaction.

$$2H_2 + O_2 \xrightarrow[\text{explosion}]{} 2H_2O$$

```
<math>
  <mrow>
    <mrow>
      <mn> 2 </mn>
      <msub> <mtext> H </mtext> <mn> 2 </mn> </msub>
    </mrow>
    <mo> + </mo>
    <msub> <mtext> O </mtext> <mn> 2 </mn> </msub>
    <munder>
      <mo> &RightArrow; </mo>
      <mtext> explosion </mtext>
    </munder>
    <mrow>
      <mn> 2 </mn>
      <msub> <mtext> H </mtext> <mn> 2 </mn> </msub>
      <mtext> O </mtext>
    </mrow>
  </mrow>
</math>
```

The *munder*, *mover* and *munderover* elements can be used to put symbols and text or formulas on top of each other. When applicable, the symbols will stretch themselves to span the natural size of the text or formula.

ms

This is a bit weird element. It behaves like *mtext* but puts quotes around the text.

$$\frac{\text{"Some Text"}}{\text{Some Text}}$$

```
<math>
  <mfrac>
    <ms> Some Text </ms>
    <mtext> Some Text </mtext>
  </mfrac>
</math>
```

You can specify the left and right boundary characters, either directly or (preferably) using entities like *quot*.

$$\text{+A Famous Quotation+}$$

```
<math>
  <ms lquote="+" rquote="+"> A Famous Quotation </ms>
</math>
```

menclose

This element is implemented but it is such a weird element that we don't yet describe it here.

merror

There is not much chance that this element will end up in a math textbook, unless the typeset output of programs is part of the story.

$$\text{Are you kidding?} \quad \frac{1+x}{0}$$

```
<math>
  <merror>
    <mtext> Are you kidding? &ThickSpace; </mtext>
    <mfrac>
      <mrow> <mn> 1 </mn> <mo> + </mo> <mi> x </mi> </mrow>
      <mn> 0 </mn>
    </mfrac>
  </merror>
</math>
```

mmultiscripts, mprescripts

This element is one of the less obvious ones. The next two examples are taken from the specification. The *multiscripts* element takes an odd number of arguments. The second and successive child elements alternate between sub- and superscript. The empty element *none* —a dedicated element *mnone* would have been a better choice— serves as a placeholder.

$$R_i{}^j{}_{kl}$$

```
<math>
  <mmultiscripts>
    <mi> R </mi>
    <mi> i </mi>
    <none/>
    <none/>
    <mi> j </mi>
    <mi> k </mi>
    <none/>
    <mi> l </mi>
    <none/>
  </mmultiscripts>
</math>
```

The *mmultiscripts* element can also be used to attach prescripts to a symbol. The next example demonstrates this. The empty *prescripts* element signals the start of the prescripts section.

$${}^{427}Qb_4$$

```
<math>
  <mmultiscripts>
    <mi> Qb </mi>
    <mn> 4 </mn>
    <none/>
    <mprescripts/>
    <mn> 427 </mn>
    <none/>
  </mmultiscripts>
</math>
```

mspace
Currently not all functionality of the *mspace* element is implemented. Over time we will see what support is needed and makes sense, especially since this command can spoil things. We only support the units that make sense, so units in terms of pixels —a rather persistent oversight in drafts— are kindly ignored.

$$\text{use}\quad\text{me}\,\text{with}\hspace{10pt}\text{care}$$

```
<math>
  <mrow>
    <mtext> use  </mtext> <mspace width="1em" />
    <mtext> me   </mtext> <mspace width="1ex" />
    <mtext> with </mtext> <mspace width="10pt"/>
    <mtext> care </mtext>
  </mrow>
</math>
```

mphantom
A phantom element hides its content but still takes its space. A phantom element can contain other elements.

$$\text{who is afraid of}\ \phantom{\text{phantom}}\ \text{elements}$$

```
<math>
  <mrow>
    <mtext>    who is afraid of </mtext>    <mspace width=".5em" />
    <mphantom> phantom          </mphantom> <mspace width=".5em" />
    <mtext>    elements         </mtext>
  </mrow>
</math>
```

mpadded
As with a few other elements, I first have to see some practical usage for this, before I implement the functionality needed.

mtable, mtr, mtd, mlabeledtr
As soon as you want to represent a matrix or other more complicated composed constructs, you end up with spacing problems. This is when tables come into view. Because presentational elements have no deep knowledge about their content, tables made with presentational MATHML will in most cases look worse than those that result from content markup.

We have implemented tables on top of the normal XML (HTML) based table support in CONTEXT, also known as natural tables. Depending on the needs, support for the *mtable* element will be extended.

The *mtable* element takes a lot of attributes. When no attributes are given, we assume that a matrix is wanted, and typeset the content accordingly.

$$(\begin{matrix} x_{1,1} & 1 & 0 \\ 0 & x_{2,2} & 1 \\ 0 & 1 & x_{3,3} \end{matrix})$$

```
<math>
  <mrow>
    <mo> ( </mo>
    <mtable>
      <mtr>
        <mtd> <msub> <mi> x </mi> <mn> 1,1 </mn> </msub> </mtd>
        <mtd> <mn> 1 </mn> </mtd>
        <mtd> <mn> 0 </mn> </mtd>
      </mtr>
      <mtr>
        <mtd> <mn> 0 </mn> </mtd>
        <mtd> <msub> <mi> x </mi> <mn> 2,2 </mn> </msub> </mtd>
        <mtd> <mn> 1 </mn> </mtd>
      </mtr>
      <mtr>
        <mtd> <mn> 0 </mn> </mtd>
        <mtd> <mn> 1 </mn> </mtd>
        <mtd> <msub> <mi> x </mi> <mn> 3,3 </mn> </msub> </mtd>
      </mtr>
    </mtable>
    <mo> ) </mo>
  </mrow>
</math>
```

$$\begin{array}{lcr} \boxed{100} & 100 & 100 \\ 10 & \boxed{10} & 10 \\ 1 & 1 & \boxed{1} \end{array}$$

```
<math>
  <mtable columnalign="left center right" color="red blue black">
    <mtr>
      <mtd frame="on"> <mn> 100 </mn> </mtd>
      <mtd           > <mn> 100 </mn> </mtd>
      <mtd           > <mn> 100 </mn> </mtd>
    </mtr>
    <mtr>
      <mtd           > <mn> 10  </mn> </mtd>
      <mtd frame="on"> <mn> 10  </mn> </mtd>
      <mtd           > <mn> 10  </mn> </mtd>
    </mtr>
    <mtr>
      <mtd           > <mn> 1   </mn> </mtd>
      <mtd           > <mn> 1   </mn> </mtd>
      <mtd frame="on"> <mn> 1   </mn> </mtd>
    </mtr>
  </mtable>
</math>
```

Although the underlying table mechanism can provide all the support needed (and even more), not all attributes are yet implemented. We will make a useful selection.

columnalign	keyword: left center (middle) right
columnspacing	a meaningful dimension
rowspacing	a meaningful dimension
frame	keyword: none (off) solid (on)
color	a named color identifier
background	a named color identifier

We only support properly named colors as back- and foreground colors. The normal ConTEXt color mapping mechanism can be used to remap colors. This permits (read:

forces) a consistent usage of colors. If you use named backgrounds . . . the sky is the limit.

malignmark
This element is used in tables and is not yet implemented, first because I still have to unravel its exact usage, but second, because it is about the ugliest piece of MATHML markup you will encounter.

mglyph
This element is for those who want to violate the ideas of general markup by popping in his or hers own glyphs. Of course one should use entities, even if they have to be defined.

▶ + ▶ = ⏩

```
<math>
  <mrow>
    <mi> <mglyph fontfamily="navifont" index="2" alt="right"/>    </mi>
    <mo> + </mo>
    <mi> <mglyph fontfamily="navifont" index="2" alt="right"/>    </mi>
    <mo> = </mo>
    <mi> <mglyph fontfamily="navifont" index="6" alt="veryright"/></mi>
  </mrow>
</math>
```

mstyle
This element is implemented but not yet discussed since I want more control over its misuse.

afterword
You may have noticed that we prefer content MATHML over presentational MATHML. So, unless you're already sick of any math coded in angle brackets, we invite you to read the next chapter too.

Content markup

In this chapter we will discuss the MATHML elements from the point of view of typesetting. We will not pay attention to other rendering techniques, like speech generation. Some elements take attributes and those often make more sense for other applications than for a typesetting engine like TEX, which has a strong math engine that knows how to handle math.

apply
If you are dealing with rather ordinary math, you will only need a subset of content MATHML. For this reason we will start with the most common elements. When you key in XML directly, you will encounter the *apply* element quite often, even in a relatively short formula like the following.

```
<math> <apply> <minus/> <cn> 1 </cn> </apply> </math>
```

In most cases the *apply* element is followed by a specification disguised as an empty element.

ci, cn, sep
These elements are used to specify identifiers and numbers. Both elements can be made more explicit by using attributes.

