Conversion of MathML to SVG via XSLT: pMML2SVG

User Documentation

Jérôme Joslet, Université de Liège `<jerome.joslet@student.ulg.ac.be>`
Justus H Piater
Professor
Université de Liège Faculty of Applied Sciences Department of Electrical Engineering and Computer Science

Conversion of MathML to SVG via XSLT: pMML2SVG: User Documentation

by Jérôme Joslet
Justus H Piater
Professor
Université de Liège Faculty of Applied Sciences Department of Electrical Engineering and Computer Science

Table of Contents

List of Tables

List of Examples

List of Equations

Chapter 1. pMML2SVG: User guide

XSLT Processors

pMML2SVG is an XSLT stylesheet and has to be used with an XSLT processor to execute the transformation. pMML2SVG takes advantage of some features newly introduced in XSLT 2 [http://www.w3.org/TR/xslt20/] such as temporary trees, functions, tunnel parameters, and strong typing. One XSLT processor that implements the full XSLT 2 recommendation is Saxon 9 [http://www.saxonica.com/].

However, all ancillary stylesheets that come with pMML2SVG are written in XSLT 1 and can be processed by any of the widespread XSLT 1 processors (xsltproc [http://xmlsoft.org/XSLT/], Saxon 6 [http://saxon.sourceforge.net/saxon6.5.5/], Xalan [http://xalan.apache.org/], …).

Basic Usage

The easiest way to use the stylesheet is to simply process a MathML file and transform it into a SVG file. Suppose that we have a MathML file, called `test.xml`, containing the following MathML code :

Example 1.1. test.xml example: source code

```
<?xml version="1.0"?>
<math xmlns="http://www.w3.org/1998/Math/MathML">
  <mrow>
    <mi>x</mi>
    <mo>+</mo>
    <msqrt>
      <mn>245</mn>
      <mo>+</mo>
      <mi>y</mi>
    </msqrt>
  </mrow>
</math>
```

We can simply transform this file into SVG using the Saxon 9 processor with the following command:

```
java -jar SAXON/saxon9.jar -s:test.xml
     -xsl:pMML2SVG/XSLT2/pmml2svg.xsl -o:test.svg
```

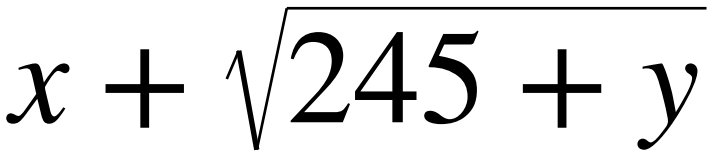
This command produces a file `test.svg` containing the following code:

Example 1.2. test.xml example: SVG transformation source code

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE svg
  PUBLIC "-//W3C//DTD SVG 1.1//EN" "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.d
<svg xmlns:pmml2svg="https://sourceforge.net/projects/pmml2svg/"
     xmlns="http://www.w3.org/2000/svg"
     version="1.1"
     width="280.0444px"
     height="73.03333333333333px"
     viewBox="0 0 280.0444 73.03333333333333">
   <metadata>
      <pmml2svg:baseline-shift>20.299999999999997</pmml2svg:baseline-shift>
   </metadata>
   <g stroke="none" fill="#000000" text-rendering="optimizeLegibility"
      font-family="STIXGeneral,STIXSize1">
      <g xmlns:doc="http://nwalsh.com/xsl/documentation/1.0"
         style="font-family: STIXGeneral,STIXSize1; fill: ; background-color: tran
         <g style="font-family: STIXGeneral,STIXSize1; fill: ; background-color: t
            <text style="font-family: STIXGeneral, STIXSize1; font-style: italic;
                  x="11.35"
                  y="52.733333333333334"
                  font-size="50">x</text>
            <text x="44.8111" y="52.733333333333334" font-size="50"
                  style="font-family: STIXGeneral, STIXSize1; fill: black; backgro
            <text style="font-family: STIXGeneral, STIXSize1; fill: black; backgro
                  x="115.1722"
                  y="52.733333333333334"
                  font-size="50">245</text>
            <text x="201.2833" y="52.733333333333334" font-size="50"
                  style="font-family: STIXGeneral, STIXSize1; fill: black; backgro
            <text style="font-family: STIXGeneral, STIXSize1; font-style: italic;
                  x="247.8444"
                  y="52.733333333333334"
                  font-size="50">y</text>
            <line x1="92.6722" y1="36.516666666666666" x2="97.1722" y2="25.91" fil
                  stroke="black"
                  stroke-width="1"/>
            <line x1="97.1722" y1="25.91" x2="103.9222" y2="63.03333333333333" fil
                  stroke="black"
                  stroke-width="2"/>
            <line x1="103.9222" y1="63.03333333333333" x2="115.1722" y2="10.000000
                  fill="none"
                  stroke="black"
                  stroke-width="1"/>
            <line x1="115.1722" y1="10.000000000000002" x2="272.5444" y2="10.00000
                  fill="none"
                  stroke="black"
                  stroke-width="1"/>
         </g>
      </g>
   </g>
</svg>
```

This code renders like that:

Example 1.3. test.xml example: SVG transformation result



The transformation result can be modified by tuning some parameters. The parameters that can be changed and their meanings are listed below.

Stylesheet parameters

`svgMasterUnit` Defines the unit that will be used in the SVG (default is `px`).

`initSize` Defines the initial font size (default is 50).

`minSize` Defines the minimal font size in the SVG rendering (default is 8). This parameter is used to render fraction, superscript, subscript, etc. which have to be smaller than the normal text.

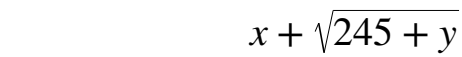
`fontName` Defines fonts that will be used to render the SVG file. To learn how to change default fonts, have a look at the section called “Changing the Font” . By default, the value of this parameter is `STIXGeneral,STIXSize1`. These fonts can be downloaded from the Internet [http://www.stixfonts.org/]. The objective of the STIX project is to provide a font that serves the scientific and engineering community.

For example, to change the initial size of the font, we can change the parameter `initSize` by adding information in the command line:

```
java -jar saxon9.jar -s:test.xml
     -xsl:pMML2SVG/XSLT2/pmml2svg.xsl -o:test.svg initSize=15
```

You will obtain the following equation:

Example 1.4. test.xml example: SVG transformation result (modified initSize)



pMML2SVG comes with an example script called `pmml2svg`. This script can be used for batch conversion and validation of a set of files using a command such as **./pmml2svg examples/*.xml**. You may want to adapt it to your needs.

In the examples directory you will also find a script that can create an XHTML file containing both `MathML` code and SVG pictures. You can open this script using Firefox and compare Gecko MathML renderer with pMML2SVG. To use this script, simply go in `examples/` directory and execute **genhtml**. It will create the file `examples.xhtml` containing all the examples in MathML and in SVG.

Toolchain

In this section, we will see how to transform a DocBook file into a PDF or an (X)HTML file by adding an intermediate pMML2SVG step to transform MathML into SVG.

The DocBook example source code that I will use can be seen in Appendix A, *DocBook test file source code* . Suppose in this section that we have a file, `test.xml`, containing this code.

Do not forget to adapt files path to your system installation when testing the commands described in this section.

DocBook to PDF renderer

To transform a DocBook file into a PDF, including pMML2SVG step, we have to do three transformations. First, the DocBook file will be transformed into an XSL-FO file. This transformation is done by using the DocBook-XSL XSLT stylesheets [1] and an XSLT 1 processor. Let see some command examples using different XSLT processors.

`xsltproc` can be used to transform `test.xml` with this command (if `xsltproc` is already installed):

```
xsltproc -o test.fo docbook-xsl-1.74.3/fo/docbook.xsl test.xml
```

Saxon 6.5.5 can also be used to do this first transformation, the following command must be used:

```
java -jar saxon.jar -o test.fo test.xml
     docbook-xsl-1.74.3/fo/docbook.xsl
```

FOP is also able to do this transformation with the following command:

```
fop -xml test.xml -xsl docbook-xsl-1.74.3/fo/docbook.xsl
    -foout test.fo
```

All these commands will create a file called `test.fo` containing the XSL-FO code that will be used to render a PDF file. If you transform directly a PDF file with this code, the MathML source code will be displayed in red on the PDF page.

The second transformation of our toolchain is MathML to SVG transformation using pMML2SVG. This time, we need an XSLT 2 processor and we will use Saxon 9. We will also use the pMML2SVG stylesheet named `fopmml2svg.xsl` from the `tools` directory of pMML2SVG distribution. This stylesheet is used to transform all MathML that it will find inside the XSL-FO code. The following command line is used:

```
java -jar saxon9.jar -xsl:pMML2SVG/tools/fopmml2svg.xsl
     -o:mathml_test.fo test.fo
```

This command creates a file called `mathml_test.fo` containing XSL-FO code with MathML transformed into SVG. This file will be used to render the final PDF output file.

The last step of our toolchain is to transform the `mathml_test.fo` file into a PDF file. This transformation is done by using FOP with the following command:

```
fop mathml_test.fo test.pdf
```

You can now open `test.pdf` with your favorite PDF viewer and see the result.

It is a real waste of time to always type all these commands each time you want to compile your DocBook document into PDF. Therefore, it is useful to write a small script that will do all these commands for you. For example, create a file, called `pmml2svgpdf` with the following source code:

[1] These stylesheets can be found on this website: `http://docbook.sourceforge.net/`

```
#!/bin/sh
fop="fop" # Path to FOP

mathxsl="pMML2SVG/tools/fopmml2svg.xsl" # Path to pmml2svg XSLT
                                        # stylesheet to treat fo
mathtransform="java -jar saxon9.jar -xsl:$mathxsl -o:" # Saxon 9

docbookxsl="docbook-xsl-1.74.3/fo/docbook.xsl" # Path to Norman
                                                # Walsh XSLT
docbooktransform="$fop -xsl $docbookxsl -xml" # FOP command

for xmlfile in $*; do
  file=${xmlfile%.*}

  # First step: DocBook to XSL-FO transformation by using N. Walsh
  # stylesheet.
  fofile=$file.fo
  echo "DocBook to XSL-FO: " $file
  $docbooktransform $xmlfile "-foout" $fofile

  # Second step: MathML to SVG transformation by using pMML2SVG.
  mmlfile=mathml_$file.fo
  echo "MathML Transformation: " $file
  $mathtransform$mmlfile $fofile

  # Third step: XSL-FO to PDF computation by using FOP.
  pdffile=$file.pdf
  echo "XSL-FO to PDF: " $file
  $fop $mmlfile $pdffile

  # This last command will clean all temporary files created
  # in the toolchain transformation.
  echo "Cleaning temporary file"
  rm -rf $fofile $mmlfile # Remove temporary file
done
```

Now, you can execute all the toolchain transformations by using this unique command:

```
./pmml2svgpdf test.xml
```

You can also use it to treat a group of DocBook file. For example, the command **./pmml2svgpdf *.xml** will transform all the XML DocBook files of the current folder into PDF files.

This document was compiled to PDF by using the same script and this toolchain transformation.

DocBook to (X)HTML renderer

To transform a DocBook file into (X)HTML, only one transformation is done. After that transformation, pMML2SVG is used to transform MathML into SVG. The main transformation uses an XSLT 1 stylesheet from Norman Walsh. Therefore, you can use both `xsltproc` or Saxon. Here are the command lines for each of these processors:

```
xsltproc -o test.xhtml docbook-xsl-1.74.3/xhtml-1_1/docbook.xsl
      test.xml
```

```
java -jar saxon.jar -o test.xhtml test.xml
     docbook-xsl-1.74.3/xhtml-1_1/docbook.xsl
```

If your browser supports MathML rendering (Firefox or Opera for example), you can open `test.xhtml` with it to see the result. To transform the MathML code into SVG, you have to use `htmlpmml2svg.xsl` stylesheet (coming with pMML2SVG distribution in the `tools/` folder) with an XSLT 2 processor. Here is a command that executes this transformation:

```
java -jar saxon9.jar -xsl:pMML2SVG/tools/htmlpmml2svg.xsl
     -o:testSVG.xhtml test.xhtml
```

If your browser supports SVG (the majority of modern browsers do), you can open `testSVG.xhtml` with it to see the result.

Likewise the PDF transformation, you can also use a script to make the transformation. Here is a sample script code that transforms a DocBook file into an (X)HTML file:

```
#!/bin/sh
mathxsl="pMML2SVG/tools/htmlpmml2svg.xsl" # pmml2svg html stylesheet
mathtransform="java -jar saxon9.jar -xsl:$mathxsl -o:" # Saxon 9

docbookxsl="docbook-xsl-1.74.3/xhtml-1_1/docbook.xsl" # DocBook to
                                                       # XHTML
                                                       # stylesheet
docbooktransform="xsltproc --output " # XSLT 1 processor
                                      # (xsltproc here)

for xmlfile in $*; do
  file=${xmlfile%.*}

  # First step: DocBook to XHTML transformation by using N. Walsh
  # stylesheet.
  tempfile=temp_$file.xhtml
  echo "DocBook to HTML: " $file
  $docbooktransform $tempfile $docbookxsl $xmlfile

  # Second step: MathML to SVG transformation by using pMML2SVG.
  htmlfile=$file.xhtml
  echo "MathML Transformation: " $file
  $mathtransform$htmlfile $tempfile

  # This last command will clean all temporary files created
  # in the toolchain transformation.
  echo "Cleaning temporary file"
  rm -rf $tempfile
done
```

This script can also be used to transform a group of DocBook files into (X)HTML files the same way as PDF script does.

Changing the Font

In this section we will see how to change the default font of pMML2SVG and how to create the metrics file that are necessary to do the transformation.

FOP font metrics

To add a new font to pMML2SVG, you need to create font metrics files and add them in the `XSLT2/` directory of pMML2SVG distribution. These font metrics files are created with `ttfreader` tool that comes with FOP. To fully support a font, you need to create four XML font metrics files with FOP `ttf-reader`. The first file, named `MyFont.xml`, contains the normal style font metrics, from `MyFont.ttf` for example. The second font metrics file, named `MyFont-Italic.xml`, contains all metrics information about the italic font, from `MyFontItalic.ttf` for example. The third file, called `MyFont-Bold.xml`, contains all information about the bold font, from `MyFontBol.ttf`. And the last file, named `MyFont-Bold-Italic.xml` contains all metrics information about the bold italic font, from `MyFontBolIta.ttf` for example.

Since the initial FOP `ttfreader` did not provide enough precision to draw the character, some modifications have been done to the source code to add more precision to it. The new source code can be found in the `ttfreader/` folder coming with the pMML2SVG distribution. To create font metrics files, this tool has to be compiled. It requires a complete FOP distribution to work correctly.

To compile the code, we will use the `make` script in the `ttfreader/` folder. First, you have to modify the first two lines of this script to adapt the script to your local installation. The first line is the path to a FOP distribution and the second line is the Java `classpath` used in the compilation. This classpath must includes the following libraries: `FOP.jar`, `commons-logging`, `commons-io` and `xmlgraphics-commons`.

After making this configuration, you can compile the tool using **./make** command. This script will produce the `TTFReader.jar` file. This final JAR file will be used to compile font metrics files.

To create a font metrics file, configuration has to be done in the `TTFReader` script that comes with pMML2SVG distribution in the `ttfreader/` directory. As the `make` file, the first two lines have to be changed in the same way. The `classpath` must contain: `TTFReader.jar`, `FOP.jar`, `commons-logging`, `commons-io` and `xmlgraphics-commons`. The `TTFReader.jar` archive must be written before `FOP.jar` in the classpath.

We can now call `TTFReader` script to create a metrics file like that:

```
./TTFReader MyFont.ttf
```

This command will produce a `MyFont.xml` file that contains all the metrics that are needed by pMML2SVG. This script can also create a group of TTF files in one shot. For example, if you want to compile your entire font folder, you can run the following command:

```
./TTFReader $HOME/.fonts/*.ttf
```

All the newly created XML metrics files will be created in `$HOME/.fonts/` folder. You can then copy all these metrics in the `XSLT2/` directory of pMML2SVG by using, for example, the following **cp** command:

```
cp $HOME/.fonts/*.xml pMML2SVG/XSLT2/
```

Once these XML metrics files have been copied into `XSLT2/` pMML2SVG folder, it will be able to use these new fonts in the transformation.

Font parameter

To tell pMML2SVG that it will use the new fonts, we can simply change the `fontName` parameter when calling the transformation. For example, the command will be (using the previous `test.xml` MathML example):

```
java -jar saxon9.jar -s:test.xml
     -xsl:pMML2SVG/XSLT2/pmml2svg.xsl -o:test.svg fontName=MyFont
```

You can also keep the default font and add your font at the beginning of the fonts list, for our example, the parameter `fontName` will be `MyFont,STIXGeneral,STIXSize1`. This last modification can be done directly in the stylesheet or using the parameter.

Note that the font has to implement all the mathematical Unicode glyphs to have a correct rendering.

Usage Examples

In this section you will see complex examples exclusively rendered by pMML2SVG. All these examples have been directly included, in MathML, in the DocBook code of this document. For more examples, see the `examples/` directory of pMML2SVG.

Equation 1.1. Alternatives

$$\begin{aligned} p_{ij} &= \begin{cases} \mu & \text{if } j \text{ can produce } i \\ 0 & \text{otherwise} \end{cases} \\ p_{iX} &= \lambda < \mu \end{aligned}$$

Equation 1.2. Elaborate likelihood

$$\begin{aligned} & P(\text{patches}|\text{pattern } j) \\ = \; & P\left(\forall\, i : \begin{matrix} n_i & \text{patches of type } i \text{ from pattern,} \\ n_{Ii} - n_i & \text{patches of type } i \text{ from noise} \end{matrix} \middle| \text{pattern } j\right) \\ = \; & \prod_i P(\text{patch of type } i|\text{pattern } j)^{n_i} \\ & \prod_i P(\text{patch of type } i|\text{noise})^{n_{Ii}-n_i} \\ = \; & \prod_i \left(p_{ij}^{\,n_i}\, p_{iX}^{\,n_{Ii}-n_i}\right) \end{aligned}$$

Equation 1.3. Complex square root

$$x = \sqrt{A + \frac{w + \frac{m^8}{\sqrt{2}}}{i^{\frac{1}{2}}} + \frac{\frac{a_i}{2}}{\frac{a_i}{5}}}$$

Equation 1.4. Newton

$$T_1 = -\frac{1}{k}\sqrt{\frac{s_1}{2}}\sqrt{s_1 s_0 - {s_0}^2} - \frac{1}{k}\left(\frac{s_1}{2}\right)^{\frac{3}{2}}\left[\pm\frac{\pi}{2} - \sin^{-1}\left(\frac{2s_0 - s_1}{s_1}\right)\right]$$

Equation 1.5. Rotation

$$\mathbf{x}' = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}\mathbf{x}$$

Equation 1.6. Swarzchild

$$\frac{dr}{dt} = -\sqrt{\frac{B^2(r)}{A(r)}\left[\frac{1}{B(r)} - E - \frac{J^2}{r^2}\right]}$$

Chapter 2. MathML Compliance

The following tables show the compliance of pMML2SVG with the MathML specification.

The first five tables address the implementation of elements and their attributes. A name in ***bold italic*** represents a MathML element. A value **Yes** in the implementation column means that the basic renderer for this element is implemented (correct default display). A name with no style is an attribute, a **Yes** in the implementation column means that this attribute is fully supported. **Partial** in the implementation always comes with a text that gives more detail about the current status of the implementation.