`type`	`set`	use a representation appropriate for sets
	`vector`	mark this element as vector

`function`	consider this element to be a function
`fn`	idem

When *set* is specified, a blackboard symbol is used when available.

$$x \in \mathbb{N}$$

```
<math>
  <apply> <in/>
    <ci> x </ci>
    <ci type="set"> N </ci>
  </apply>
</math>
```

The *function* specification makes sense when the *ci* element is used in for instance a differential equation.

`type`	`integer`	a whole number with an optional base
	`logical`	a boolean constant
	`rational`	a real number
	`complex-cartesian`	a complex number in $x + iy$ notation
	`complex`	idem
	`complex-polar`	a complex number in polar notation ...

You're lucky when your document uses decimal notation, otherwise you will end up with long specs if you want to be clear in what numbers are used.

$$\mathrm{1A2C}_{16} + 0101_{16} = \mathrm{1B2D}_{16}$$

```
<math>
  <apply> <eq/>
    <apply> <plus/>
      <cn type="integer" base="16"> 1A2C </cn>
      <cn type="integer" base="16"> 0101 </cn>
    </apply>
    <cn type="integer" base="16"> 1B2D </cn>
  </apply>
</math>
```

Complex numbers have two components. These are separated by the *sep* element. In the following example we see that instead of using a *ci* with set specifier, the empty element *complexes* can be used. We will see some more of those later.

$$2 + 5\mathrm{i} \in \mathbb{C}$$

```
<math>
  <apply> <in/>
    <cn type="complex"> 2 <sep/> 5 </cn>
    <complexes/>
  </apply>
</math>
```

eq, neq, gt, lt, geq, leq

Expressions, and especially those with *eq* are typical for math. Because such expressions can be quite large, there are provisions for proper alignment.

lt	$a < b$	leq	$a \leq b$
eq	$a = b$	neq	$a \neq b$
gt	$a > b$	geq	$a \geq b$

$$a \leq b \leq c$$

```
<math>
  <apply> <leq/>
    <ci> a </ci>
    <ci> b </ci>
    <ci> c </ci>
  </apply>
</math>
```

equivalent, approx, implies
Equivalence, approximations, and implications are handled like *eq* and alike and have their own symbols.

$$a + b \equiv b + a$$

```
<math>
  <apply> <equivalent/>
    <apply> <plus/> <ci> a </ci> <ci> b </ci> </apply>
    <apply> <plus/> <ci> b </ci> <ci> a </ci> </apply>
  </apply>
</math>
```

This document is typeset with PDFTEX version 3.14159, and given that TEX is written by a mathematician, it will be no surprise that:

$$3.14159 \approx \pi$$

```
<math>
  <apply> <approx/>
    <cn> 3.14159 </cn>
    <pi/>
  </apply>
</math>
```

$$x + 4 = 9 \Rightarrow x = 5$$

```
<math>
  <apply> <implies/>
    <apply> <eq/>
      <apply> <plus/>
        <ci> x </ci>
        <cn> 4 </cn>
      </apply>
      <cn> 9 </cn>
    </apply>
    <apply> <eq/>
      <ci> x </ci>
      <cn> 5 </cn>
    </apply>
  </apply>
</math>
```

minus, plus
Addition and subtraction are main building blocks of math so you will meet them often.

$$37 - x$$

```
<math>
  <apply> <minus/>
    <cn> 37 </cn>
    <ci> x </ci>
  </apply>
</math>
```

In most cases there will be more than one argument to take care of, but especially *minus* will be used with one argument too. Although `<cn> -37 </cn>` is valid, using *minus* is sometimes more clear.

$$-37$$

```
<math>
  <apply> <minus/>
    <cn> 37 </cn>
  </apply>
</math>
```

You should pay attention to combinations of *plus* and *minus*. Opposite to presentational MathML, in content markup you don't think and code sequential.

$$-x + 37$$

```
<math>
  <apply> <plus/>
    <apply> <minus/>
      <ci> x </ci>
    </apply>
    <cn> 37 </cn>
  </apply>
</math>
```

times

Multiplication is another top ten element. Although `3p` as content of the *ci* element would have rendered the next example as well, you really should split off the number and mark it as *cn*. When this is done consistently, we can comfortably change the font of numbers independent of the font used for displaying identifiers.

$$3p$$

```
<math>
  <apply> <times/>
    <cn> 3 </cn>
    <ci> p </ci>
  </apply>
</math>
```

In a following chapter we will see how we can add multiplication signs between variables and constants.

divide

When typeset, a division is characterized by a horizontal rule. Some elements, like the differential element *diff*, generate their own division.

This example also demonstrates how to mix *plus* and *minus*.

```
[file pc-s-001.xml does not exist]
```

power
In presentational MATHML you think in super- and subscripts, but in content MATHML these elements are not available. There you need to think in terms of *power*.

$$x^2 + \sin^2 x$$

```
<math>
  <apply> <plus/>
    <apply> <power/>
      <ci> x </ci>
      <cn> 2 </cn>
    </apply>
    <apply> <power/>
      <apply> <sin/>
        <ci> x </ci>
      </apply>
      <cn> 2 </cn>
    </apply>
  </apply>
</math>
```

The *power* element is clever enough to determine where the superscript should go. In the case of the sinus function, by default it will go after the function identifier.

root, degree
If you study math related DTD's —this are the formal descriptions for SGML or XML element collections— you will notice that there are not that many elements that demand a special kind of typography: differential equations, limits, integrals and roots are the most distinctive ones.

$$\sqrt[3]{64} = 4$$

```
<math>
  <apply> <eq/>
    <apply> <root/>
      <degree> 3 </degree>
      <ci> 64 </ci>
    </apply>
    <cn> 4 </cn>
  </apply>
</math>
```

Contrary to *power*, the *root* element uses a specialized child element to denote the degree. The positive consequence of this is that there cannot be a misunderstanding about what role the child element plays, while in for instance *power* you need to know that the second child element denotes the degree.

sin, cos, tan, cot, scs, sec, . . .
All members of the family of goniometric functions are available as empty element. When needed, their argument is surrounded by braces. They all behave the same.

sin	arcsin	sinh	arcsinh
cos	arccos	cosh	arccosh
tan	arctan	tanh	arctanh
cot	arccot	coth	arccoth
csc	arccsc	csch	arccsch
sec	arcsec	sech	arcsech

These functions are normally typeset in a non italic (often roman) font shape.

By default the typesetting engine will minimize the number of braces that surrounds the argument of a function.

```
[file wh-g-001.xml does not exist]
```

You can specify π as an entity *pi* (coded as `&pi;`) or as empty element *pi*. In many cases it is up to your taste which one you use. There are many symbols that are only available as entity, so in some respect there is no real reason to treat π different.

$$\cos\pi = -1$$

```
<math>
  <apply> <eq/>
    <apply> <cos/>
      <pi/>
    </apply>
    <apply> <minus/>
      <cn> 1 </cn>
    </apply>
  </apply>
</math>
```

log, ln, exp

The *log* and *ln* are typeset similar to the previously discussed goniometric functions. The *exp* element is a special case of *power*. The constant e can be specified with *exponentiale*.

$$\ln(\mathrm{e}+2) \approx 1.55$$

```
<math>
  <apply> <approx/>
    <apply> <ln/>
      <apply> <plus/>
        <exponentiale/>
        <cn> 2 </cn>
      </apply>
    </apply>
    <cn> 1.55 </cn>
  </apply>
</math>
```

$$\mathrm{e}^2 = 7.3890560989307$$

```
<math>
  <apply> <eq/>
    <apply> <exp/>
      <cn> 2 </cn>
    </apply>
    <cn> 7.3890560989307 </cn>
  </apply>
</math>
```

quotient, rem

The result of a division can be a rational number, so $\frac{5}{4}$ is equivalent to 1.25 and 1.25×4 gives 5. An integer division will give 1 with a remainder 2. Many computer languages provide a `div` and `mod` function, and since MATHML is also meant for computation, it provides similar concepts, represented by the elements *quotient* and *rem*. The representation of *quotient* is rather undefined, but the next one is among the recommended alternatives.

$$\lfloor a/b \rfloor$$

```
<math>
  <apply> <quotient/>
    <ci> a </ci>
    <ci> b </ci>
  </apply>
</math>
```

factorial
Showing the representation of a factorial is rather dull, so we will use a few more elements as well as a processing instruction to illustrate the usage of *factorial*.

$$n! = n \times (n-1) \times (n-2) \times \cdots \times 1$$

```
<math>
  <?context-mathml-directive times symbol yes ?>
  <apply> <eq/>
    <apply> <factorial/>
      <ci> n </ci>
    </apply>
    <apply> <times/>
      <ci> n </ci>
      <apply> <minus/> <ci> n </ci> <cn> 1 </cn> </apply>
      <apply> <minus/> <ci> n </ci> <cn> 2 </cn> </apply>
      <ci> &cdots; </ci>
      <cn> 1 </cn>
    </apply>
  </apply>
</math>
```

The processing instruction is responsible for the placement of the $\times$ symbols.

min, max, gcd, lcm
These functions can handle more than two arguments. When typeset, these are separated by comma's.

$$z = \min \left\{ (x+y)\,, 2x\,, \frac{1}{y} \right\}$$

```
<math>
  <apply> <eq/>
    <ci> z </ci>
    <apply> <min/>
      <apply> <plus/>   <ci> x </ci> <ci> y </ci> </apply>
      <apply> <times/>  <cn> 2 </cn> <ci> x </ci> </apply>
      <apply> <divide/> <cn> 1 </cn> <ci> y </ci> </apply>
    </apply>
  </apply>
</math>
```

and, or, xor, not
Logical expressions can be defined using these elements. The operations are represented by symbols and braces are applied when needed.

$$1001_2 \wedge 0101_2 = 0001_2$$

```
<math>
  <apply> <eq/>
    <apply> <and/>
      <cn type="integer" base="2"> 1001 </cn>
      <cn type="integer" base="2"> 0101 </cn>
    </apply>
    <cn type="integer" base="2"> 0001 </cn>
  </apply>
</math>
```

set, bvar
The appearance of a *set* depends on the presence of the child element *bvar*. In its simplest form, a set is represented as a list.

$$\{1,4,8\} \neq \emptyset$$

```
<math>
  <apply> <neq/>
    <set>
      <cn> 1 </cn>
      <cn> 4 </cn>
      <cn> 8 </cn>
    </set>
    <emptyset/>
  </apply>
</math>
```

A set can be distinguished from a vector by its curly braces. The simplest case is just a comma separated list. The next example demonstrates the declarative case. Without doubt, there will be other alternatives.

$$\{x \mid 2 < x < 8\}$$

```
<math>
  <set>
    <bvar><ci> x </ci></bvar>
    <condition>
      <apply> <lt/>
        <cn> 2 </cn>
        <ci> x </ci>
        <cn> 8 </cn>
      </apply>
    </condition>
  </set>
</math>
```

list
This element is used in different contexts. When used as a top level element, a list is typeset as follows.

$$[1,1,3]$$

```
<math>
  <list>
    <cn> 1 </cn>
    <cn> 1 </cn>
    <cn> 3 </cn>
  </list>
</math>
```

When used in a context like *partialdiff*, the list specification becomes a subscript.

$$D_{1,1,3}f$$

```
<math>
  <apply> <partialdiff/>
    <list>
      <cn> 1 </cn>
      <cn> 1 </cn>
      <cn> 3 </cn>
    </list>
    <ci type="fn"> f </ci>
  </apply>
</math>
```

The function specification in this formula (which is taken from the specs) can also be specified as `<fn> <ci> f </ci> </fn>` (which is more clear).

union, intersect, . . .
There is a large number of set operators, each represented by a distinctive symbol.

union	$U \cup V$		
intersect	$U \cap V$		
in	$U \in V$	notin	$U \notin V$
subset	$U \subset V$	notsubset	$U \not\subset V$
prsubset	$U \subseteq V$	notprsubset	$U \not\subseteq V$
setdiff	$U \setminus V$		

These operators are applied as follows:

$$U \cup V$$

```
<math>
  <apply> <union/>
    <ci> U </ci>
    <ci> V </ci>
  </apply>
</math>
```

conjugate, arg, real, imaginary
The visual representation of *conjugate* is a horizontal bar with a width matching the width of the expression.

$$\overline{x + \mathrm{i}y}$$

```
<math>
  <apply> <conjugate/>
    <apply> <plus/>
      <ci> x </ci>
      <apply> <times/>
        <cn> &ImaginaryI; </cn>
        <ci> y </ci>
      </apply>
    </apply>
  </apply>
</math>
```

The *arg*, *real* and *imaginary* elements trigger the following appearance.

$$\arg(x + \mathrm{i}y)$$

$$\Re(x + \mathrm{i}y)$$

$$\Im(x + \mathrm{i}y)$$

abs, floor, ceiling
There are a couple of functions that turn numbers into positive or rounded ones. In computer languages names are used, but in math we use special boundary characters.

$$|-5| = 5$$

$$\lfloor 5.5 \rfloor = 5$$

$$\lceil 5.5 \rceil = 6$$

```
<math>
  <apply> <eq/>
    <apply> <abs/> <cn> -5 </cn> </apply>
    <cn> 5 </cn>
  </apply>
</math>
<math>
  <apply> <eq/>
    <apply> <floor/> <cn> 5.5 </cn> </apply>
    <cn> 5 </cn>
  </apply>
</math>
<math>
  <apply> <eq/>
    <apply> <ceiling/> <cn> 5.5 </cn> </apply>
    <cn> 6 </cn>
  </apply>
</math>
```

interval
An interval is visualized as: $[1, 10]$. The *interval* element is a container element and has a begin and endtag. You can specify the closure as attribute:

$$(a, b]$$

```
<math>
  <interval closure="open-closed">
    <ci> a </ci>
    <ci> b </ci>
  </interval>
</math>
```

The following closures are supported:

open	(a, b)
closed	$[a, b]$
open-closed	$(a, b]$
closed-open	$[a, b)$

inverse

This operator is applied to a function. The following example demonstrates that this is one of the few cases (if not the only one) where the first element following an *apply* begintag is an *apply* itself.

$$\sin^{-1} x$$

```
<math>
  <apply>
    <apply> <inverse/> <sin/> </apply>
    <ci> x </ci>
  </apply>
</math>
```

reln

This element is a left–over from the first MATHML specification and its usage is no longer advocated. Its current functionality matches the functionality of *apply*.

cartesianproduct, vectorproduct, scalarproduct, outerproduct

Often the context of the formula will provide information of what kind of multiplication is meant, but using different symbols to represent the kind of product certainly helps.

$$a \times b$$

```
<math>
  <apply> <cartesianproduct/>
    <ci> a </ci>
    <ci> b </ci>
  </apply>
</math>
```

We have:

cartesian	$a \times b$
vector	$a \times b$
scalar	$a \cdot b$
outer	$a \otimes b$

sum, product, limit, lowlimit, uplimit, bvar

Sums, products and limits have a distinctive look, especially when they have upper and lower limits attached. Unfortunately there is no way to specify the x_i in content MATHML. In the next chapter we will see how we can handle that.

$$\sum_{i=1}^{n} \frac{1}{x}$$

```
<math>
  <apply> <sum/>
    <bvar> <ci> i </ci> </bvar>
    <lowlimit> <cn> 1 </cn> </lowlimit>
    <uplimit> <ci> n </ci> </uplimit>
    <apply> <divide/>
      <cn> 1 </cn>
      <ci> x </ci>
    </apply>
  </apply>
</math>
```

When we omit the limits, the *bvar* is still typeset.

$$\prod_{i} \frac{1}{x}$$

```
<math>
  <apply> <product/>
    <bvar>
      <ci> i </ci>
    </bvar>
    <apply> <divide/>
      <cn> 1 </cn>
      <ci> x </ci>
    </apply>
  </apply>
</math>
```

You can specify the condition under which the function is applied.

$$\prod_{x \in \mathbb{R}} f(x)$$

```
<math>
  <apply> <product/>
    <bvar>
      <ci> x </ci>
    </bvar>
    <condition>
      <apply> <in/>
        <ci> x </ci>
        <ci type="set"> R </ci>
      </apply>
    </condition>
    <apply> <ci type="fn"> f </ci>
      <ci> x </ci>
    </apply>
  </apply>
</math>
```

$$\lim_{x \rightarrow 0} \sin x$$

```
<math>
  <apply> <limit/>
    <bvar>
      <ci> x </ci>
    </bvar>
    <lowlimit>
      <cn> 0 </cn>
    </lowlimit>
    <apply> <sin/>
      <ci> x </ci>
    </apply>
  </apply>
</math>
```

int, diff, partialdiff, bvar, degree

These elements reach a high level of abstraction. The best way to learn how to use them is to carefully study some examples.

$$\frac{\mathrm{d}\int_p^q f(x,a)\,\mathrm{d}x}{\mathrm{d}a}$$

```
<math>
  <apply> <diff/>
    <bvar> <ci> a </ci> </bvar>
    <apply> <int/>
      <lowlimit> <ci> p </ci> </lowlimit>
      <uplimit>  <ci> q </ci> </uplimit>
      <bvar> <ci> x </ci> </bvar>
      <apply>
        <fn> <ci> f </ci> </fn>
        <ci> x </ci>
        <ci> a </ci>
      </apply>
    </apply>
  </apply>
</math>
```

The *bvar* element is essential, since it is used to automatically generate some of the components that make up the visual appearance of the formula. If you look at the formal specification of these elements, you will notice that the appearance may depend on your definition. How the formula shows up, depends not only on the *bvar* element, but also on the optional *degree* element within.