The last two tables show the attributes that are common to all MathML elements. These are mainly style attributes.

Table 2.1. Conformance: MathML elements: Tokens

Tag name	Implementation
mi	**Yes**
mn	**Yes**
mo	**Yes**
form	**Yes**
fence	**Yes**
separator	**Yes**
lspace	**Yes**
rspace	**Yes**
stretchy	**Partial**: More operators can be stretched but are not yet supported
Operator Dictionary	**Yes**
symmetric	**Yes**
maxsize	**Yes**
minsize	**Yes**
largeop	**Yes**
movablelimits	No
accent	**Yes**
Invisible operator	**Yes**
mtext	**Yes**
mspace	**Yes**
width	**Yes**
height	**Yes**
depth	**Yes**
linebreak	No
ms	**Yes**
lquote	**Yes**
rquote	**Yes**

Tag name	Implementation
mglyph	No
alt	No
fontfamily	No
index	No

Table 2.2. Conformance: MathML elements: General layout

Tag name	Implementation
mrow	**Yes**
mfrac	**Yes**
linethickness	**Yes**
numalign	**Yes**
denomalign	**Yes**
bevelled	No
msqrt	**Yes**
mroot	**Partial**: Size of base index > 1 not yet supported
mstyle	**Yes**
other elements attributes	No
scriptlevel	**Yes**
displaystyle	**Yes**
scriptsizemultiplier	**Yes**
scriptminsize	No
background	No
veryverythinmathspace	**Yes**
verythinmathspace	**Yes**
thinmathspace	**Yes**
mediummathspace	**Yes**
thickmathspace	**Yes**
verythickmathspace	**Yes**
veryverythickmathspace	**Yes**
merror	**Yes**
mpadded	No
width	No
lspace	No
height	No
depth	No
mphantom	**Yes**
mfenced	**Yes**
open	**Yes**

Tag name	Implementation
close	**Yes**
separators	**Yes**
menclose	**Yes**
notation	**Yes**

Table 2.3. Conformance: MathML elements: Scripts and limits

Tag name	Implementation
msub	**Yes**
subscriptshift	**Yes**
msup	**Yes**
superscriptshift	**Yes**
msubsup	**Yes**
subscriptshift	**Yes**
superscriptshift	**Yes**
munder	**Yes**
accentunder	**Yes**
mover	**Yes**
accent	**Yes**
munderover	**Yes**
accent	**Yes**
accentunder	**Yes**
mmultiscripts	No
subscriptshift	No
superscriptshift	No

Table 2.4. Conformance: MathML elements: Tables and matrices

Tag name	Implementation
mtable	**Yes**
align	No
rowalign	No
columnalign	**Yes**
groupalign	No
alignmentscope	No
columnwidth	No
width	No
rowspacing	No
columnspacing	No
rowlines	No

Tag name	Implementation
columnlines	No
frame	No
framespacing	No
equalrows	No
equalcolumns	No
displaystyle	No
side	No
minlabelspacing	No
mtr	**Yes**
rowalign	No
columnalign	**Yes**
groupalign	No
mlabeledtr	No
rowalign	No
columnalign	No
groupalign	No
mtd	**Yes**
rowspan	No
columnspan	No
rowalign	No
columnalign	**Yes**
groupalign	No

Table 2.5. Conformance: MathML elements: Enlivening expressions

Tag name	Implementation
maction	**Yes**
actiontype	No
selection	No

Table 2.6. Attributes common to all elements: tokens and general layout

Tag name	mathvariant	mathsize	mathbackground	mathcolor
Tokens				
mi	**Partial** (bold, italic, bold-italic). One letter default italic is implemented.	No	No	**Yes**
mn	**Partial** (bold, italic, bold-italic)	No	No	**Yes**
mo	**Partial** (bold, italic, bold-italic)	No	No	**Yes**

Tag name	mathvariant	mathsize	mathbackground	mathcolor
ms	**Partial** (bold, italic, bold-italic)	No	No	**Yes**
mtext	**Partial** (bold, italic, bold-italic)	No	No	**Yes**
mspace	**Partial** (bold, italic, bold-italic)	No	No	**Yes**
mglyph	No	No	No	No
General layout				