$$f'$$

```
<math>
  <apply> <diff/>
    <ci> f </ci>
  </apply>
</math>
```

$$\frac{\mathrm{d}^2 f(x)}{\mathrm{d}x^2}$$

```
<math>
  <apply> <diff/>
    <bvar>
      <ci> x </ci>
      <degree> <cn> 2 </cn> </degree>
    </bvar>
    <apply> <fn> <ci> f </ci> </fn>
      <ci> x </ci>
    </apply>
  </apply>
</math>
```

$$\frac{\partial^4 f}{\partial x^2 \partial y \partial x}$$

```
<math>
  <apply> <partialdiff/>
    <bvar>
      <degree> <cn> 2 </cn> </degree>
      <ci> x </ci>
    </bvar>
    <bvar> <ci> y </ci> </bvar>
    <bvar> <ci> x </ci> </bvar>
    <degree> <cn> 4 </cn> </degree>
    <ci type="fn"> f </ci>
  </apply>
</math>
```

$$\frac{\partial^k f(x,y)}{\partial x^m \partial y^n}$$

```
<math>
  <apply> <partialdiff/>
    <bvar>
      <ci> x </ci> <degree> <ci> m </ci> </degree>
    </bvar>
    <bvar>
      <ci> y </ci> <degree> <ci> n </ci> </degree>
    </bvar>
    <degree> <ci> k </ci> </degree>
    <apply> <ci type="fn"> f </ci>
      <ci> x </ci>
      <ci> y </ci>
    </apply>
  </apply>
</math>
```

$$\frac{\partial^{m+n} f(x,y)}{\partial x^m \partial y^n}$$

```
<math>
  <apply> <partialdiff/>
    <bvar>
      <ci> x </ci> <degree> <ci> m </ci> </degree>
    </bvar>
    <bvar>
      <ci> y </ci> <degree> <ci> n </ci> </degree>
    </bvar>
    <apply> <ci type="fn"> f </ci>
      <ci> x </ci>
      <ci> y </ci>
    </apply>
  </apply>
</math>
```

When a degree is not specified, it is deduced from the context, but since this is not 100% watertight, you can best be complete in your specification.

These examples are taken from the MATHML specification. In the example document that comes with this manual you can find a couple of more.

fn
There are a lot of predefined functions and operators. If you want to introduce a new one, the *fn* element can be used. In the following example we have turned the $\pm$ and $\mp$ symbols into (coupled) operators.

$$x \pm 1\, x \mp 1 = x^2 - 1$$

```
<math>
  <apply> <eq/>
    <apply> <times/>
      <apply> <fn> <ci> &plusminus; </ci> </fn>
        <ci> x </ci>
        <cn> 1 </cn>
      </apply>
      <apply> <fn> <ci> &minusplus; </ci> </fn>
        <ci> x </ci>
        <cn> 1 </cn>
      </apply>
    </apply>
    <apply> <minus/>
      <apply> <power/>
        <ci> x </ci>
        <cn> 2 </cn>
      </apply>
      <cn> 1 </cn>
    </apply>
  </apply>
</math>
```

The typeset result depends on the presence of a handler, which in this case happens to be true.

matrix, matrixrow
Matrices are one of the building blocks of linear algebra and therefore both presentational and content MATHML have dedicated elements for defining them.

$$\begin{pmatrix} 23 & 87 & c \\ 41 & b & 33 \\ a & 65 & 16 \end{pmatrix}$$

```
<math>
  <matrix>
    <matrixrow> <cn> 23 </cn> <cn> 87 </cn> <ci>  c </ci> </matrixrow>
    <matrixrow> <cn> 41 </cn> <ci>  b </ci> <cn> 33 </cn> </matrixrow>
    <matrixrow> <ci>  a </ci> <cn> 65 </cn> <cn> 16 </cn> </matrixrow>
  </matrix>
</math>
```

vector
We make a difference between a vector specification and a vector variable. A specification is presented as a list:

$$(x,y)$$

```
<math>
  <vector>
    <ci> x </ci>
    <ci> y </ci>
  </vector>
</math>
```

When the *vector* element has one child element, we use a right arrow to identify the variable as vector.

$$\vec{A} \times \vec{B}$$

```
<math>
  <apply> <vectorproduct/>
    <vector> <ci> A </ci> </apply>
    <vector> <ci> B </ci> </apply>
  </vector>
</math>
```

grad, curl, ident, divergence

These elements expand into named functions, but we can imagine that in the future a more appropriate visualization will be provided as an option.

$$\operatorname{grad} A \neq \operatorname{curl} B \neq \operatorname{id} C \neq \operatorname{div} D$$

```
<math>
  <apply> <neq/>
    <apply> <grad/>       <ci> A </ci> </apply>
    <apply> <curl/>       <ci> B </ci> </apply>
    <apply> <ident/>      <ci> C </ci> </apply>
    <apply> <divergence/> <ci> D </ci> </apply>
  </apply>
</math>
```

lambda, bvar

The lambda specification of a function needs a *bvar* element. The visualization can be influenced with processing instructions as described in a later chapter.

$$x \mapsto \sin\left(x - \frac{x}{2}\right)$$

```
<math>
  <lambda>
    <bvar> <ci> x </ci> </bvar>
    <apply> <sin/>
      <apply> <minus/>
        <ci> x </ci>
        <apply> <divide/>
          <ci> x </ci>
          <cn> 2 </cn>
        </apply>
      </apply>
    </apply>
  </lambda>
</math>
```

piecewise, piece, otherwise

There are not so many elements that deal with combinations of formulas or conditions. The *piecewise* is the only real selector available. The following example defines how the state of n depends on the state of x.

$$n = \begin{cases} -1 & x < 0 \\ 1 & x > 0 \\ 0 & \text{otherwise} \end{cases}$$

```
<math>
  <apply> <eq/>
    <ci> n </ci>
    <piecewise>
      <piece>
        <apply> <minus/>
          <cn> 1 </cn>
        </apply>
        <apply> <lt/>
          <ci> x </ci>
          <cn> 0 </cn>
        </apply>
      </piece>
      <piece>
        <cn> 1 </cn>
        <apply> <gt/>
          <ci> x </ci>
          <cn> 0 </cn>
        </apply>
      </piece>
      <otherwise>
        <cn> 0 </cn>
      </otherwise>
    </piecewise>
  </apply>
</math>
```

We could have use a third *piece* instead of (optional) *otherwise*.

forall, exists, condition
Conditions are often used in combination with elements like *forall*. There are several ways to convert and combine them in formulas and environments, so you may expect more alternatives in the future.

$$\forall_x \; x < 9 \,|\, x < 10$$

```
<math>
  <apply> <forall/>
    <bvar> <ci> x </ci> </bvar>
    <condition>
      <apply> <lt/>
        <ci> x </ci>
        <cn> 9 </cn>
      </apply>
    </condition>
    <apply> <lt/>
      <ci> x </ci>
      <cn> 10 </cn>
    </apply>
  </apply>
</math>
```

The next example is taken from the specifications with a few small changes.

$$\forall_x \;\; x \in \mathbb{N} \,|\, \exists_{p,q} \;\; p \in \mathbb{P} \wedge q \in \mathbb{P} \wedge p + q = 2x$$

```
<math>
  <apply> <forall/>
    <bvar> <ci> x </ci> </bvar>
    <condition>
      <apply> <in/>
        <ci> x </ci>
        <ci type="set"> N </ci>
      </apply>
    </condition>
    <apply> <exists/>
      <bvar> <ci> p </ci> </bvar>
      <bvar> <ci> q </ci> </bvar>
      <condition>
        <apply> <and/>
          <apply> <in/>
            <ci> p </ci>
            <ci type="set"> P </ci>
          </apply>
          <apply> <in/>
            <ci> q </ci>
            <ci type="set"> P </ci>
          </apply>
          <apply> <eq/>
            <apply> <plus/> <ci> p </ci> <ci> q </ci> </apply>
            <apply> <times/> <cn> 2 </cn> <ci> x </ci> </apply>
          </apply>
        </apply>
      </condition>
    </apply>
  </apply>
</math>
```

factorof, tendsto

The *factorof* element is applied to its two child elements and contrary to most functions, the symbol is placed between the elements instead of in front.

$$a \mid b$$

```
<math>
  <apply> <factorof/>
    <ci> a </ci>
    <ci> b </ci>
  </apply>
</math>
```

The same is true for the *tendsto* element.

$$a \rightarrow b$$

```
<math>
  <apply> <tendsto/>
    <ci> a </ci>
    <ci> b </ci>
  </apply>
</math>
```

compose
This is a nasty element since it has to take care of braces in special ways and therefore has to analyse its child elements.

$$f \circ g \circ h$$

```
<math>
  <apply> <compose/>
    <ci type="fn"> f </ci>
    <ci type="fn"> g </ci>
    <ci type="fn"> h </ci>
  </apply>
</math>
```

$$(f \circ g)\, x$$

```
<math>
  <apply>
    <apply> <compose/>
      <fn> <ci> f </ci> </fn>
      <fn> <ci> g </ci> </fn>
    </apply>
    <ci> x </ci>
  </apply>
</math>
```

laplacian
A laplacian function is typeset using a ∇ (nabla) symbol.

$$\nabla^2 x$$

```
<math>
  <apply> <laplacian/>
    <ci> x </ci>
  </apply>
</math>
```

mean, sdev, variance, median, mode
When statistics shows up in math text books, the *sum* element is likely to show up, probably in combination with the for statistics meaningful symbolic representation of variables. The mean value of a series of observations is defined as:

$$\overline{x} = \frac{\sum x}{n}$$

```
<math>
  <apply> <eq/>
    <apply> <mean/>
      <ci> x </ci>
    </apply>
    <apply> <divide/>
      <apply> <sum/>
        <ci> x </ci>
      </apply>
      <ci> n </ci>
    </apply>
  </apply>
</math>
```

or more beautiful:

$$\overline{x} = \frac{1}{n} \sum x$$

```
<math>
  <apply> <eq/>
    <apply> <mean/>
      <ci> x </ci>
    </apply>
    <apply> <times/>
      <apply> <divide/>
        <cn> 1 </cn>
        <ci> n </ci>
      </apply>
      <apply> <sum/>
        <ci> x </ci>
      </apply>
    </apply>
  </apply>
</math>
```

Of course this definition is not that perfect, but we will present a better alternative in the chapter on combined markup. The definition of the standard deviation is more complicated:

$$\sigma(x) \approx \sqrt{\frac{\sum (x - \overline{x})^2}{n - 1}}$$

```
<math>
  <apply> <approx/>
    <apply> <sdev/>
      <ci> x </ci>
    </apply>
    <apply> <root/>
      <apply> <divide/>
        <apply> <sum/>
          <apply> <power/>
            <apply> <minus/>
              <ci> x </ci>
              <apply> <mean/>
                <ci> x </ci>
              </apply>
            </apply>
            <cn> 2 </cn>
          </apply>
        </apply>
        <apply> <minus/>
          <ci> n </ci>
          <cn> 1 </cn>
        </apply>
      </apply>
    </apply>
  </apply>
</math>
```

The next example demonstrates the usage of the *variance* in its own definition.

$$\sigma(x)^2 = \overline{(x-\overline{x})^2} \approx \frac{1}{n-1}\sum(x-\overline{x})^2$$

```
<math>
  <apply> <eq/>
    <apply> <variance/>
      <ci> x </ci>
    </apply>
    <apply> <approx/>
      <apply> <mean/>
        <apply> <power/>
          <apply> <minus/>
            <ci> x </ci>
            <apply> <mean/>
              <ci> x </ci>
            </apply>
          </apply>
          <cn> 2 </cn>
        </apply>
      </apply>
      <apply> <times/>
        <apply> <divide/>
          <cn> 1 </cn>
          <apply> <minus/>
            <ci> n </ci>
            <cn> 1 </cn>
          </apply>
        </apply>
        <apply> <sum/>
          <apply> <power/>
            <apply> <minus/>
              <ci> x </ci>
              <apply> <mean/>
                <ci> x </ci>
              </apply>
            </apply>
            <cn> 2 </cn>
          </apply>
        </apply>
      </apply>
    </apply>
  </apply>
</math>
```

The *median* and *mode* of a series of observations have no special symbols and are presented as is.

moment, momentabout, degree

Because MathML is used for a wide range of applications, there can be information in a definition that does not end up in print but is only used in some cases. This is illustrated in the next example.

$$\left\langle X^3 \right\rangle$$

```
<math>
  <apply> <moment/>
    <degree>
      <cn> 3 </cn>
    </degree>
    <momentabout>
      <ci> p </ci>
    </momentabout>
    <ci> X </ci>
  </apply>
</math>
```

determinant, transpose, selector

This threesome is used to manipulate matrices, either or not in a symbolic way. A simple determinant or transpose looks like:

$$|\boldsymbol{A}|$$

```
<math>
  <apply> <determinant/>
    <ci type="matrix"> A </ci>
  </apply>
</math>
```

$$\boldsymbol{A}^{\mathrm{T}}$$

```
<math>
  <apply> <transpose/>
    <ci type="matrix"> A </ci>
  </apply>
</math>
```

When the *determinant* element is applied to full blown matrix, the braces are omitted and replaced by the vertical bars.

```
[file wh-m-002.xml does not exist]
```

The *selector* element honors the braces.

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}_1$$

```
<math>
  <apply> <selector/>
    <matrix>
      <matrixrow> <cn> 1 </cn> <cn> 2 </cn> </matrixrow>
      <matrixrow> <cn> 3 </cn> <cn> 4 </cn> </matrixrow>
    </matrix>
    <cn> 1 </cn>
  </apply>
</math>
```

card
A cardinality is visualized using vertical bars. [But what exactly is meant with cardinality?]

$$|A| = 5$$

```
<math>
  <apply> <eq/>
    <apply> <card/>
      <ci> A </ci>
    </apply>
    <ci> 5 </ci>
  </apply>
</math>
```

domain, codomain, image
The next couple of examples are taken from the MathML specification and demonstrate the usage of the not that spectacular domain related elements.

$$\mathrm{domain}(f) = \mathbb{R}$$

```
<math>
  <apply> <eq/>
    <apply> <domain/>
      <fn> <ci> f </ci> </fn>
    </apply>
    <reals/>
  </apply>
</math>
```

These are typically situations where the *fn* element may show up.

$$\mathrm{codomain}(f) = \mathbb{Q}$$

```
<math>
  <apply> <eq/>
    <apply> <codomain/>
      <fn> <ci> f </ci> </fn>
    </apply>
    <rationals/>
  </apply>
</math>
```

This example from the MathML specification demonstrates a typical usage of the *image* element. As with the previous two, it is applied to a function, in this case the predefined *sin*.

$$\mathrm{image}(\sin) = [-1, 1]$$

```
<math>
  <apply> <eq/>
    <apply> <image/>
      <sin/>
    </apply>
    <interval>
      <cn> -1 </cn>
      <cn>  1 </cn>
    </interval>
  </apply>
</math>
```

domainofapplication

This is another seldom used element. Actually, this element is a further specification of the outer level applied function.

$$\int_C f$$

```
<math>
  <apply> <int/>
    <domainofapplication>
      <ci> C </ci>
    </domainofapplication>
    <ci> f </ci>
  </apply>
</math>
```

semantics, annotation, annotation-xml

We will never know what Albert Einstein would have thought about MATHML. But we do know for sure that coding one of his famous findings in XML takes much more tokens that it takes in TEX.

Within a *semantics* element there can be many *annotation* elements. When using CONTEXT, the elements that can be identified as being encoded in TEX will be treated as such. Currently, the related *annotation-xml* element is ignored.

$$e = mc^2$$

```
<math>
  <semantics>
    <apply> <eq/>
      <ci> e </ci>
      <apply> <times/>
        <ci> m </ci>
        <apply> <power/>
          <ci> c </ci>
          <cn> 2 </cn>
        </apply>
      </apply>
    </apply>
    <annotation encoding="TeX">
      e = m c^2
    </annotation>
  </semantics>
</math>
```

integers, reals, ...

Sets are characterized with special (often blackboard) symbols. These symbols are not always available.

integers	$\mathbb{Z}$
reals	$\mathbb{R}$
rationals	$\mathbb{Q}$
naturalnumbers	$\mathbb{N}$

complexes	$\mathbb{C}$
primes	$\mathbb{P}$

pi, imaginaryi, exponentiale

Being a greek character, π is a distinctive character. In most math documents the imaginary i and exponential e are typeset as any math identifier.

pi	π
imaginaryi	i
exponentiale	e

eulergamma, infinity, emptyset

There are a couple of more special tokens. As with the other ones, they can be changed by reassigning the corresponding entities.

eulergamma	γ
infinity	∞
emptyset	Ø

notanumber

Because MathML is used for more purposes than typesetting, there are a couple of elements that do not make much sense in print. One of these is *notanumber*, which is issued by programs as error code or string.

$$\frac{x}{0} = \text{NaN}$$

```
<math>
  <apply> <eq/>
    <apply> <divide/>
      <ci> x </ci>
      <cn> 0 </cn>
    </apply>
    <notanumber/>
  </apply>
</math>
```

true, false

When assigning to a boolean variable, or in boolean expressions one can use 0 or 1 to identify the states, but if you want to be more verbose, you can use these elements.

$$1_2 \equiv \text{true}$$

```
<math>
  <apply> <equivalent/>
    <cn type="integer" base="2"> 1 </cn>
    <true/>
  </apply>
</math>
```

declare

Reusing definitions would be a nice feature, but for the moment the formal specification of this element currently does not give us the freedom to use it the way we want.

$$\text{declare}\, A \ \text{ as } \ (a,b,c)$$

```
<math>
  <declare>
    <ci> A </ci>
    <vector>
      <ci> a </ci>
      <ci> b </ci>
      <ci> c </ci>
    </vector>
  </declare>
</math>
```

csymbol
This element will be implemented as soon as I have an application for it.