mrow	**Partial** (bold, italic, bold-italic)	No	No	**Yes**
math	**Partial** (bold, italic, bold-italic)	No	No	**Yes**
mfrac	No	No	No	No
msqrt	No	No	No	No
mroot	No	No	No	No
mstyle	**Partial** (bold, italic, bold-italic)	No	No	**Yes**
merror	**Partial** (bold, italic, bold-italic)	No	No	**Yes**
mpadded	No	No	No	No
mphantom	**Partial** (bold, italic, bold-italic)	No	No	**Yes**
mfenced	**Partial** (bold, italic, bold-italic)	No	No	**Yes**
menclose	**Partial** (bold, italic, bold-italic)	No	No	**Yes**

Table 2.7. Attributes common to all elements: scripts and limits, tables and matrices and enlivening expressions

Tag name	mathvariant	mathsize	mathbackground	mathcolor
Script and limit				
msub	No	No	No	No
msup	No	No	No	No
msubsup	No	No	No	No
munder	No	No	No	No
mover	No	No	No	No
munderover	No	No	No	No
mmultiscripts	No	No	No	No
Tables and matrices				
mtable	No	No	No	No
mlabeledtr	No	No	No	No

Tag name	mathvariant	mathsize	mathbackground	mathcolor
mtr	No	No	No	No
mtd	No	No	No	No
Enlivening expressions				
maction	No	No	No	No

Appendix A. DocBook test file source code

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE math PUBLIC "-//W3C//DTD MathML 2.0//EN"
                      "http://www.w3.org/TR/MathML2/dtd/mathml2.dtd">

<book xmlns="http://docbook.org/ns/docbook"
      version="5.0" xml:lang="en">
  <chapter>
    <title>Sample transformation</title>

    <para>
      Simple example
      <inlineequation>
 <math xmlns="http://www.w3.org/1998/Math/MathML">
   <mrow>
     <mi>p</mi>
     <mo>+</mo>
     <mn>547</mn>
   </mrow>
 </math>
      </inlineequation>
      in some text !
    </para>

    <para>
      Swarzild
      <inlineequation>
 <math xmlns="http://www.w3.org/1998/Math/MathML">
   <mrow>
     <mfrac>
       <mrow>
 <mi>d</mi>
 <mi>r</mi>
       </mrow>
       <mrow>
 <mi>d</mi>
 <mi>t</mi>
       </mrow>
     </mfrac>
     <mo>=</mo>
     <mrow>
       <mo>&minus;</mo>
       <msqrt>
 <mfrac>
   <mrow>
     <msup>
       <mi>B</mi>
       <mn>2</mn>
     </msup>
```

```
      <mfenced>
        <mi>r</mi>
      </mfenced>
    </mrow>
    <mrow>
      <mi>A</mi>
      <mfenced>
        <mi>r</mi>
      </mfenced>
    </mrow>
  </mfrac>
  <mo>&InvisibleTimes;</mo>
  <mfenced open="[" close="]">
    <mrow>
      <mfrac>
        <mn>1</mn>
        <mrow>
  <mi>B</mi>
  <mfenced>
    <mi>r</mi>
  </mfenced>
        </mrow>
      </mfrac>
      <mo>&minus;</mo>
      <mi>E</mi>
      <mo>&minus;</mo>
      <mfrac>
        <msup>
  <mi>J</mi>
  <mn>2</mn>
        </msup>
        <msup>
  <mi>r</mi>
  <mn>2</mn>
        </msup>
      </mfrac>
    </mrow>
  </mfenced>
       </msqrt>
     </mrow>
   </mrow>
</math>
      </inlineequation>
      in a row.
    </para>

    <para>
      And a matrix
      <inlineequation>
<math xmlns="http://www.w3.org/1998/Math/MathML">
  <mrow>
    <mfrac><mn>587</mn><mn>30</mn></mfrac>
    <mo>+</mo>
    <mo>(</mo>
```

```
      <mtable>
        <mtr>
  <mtd><mfrac><mn>587</mn><mn>30</mn></mfrac></mtd>
  <mtd><mn>34895</mn></mtd>
        </mtr>
        <mtr>
  <mtd><mfrac><mn>587</mn><mn>30</mn></mfrac></mtd>
  <mtd><mn>684</mn></mtd>
        </mtr>
      </mtable>
      <mo>)</mo>
    </mrow>
 </math>
       </inlineequation>
       , also in a row
     </para>
  </chapter>
</book>
```