Mixed markup

The advantage of presentational markup is that you can build complicated formulas using super- and subscripts and other elements. The drawback is that the look and feel is rather fixed and cannot easily be adapted to the purpose that the document serves. Take for instance the difference between

$$\log_2 x$$

and

$${}^2\log x$$

Both formulas were defined in content MathML, so no explicit super- and subscripts were used. In the next chapter we will see how to achieve such different appearances.

There are situations where content MathML is not rich enough to achieve the desired output. This omission in content MathML forces us to fall back on presentational markup.

$$P_1 = P_2 = 1.01 \approx 1$$

Here we used presentational elements inside a content *ci* element. We could have omitted the outer *ci* element, but since the content MathML parser may base its decisions on the content elements it finds, it is best to keep the outer element there.

```
<math>
  <apply> <eq/>
    <ci> <msub> <mi> P </mi> <mi> 1 </mi> </msub> </ci>
    <ci> <msub> <mi> P </mi> <mi> 2 </mi> </msub> </ci>
    <apply> <approx/>
      <cn> 1.01 </cn>
      <cn> 1 </cn>
    </apply>
  </apply>
</math>
```

The lack of an index element can be quite prominent. For instance, when in an expose about rendering we want to explore the mapping from coordinates in user space to those in device space, we use the following formula.

$$\left(D_x, D_y, 1\right) = \left(U_x, U_y, 1\right) \begin{pmatrix} s_x & r_x & 0 \\ r_y & s_y & 0 \\ t_x & t_y & 1 \end{pmatrix}$$

```
<math>
  <apply> <eq/>
    <vector>
      <ci> <msub> <mi> D </mi> <mi> x </mi> </msub> </ci>
      <ci> <msub> <mi> D </mi> <mi> y </mi> </msub> </ci>
      <cn> 1 </cn>
    </vector>
    <apply> <times/>
      <vector>
        <ci> <msub> <mi> U </mi> <mi> x </mi> </msub> </ci>
        <ci> <msub> <mi> U </mi> <mi> y </mi> </msub> </ci>
        <cn> 1 </cn>
      </vector>
      <matrix>
        <matrixrow>
          <ci> <msub> <mi> s </mi> <mi> x </mi> </msub> </ci>
          <ci> <msub> <mi> r </mi> <mi> x </mi> </msub> </ci>
          <cn> 0 </cn>
        </matrixrow>
        <matrixrow>
          <ci> <msub> <mi> r </mi> <mi> y </mi> </msub> </ci>
          <ci> <msub> <mi> s </mi> <mi> y </mi> </msub> </ci>
          <cn> 0 </cn>
        </matrixrow>
        <matrixrow>
          <ci> <msub> <mi> t </mi> <mi> x </mi> </msub> </ci>
          <ci> <msub> <mi> t </mi> <mi> y </mi> </msub> </ci>
          <cn> 1 </cn>
        </matrixrow>
      </matrix>
    </apply>
  </apply>
</math>
```

Again, the *msub* element provides a way out, as in the next examples, which are adapted versions of formulas we used when demonstrating the statistics related elements.

$$\overline{x} = \frac{1}{n} \sum_{i} x$$

```
<math>
  <apply> <eq/>
    <apply> <mean/>
      <ci> x </ci>
    </apply>
    <apply> <times/>
      <apply> <divide/>
        <cn> 1 </cn>
        <ci> n </ci>
      </apply>
      <apply> <sum/>
        <bvar> <ci> i </ci> </bvar>
        <ci> x </ci>
      </apply>
    </apply>
  </apply>
</math>
```

$$\overline{x} = \frac{1}{n} \sum_{i=1}^{n} x$$

```
<math>
  <apply> <eq/>
    <apply> <mean/>
      <ci> x </ci>
    </apply>
    <apply> <times/>
      <apply> <divide/>
        <cn> 1 </cn>
        <ci> n </ci>
      </apply>
      <apply> <sum/>
        <bvar> <ci> i </ci> </bvar>
        <lowlimit> <cn> 1 </cn> </lowlimit>
        <uplimit> <cn> n </cn> </uplimit>
        <ci> x </ci>
      </apply>
    </apply>
  </apply>
</math>
```

$$\overline{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

```
<math>
  <apply> <eq/>
    <apply> <mean/>
      <ci> x </ci>
    </apply>
    <apply> <times/>
      <apply> <divide/>
        <cn> 1 </cn>
        <ci> n </ci>
      </apply>
      <apply> <sum/>
        <bvar> <ci> i </ci> </bvar>
        <lowlimit> <cn> 1 </cn> </lowlimit>
        <uplimit> <cn> n </cn> </uplimit>
        <ci> <msub> <mi> x </mi> <mi> i </mi> </msub> </ci>
      </apply>
    </apply>
  </apply>
</math>
```

Directives

Some elements can be tuned by changing their attributes. Especially when formulas are defined by a team of people or when they are taken from a repository, there is a good chance that inconsistencies will show up.

In ConTEXt, you can influence the appearance by setting the typesetting parameters of (classes of) elements. You can do this either by adding processing instructions, or by using the ConTEXt command `\setupMMLappearance`. Although the first method is more in the spirit of XML, the second method is more efficient and consistent. As a processing instruction, a directive looks like:

```
<?context-mathml-directive element key value ?>
```

This is equivalent to the ConTEXt command:

```
\setupMMLappearance [element] [key=value]
```

Some settings concern a group of elements, in which case a group classification (like `sign`) is used.

scripts

By default, nested super- and subscripts are kind of isolated from each other. If you want a combined script, there is the *msubsup*. You can however force combinations with a directive.

$${x_1}^2$$

$$x_1^2$$

```
<math>
  <msup>
    <msub> <mi> x </mi> <mn> 1 </mn> </msub>
    <mn> 2 </mn>
  </msup>
</math>
<?context-mathml-directive scripts alternative b ?>
<math>
  <msup>
    <msub> <mi> x </mi> <mn> 1 </mn> </msub>
    <mn> 2 </mn>
  </msup>
</math>
```

sign

The core element of MathML is *apply*. Even simple formulas will often have more than one (nested) *apply*. The most robust way to handle nested formulas is to use braces around each sub formula. No matter how robust this is, when presented in print we want to use as less braces as possible.

$$7 + 5 -3$$

This expression shows addition as well as subtraction.

```
<math>
  <apply> <plus/>
    <cn> 7 </cn>
    <cn> 5 </cn>
    <apply> <minus/>
      <cn> 3 </cn>
    </apply>
  </apply>
</math>
```

In principle subtraction is adding negated numbers, so it would have been natural to have just an addition (*plus*) and negation operator. However, MATHML provides both a *plus* and *minus* operator, where the latter can be used as a negation. So in fact we have:

$$7 + 5 + (-3)$$

Now imagine that a teacher wants to stress this negation in the way presented here, using parentheses. Since all the examples shown here are typeset directly from the MATHML source, you may expect a solution, so here it is:

```
<math>
  <?context-mathml-directive sign reduction no ?>
  <apply> <plus/>
    <cn> 7 </cn>
    <cn> 5 </cn>
    <apply> <minus/>
      <cn> 3 </cn>
    </apply>
  </apply>
</math>
```

By default signs are reduced, but one can disable that at the document and/or formula level using a processing instruction at the top of the formula. There are of course circumstances where the parentheses cannot be left out.

$$a + b + c + d$$

```
<math>
  <apply> <plus/>
    <ci> a </ci>
    <apply> <plus/> <ci> b </ci> <ci> c </ci> </apply>
    <ci> d </ci>
  </apply>
</math>
```

$$a - (b - c) - d$$

```
<math>
  <apply> <minus/>
    <ci> a </ci>
    <apply> <minus/> <ci> b </ci> <ci> c </ci> </apply>
    <ci> d </ci>
  </apply>
</math>
```

$$a + b - c + d$$

```
<math>
  <apply> <plus/>
    <ci> a </ci>
    <apply> <minus/> <ci> b </ci> <ci> c </ci> </apply>
    <ci> d </ci>
  </apply>
</math>
```

$$a - (b + c) - d$$

```
<math>
  <apply> <minus/>
    <ci> a </ci>
    <apply> <plus/> <ci> b </ci> <ci> c </ci> </apply>
    <ci> d </ci>
  </apply>
</math>
```

Another place where parentheses are not needed is the following:

```
<math>
  <apply> <minus/>
    <apply> <exp/>
      <cn> 3 </cn>
    </apply>
  </apply>
</math>
```

This means that the interpreter of this kind of MathML has to analyze child elements in order to choose the right way to typeset the formula. The output will look like:

$$-\mathrm{e}^3$$

By default, as less braces as possible are used. As demonstrated, a special case is when *plus* and *minus* have one sub element to deal with. If you really want many braces there, you can turn off sign reduction.

sign	reduction	yes	use as less braces as possible
		no	always use braces

We will demonstrate these alternatives with an example.

$$a + \sin b + c^5 + \sin^2 d + e$$

We need quite some code to encode this formula.

```
<math>
  <apply> <plus/>
    <ci> a </ci>
    <apply> <sin/>
      <ci> b </ci>
    </apply>
    <apply> <power/>
      <ci> c </ci>
      <cn> 5 </cn>
    </apply>
    <apply> <power/>
      <apply> <sin/>
        <ci> d </ci>
      </apply>
      <cn> 2 </cn>
    </apply>
    <ci> e </ci>
  </apply>
</math>
```

With power reduction turned off, we get:

$$a + \sin b + c^5 + (\sin d)^2 + e$$

As directive we used:

```
<?context-mathml-directive power reduction no ?>
```

The following example illustrates that we should be careful in coding such formulas; here the *power* is applied to the argument of *sin*.

$$a + \sin b + c^5 + \sin\left(d^2\right) + e$$

```
<math>
  <apply> <plus/>
    <ci> a </ci>
    <apply> <sin/>
      <ci> b </ci>
    </apply>
    <apply> <power/>
      <ci> c </ci>
      <cn> 5 </cn>
    </apply>
    <apply> <sin/>
      <apply> <power/>
        <ci> d </ci>
        <cn> 2 </cn>
      </apply>
    </apply>
    <ci> e </ci>
  </apply>
</math>
```

divide
Divisions can be very space consuming but there is a way out: using a forward slash symbol. You can set the level at which this will take place. By default, fractions are typeset in the traditional way.

$$\frac{1}{1+\frac{1}{x}}$$

```
<math>
  <apply> <divide/>
    <cn> 1 </cn>
    <apply> <plus/>
      <cn> 1 </cn>
      <apply> <divide/>
        <cn> 1 </cn>
        <ci> x </ci>
      </apply>
    </apply>
  </apply>
</math>
```

$$\frac{1}{1+\frac{1}{1+\frac{1}{x}}}$$

```
<math>
  <apply> <divide/>
```

```
    <cn> 1 </cn>
    <apply> <plus/>
      <cn> 1 </cn>
      <apply> <divide/>
        <cn> 1 </cn>
        <apply> <plus/>
          <cn> 1 </cn>
          <apply> <divide/>
            <cn> 1 </cn>
            <ci> x </ci>
          </apply>
        </apply>
      </apply>
    </apply>
  </apply>
</math>
```

$$\frac{1}{1+1/x}$$

$$\frac{1}{1+1/\left(1+1/x\right)}$$

```
<?context-mathml-directive divide level 1?>
```

$$\frac{1}{1+\frac{1}{x}}$$

$$\frac{1}{1+\frac{1}{1+1/x}}$$

```
<?context-mathml-directive divide level 2?>
```

relation

You should keep in mind that (at least level 2) content MathML is not that rich in terms of presenting your ideas in a visually attractive way. On the other hand, because the content is highly structured, some intelligence can be applied when typesetting them. By default, a relation is not vertically aligned but typeset horizontally.

If an application just needs raw formulas, definitions like the following are all right.

```
<math>
  <apply> <eq/>
    <apply> <plus/>
      <ci> a </ci>
      <ci> b </ci>
      <ci> c </ci>
    </apply>
    <apply> <plus/>
      <ci> d </ci>
      <ci> e </ci>
    </apply>
    <apply> <plus/>
      <ci> f </ci>
      <ci> g </ci>
      <ci> h </ci>
      <ci> i </ci>
    </apply>
    <cn> 123 </cn>
  </apply>
</math>
```

The typeset result will bring no surprises:

$$a + b + c = d + e = f + g + h + i = 123$$

But, do we want to show a formula that way? And what happens with much longer formulas? You can influence the appearance with processing instructions.

`relation`	`align`	`no`	don't align relations
		`left`	align all relations left
		`right`	align all relations right
		`first`	place the leftmost relation left
		`last`	place the rightmost relation right

The next couple of formulas demonstrate in what way the previously defined formula is influenced by the processing instructions.

$$\begin{aligned} a + b + c &= \\ d + e &= \\ f + g + h + i &= \\ 123 & \end{aligned}$$

```
<?context-mathml-directive relation align left ?>
```

$$\begin{aligned} & a + b + c \\ = {} & d + e \\ = {} & f + g + h + i \\ = {} & 123 \end{aligned}$$

```
<?context-mathml-directive relation align right ?>
```

$$\begin{aligned} a + b + c &= d + e \\ &= f + g + h + i \\ &= 123 \end{aligned}$$

```
<?context-mathml-directive relation align first ?>
```

$$\begin{aligned} a + b + c &= \\ d + e &= \\ f + g + h + i &= 123 \end{aligned}$$

```
<?context-mathml-directive relation align last ?>
```

base

When in a document several number systems are used, it can make sense to mention the base of the number. There are several ways to identify the base.

`base`	`symbol`	`numbers`	a (decimal) number
		`characters`	one character
		`text`	a mnemonic
		`no`	no symbol

By default, when specified, a base is identified as number.

```
<math>
  <cn type="integer" base="8"> 1427 </cn>
</math>
```

$$1427_8$$

```
<?context-mathml-directive base symbol numbers ?>
```

$$1427_{\mathrm{O}}$$

```
<?context-mathml-directive base symbol characters ?>
```

$$1427_{\mathrm{OCT}}$$

```
<?context-mathml-directive base symbol text ?>
```

function

There is a whole bunch of functions available as empty element, like *sin* and *log*. When a function is applied to a function, braces make not much sense and placement is therefore disabled.

`function`	`reduction`	`yes`	chain functions without braces
		`no`	put braces around nested functions

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

$$\sin x$$

```
<?context-mathml-directive function reduction yes?>
```

$$\sin(x)$$

```
<?context-mathml-directive function reduction no?>
```

limits

When limits are placed on top of the limitation symbol, this generally looks better than when they are placed alongside. You can also influence limit placement per element. This feature is available for *int*, *sum*, *product* and *limit*.

`limit`	`location`	`top`	place limits on top of the symbols
		`right`	attached limits as super/subscripts

```
<math>
  <apply> <int/>
    <bvar> <ci> x </ci> </bvar>
    <lowlimit> <cn> 0 </cn> </lowlimit>
    <uplimit> <cn> 1 </cn> </uplimit>
  </apply>
</math>
```

$$\int\limits_0^1 \mathrm{d}x$$

```
<?context-mathml-directive int location top?>
```

$$\int_0^1 \mathrm{d}x$$

```
<?context-mathml-directive int location right?>
```

declare

Currently declarations are not supposed to end up in print. By default we typeset a message, but you can as well completely hide declarations.

declare	state	start	show declarations
		stop	ignore (hide) declarations

lambda

There is more than one way to visualize a lambda function. As with some other settings, changing the appearance can best take place at the document level.

lambda	alternative	b	show lambda as arrow
		a	show lambda as set

```
<math>
  <lambda>
    <bvar> <ci> x </ci> </bvar>
    <apply> <log/>
      <ci> x </ci>
    </apply>
  </lambda>
</math>
```

$$\lambda(x, \log x)$$

```
<?context-mathml-directive lambda alternative a?>
```

$$x \mapsto \log x$$

```
<?context-mathml-directive lambda alternative b?>
```

power

Taking the power of a function looks clumsy when braces are put around the function. Therefore, by default, the power is applied to the function symbol instead of the whole function.

power	reduction	yes	attach symbol to function symbol
		no	attach symbol to function argument

```
<math>
  <apply> <power/>
    <apply> <ln/>
      <ci> x </ci>
    </apply>
    <cn> 3 </cn>
  </apply>
</math>
```

$$\ln^3 x$$

```
<?context-mathml-directive power reduction yes?>
```

$$(\ln x)^3$$

```
<?context-mathml-directive power reduction no?>
```

diff

Covering all kind of differential formulas is not trivial. Currently we support two locations for the operand (function). By default the operand is placed above the division line.

`diff`	`location`	`top`	put the operand in the fraction
		`right`	put the operand after the fraction

```
<math>
  <apply> <diff/>
    <bvar>
      <ci> x </ci>
      <degree> <cn> 2 </cn> </degree>
    </bvar>
    <apply> <fn> <ci> f </ci> </fn>
      <apply> <plus/>
        <apply> <times/>
          <cn> 2 </cn>
          <ci> x </ci>
        </apply>
        <cn> 1 </cn>
      </apply>
    </apply>
  </apply>
</math>
```

$$\frac{\mathrm{d}^2 f(2x+1)}{\mathrm{d}x^2}$$

```
<?context-mathml-directive diff location top?>
```

$$\frac{\mathrm{d}^2}{\mathrm{d}x^2}\,(f(2x+1))$$

```
<?context-mathml-directive diff location right?>
```

vector

Depending on the complication of a vector or on the available space, you may wish to typeset a vector horizontally or vertically. By default a vector is typeset horizontally.

`vector`	`direction`	`horizontal`	put vector elements alongside
		`vertical`	stack vector elements

```
<math>
  <apply> <eq/>
    <vector>
      <ci> x </ci>
      <ci> y </ci>
      <ci> z </ci>
    </vector>
    <vector>
      <cn> 1 </cn>
      <cn> 0 </cn>
      <cn> 1 </cn>
    </vector>
  </apply>
</math>
```

$$(x,y,z) = (1,0,1)$$

```
<?context-mathml-directive vector direction horizontal?>
```

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

```
<?context-mathml-directive vector direction vertical?>
```

times
Depending on the audience, a multiplication sign is implicit (absent) or represented by a regular times symbol or a dot.

`times`	`symbol`	`no`	don’t add a symbol
		`yes`	separate operands by a times ($\times$)
		`dot`	separate operands by a dot ($\cdot$)

```
<math>
  <apply> <plus/>
    <ci> x </ci>
    <apply> <times/>
      <cn> 2 </cn>
      <ci> x </ci>
    </apply>
  </apply>
</math>
```

$$x + 2x$$

```
<?context-mathml-directive times symbol no?>
```

$$x + 2 \times x$$

```
<?context-mathml-directive times symbol yes?>
```

$$x + 2 \cdot x$$

```
<?context-mathml-directive times symbol dot?>
```

log
The location of a logbase depends on tradition and/or preference, which is why we offer a few alternatives: as pre superscript (in the right top corner before the symbol) or as post subscript (in the lower left corner after the symbol).

`log`	`location`	`right`	place logbase at the right top
		`left`	place logbase at the lower left

```
<math>
  <apply> <log/>
    <logbase>
      <ci> 3 </ci>
    </logbase>
    <apply> <plus/>
      <ci> x </ci>
      <cn> 1 </cn>
    </apply>
  </apply>
</math>
```

$$\log_3 (x + 1)$$

```
<?context-mathml-directive log location right?>
```

$$^{3}\log(x+1)$$

```
<?context-mathml-directive log location left?>
```

polar

The location of a logbase depends on tradition and/or preference, which is why we offer a few alternatives: as pre superscript (in the right top corner before the symbol) or as post subscript (in the lower left corner after the symbol).

`polar`	`alternative`	`a`	explicit polar notation
		`b`	exponential power notation
		`c`	exponential function notation

```
<math>
  <cn type="polar"> 2 <sep/> <pi/> </cn>
</math>
```

$$\mathrm{Polar}\,(2,\pi)$$

```
<?context-mathml-directive polar alternative a?>
```

$$2\mathrm{e}^{\pi\,\mathrm{i}}$$

```
<?context-mathml-directive polar alternative b?>
```

$$2\exp(\pi\,\mathrm{i})$$

```
<?context-mathml-directive polar alternative c?>
```

e-notation

Depending on the context, you may want to typeset the number `1.23e4` not as this sequence, but using a multiplier construct. As with the *times*, we support both multiplication symbols.

`enotation`	`symbol`	`no`	no interpretation
		`yes`	split exponent, using ×
		`dot`	split exponent, using ·

```
<math>
  <cn type="e-notation">10<sep/>23</cn>
</math>
```

$$10\mathrm{e}23$$

```
<?context-mathml-directive enotation symbol no?>
```

$$10 \times 10^{23}$$

```
<?context-mathml-directive enotation symbol yes?>
```

$$10 \cdot 10^{23}$$

```
<?context-mathml-directive enotation symbol dot?>
```

Typesetting modes

Math can be typeset in line or in display. In order not to widen up the text of a paragraph too much, inline math is typeset more cramped. Since MATHML does provide just a general purpose *math* element we have to provide the information needed using other elements. Consider the following text.

To what extent is math supposed to reflect the truth and nothing but the truth? Consider the simple expression $10 = 3 + 7$. Many readers will consider this the truth, but then, can we assume that the decimal notation is used?

$$10 = 3 + x$$

In many elementary math books, you can find expressions like the previous. Because in our daily life we use the decimal numbering system, we can safely assume that $x = 7$. But, without explicitly mentioning this boundary condition, more solutions are correct.

$$10 = 3 + 5 \tag{1a}$$

In formula 1a we see an at first sight wrong formula. But, if we tell you that octal numbers are used, your opinion may change instantly. A rather clean way out of this confusion is to extend the notation of numbers by explicitly mentioning the base.

$$10_8 = 3_8 + 5_8 \tag{1b}$$

Of course, when a whole document is in octal notation, a proper introduction is better than annotated numbers as used in formula 1b.

In terms of XML this can look like:

```
To what extent is math supposed to reflect the truth and nothing but
the truth? Consider the simple expression <math> <apply> <eq/> <cn>
10 </cn> <apply> <plus/> <cn> 3 </cn> <cn> 7 </cn> </apply> </apply>
</math>. Many readers will consider this the truth, but then, can we
assume that the decimal notation is used?

<formula>
  <math>
    <apply> <eq/>
      <cn> 10 </cn>
      <apply> <plus/>
        <cn> 3 </cn>
        <ci> x </ci>
      </apply>
    </apply>
  </math>
</formula>

In many elementary math books, you can find expressions like the
previous. Because in our daily life we use the decimal numbering system,
we can safely assume that <math> <apply> <eq/> <ci> x </ci> <cn> 7 </cn>
</apply> </math>. But, without explicitly mentioning this boundary
condition, more solutions are correct.

<formula label="octal" sublabel="a">
  <math>
    <apply> <eq/>
      <cn> 10 </cn>
      <apply> <plus/>
        <cn> 3 </cn>
        <cn> 5 </cn>
      </apply>
    </apply>
  </math>
</formula>

In <textref label="octal">formula</textref> we see an at first sight
wrong formula. But, if we tell you that octal numbers are used, your
opinion may change instantly. A rather clean way out of this confusion
is to extend the notation of numbers by explicitly mentioning the base.

<subformula label="octal base" sublabel="b">
  <math>
    <apply> <eq/>
      <cn type="integer" base="8"> 10 </cn>
      <apply> <plus/>
        <cn type="integer" base="8"> 3 </cn>
        <cn type="integer" base="8"> 5 </cn>
      </apply>
    </apply>
  </math>
</subformula>

Of course, when a whole document is in octal notation, a proper
introduction is better than annotated numbers as used in <textref
label="octal base">formula</textref>.
```

Math that is part of the text flow is automatically handled as in line math. If needed you can encapsulate the code in an *imath* environment. Display math is recognized as such when it is a separate paragraph, but since this is more a TEX feature than an XML one, you should encapsulate display math either in a *dmath* element or in a *formula* or *subformula* element.

Getting started

A comfortable way to get accustomed to MATHML is to make small documents of the following form:

```
\usemodule[mathml]

\starttext

\startXMLdata
<math>
  <apply> <cos/>
    <ci> x </ci>
  </apply>
</math>
\stopXMLdata

\stoptext
```

As you see, we can mix MATHML with normal TEX code. A document like this is processed in the normal way using TEXEXEC. If you also want to see the original code, you can say:

```
\usemodule[mathml]
```

```
\starttext

\startbuffer
<math>
  <apply> <cos/>
    <ci> x </ci>
  </apply>
</math>
\stopbuffer

\processXMLbuffer

\typebuffer

\stoptext
```

Like TEX and METAPOST code, buffers can contain MATHML code. The advantage of this method is that we only have to key in the data once. It also permits you to experiment with processing instructions.

```
\startbuffer[mml]
<math>
  <apply> <log/>
    <logbase> <cn> 3.5 </cn> </logbase>
    <ci> x </ci>
  </apply>
</math>
\stopbuffer

\startbuffer[pi]
 <?context-mathml-directive log location right?>
\stopbuffer

\processXMLbuffer[pi,mml]

\startbuffer[pi]
 <?context-mathml-directive log location left?>
\stopbuffer

\processXMLbuffer[pi,mml]
```

If you like coding your documents in TEX but want to experiment with MATHML, combining both languages in the way demonstrated here may be an option. When you provide enough structure in your TEX code, converting a document to XML is then not that hard to do. Where coding directly in XML is kind of annoying, coding MATHML is less cumbersome, because you can structure your formulas pretty well, especially since the fragments are small so that proper indentation is possible.

Further reading

The MathML spec

You can fetch the latest version of this document, which is written by the MATHML committee, can be fetched from the www.w3c.org web–site. Depending on the state of development, you can grab the draft, recommendation or standard.

The TEXbook
Once you have read this book, you will see why MATHML is not embraced by those who love to optimize the look and feel of their math formulas. Although your documents will be more consistent when you code in (content) MATHML, you also lose many fine points of math typesetting. Of course you can always fall back on the *annotation* element. This book is written by Donald Knuth and published by Addison Wesley.

The XML Companion
Written by Neil Bradley and published by Addison Wesley, this book is a good introduction to XML and its relatives. More compact but not less useful, is Robert Eckstein's *XML Pocket Reference*, published by O'Reilly.

XML in ConTEXt
This document describes how you can use CONTEXT for processing your XML documents. You may also want to take a look at the beginners manual, the reference manual, guides and examples that can be fetched from www.pragma-ade.com.

MathML in ConTEXt
We have keyed in a lot of realistic MATHML examples and turned them into a document suitable for viewing on your computer display. Over time, this collection will grow.