An OASIS DITA Adoption Technical Committee Publication

DITA 1.3 Feature Article:
Making the Most of the New Math Specializations in DITA 1.3

Author: Autumn Cuellar, Design Science
On behalf of the DITA Adoption Technical Committee

Date: 31 March 2015

OASIS (Organization for the Advancement of Structured Information Standards) is a not-for-profit, international consortium that drives the development, convergence, and adoption of e-business standards. Members themselves set the OASIS technical agenda, using a lightweight, open process expressly designed to promote industry consensus and unite disparate efforts. The consortium produces open standards for Web services, security, e-business, and standardization efforts in the public sector and for application-specific markets. OASIS was founded in 1993. More information can be found on the OASIS website at http://www.oasis-open.org.

The OASIS DITA Adoption Technical Committee members collaborate to provide expertise and resources to educate the marketplace on the value of the DITA OASIS standard. By raising awareness of the benefits offered by DITA, the DITA Adoption Technical Committee expects the demand for, and availability of, DITA conforming products and services to increase, resulting in a greater choice of tools and platforms and an expanded DITA community of users, suppliers, and consultants.

DISCLAIMER: All examples presented in this article were produced using one or more tools chosen at the author's discretion and in no way reflect endorsement of the tools by the OASIS DITA Adoption Technical Committee.

This white paper was produced and approved by the OASIS DITA Adoption Technical Committee as a Committee Draft. It has not been reviewed and/or approved by the OASIS membership at-large.

Table of Contents

Introduction

Mathematics and DITA, because of its use in technical communications, have gone hand-in-hand for years. MathML is the W3C standard for encoding mathematical information in XML. DITA is largely implemented as an XML model. For this reason, the two standards seem a natural fit, and in part due to requirements for MathML support in DITA, the DITA 1.1 specification introduced the `<foreign>` element as a way in which to add MathML and other separately maintained vocabularies to DITA documents. Thus, many organizations have been successfully using MathML in DITA since version 1.1 became a recommendation.

In recent years, HTML5 and EPUB 3 have been finalized as recommendations and have become popular outputs for DITA documents. As it happens, HTML5 and EPUB 3 both include support for MathML. As support and awareness for MathML have grown and as the DITA community also has grown, it has become increasingly clear that the DITA community would benefit from standardized domain specializations for mathematical content. In response, the DITA Technical Committee has approved the introduction of two new math specializations to DITA 1.3: an equation specialization and a MathML specialization. Presumably if you're using an XML implementation of DITA, you would want to store your mathematical content in MathML, but in recognition of the fact that organizations may have reasons to use other formats for mathematical expressions, the DITA Technical Committee maintains the two separate math specializations.

This paper provides an explanation of the elements of the equation specialization, gives an introduction to the MathML specialization, covers benefits and drawbacks of MathML in DITA, and finally gives an overview of tools to aid you in adding MathML support to your DITA documents today.

Equation Specialization

The equation specialization gives a general way in which to add formulae to your DITA documents. Equations, if numbered, usually follow a different numbering sequence than other figures in your documents; therefore, equations are categorized separately from figures. The DITA Technical Committee recognizes that there are a number of formats in which an equation can be specified: as bitmap images, SVG, or TeX strings, to name a few in addition to MathML. Because of this, the equation specialization is format-agnostic.

In fact, the DITA 1.3 specification allows that one may want to include alternative forms of a single equation within the DITA source. The DITA processor would then be allowed to choose the format most appropriate for the output or use all formats as may be preferable in some outputs. For example, HTML5 currently includes support for MathML, but not all browsers can render MathML as yet. Thus, if you are serving dynamic webpages that can respond to the browser in which the page is being loaded, you have the option of outputting both MathML and fallback images to cover all cases.

A note of caution, however: though DITA 1.3 does allow you to specify alternative formats for a single equation, it does not require that the processor give precedence to any format nor does it deem that precedence should be determined by the order in which the alternative equations appear. Therefore, you may wish to institute your own policies for resolving conflicts or differences that may arise between alternative versions of a single equation.

Block vs. Inline

The equation specialization features two main equation types: `<equation-block>` and `<equation-inline>`. The use of these two equation types implies different formatting for each. The differences between these two equation types are probably intuitive: equation-block is for equations that appear on their own line, typically centered within the page or column, and sometimes numbered; equation-inline, on the other hand, appears in line with the text.

DITA 1.3's `<equation-inline>` element is derived from <ph> and can therefore be used wherever <ph> is available, such as in a general line of text or in a title. DITA 1.3's `<equation-block>` element, on the other hand, is derived from `<div>`, a new element in DITA 1.3. This gives authors a great deal of flexibility in placement of block equations. For example, an equation can occur after a paragraph or within a paragraph:

After a paragraph	**Within a paragraph**
the steps. First, the percentage was converted to a fraction. $64\% = \frac{64}{100} = \frac{16}{25}$ From that point, the problem was much	Thus, the expression $u(x) = \sum_{n=1}^{M} u_n$ is a solution for every M=1,2…

`<p>... to a fraction.</p>`
`<equation-block><image keyref=”percent-to-fraction”/></equation-block>`
`<p>From that point,... </p>`
	`<p>Thus, the expression`
	`<equation-block><image keyref=”summation1”/></equation-block>`
	`is a solution for... </p>`

An equation-block can also contain `<equation-number>`. This will be covered in more detail shortly.

If either <equation-block> or <equation-inline> has more than one child, it should be assumed that each child represents the same equation. For instance, using the following DITA, one would expect that the expression given in the image equation1.png is the same as the expression given by the contained MathML string:

```
<equation-block>
   <image href="equation1.png" />
   <mathml>
      <m:math xmlns:m="http://www.w3.org/1998/Math/MathML">
         <m:mfrac>
            <m:mn>1</m:mn>
            <m:mn>2</m:mn>
         </m:mfrac>
      </m:math>
   </mathml>
</equation-block>
```

Figure

Though <equation-block> and <equation-inline> will likely suffice for most organizations adding mathematics to DITA content, DITA 1.3 also introduces <equation-figure> for more complex mathematical descriptions. Equation-figure, much like the general DITA <fig> element from which it is derived, allows captions or descriptions in the form of titles, paragraphs, and lists.

With <equation-figure>, authors have the option of including the equation directly (such as by adding <mathml> or <image> as a child element of <equation-figure>) or by including the equation as an <equation-block> or <equation-inline>. The <equation-figure> element is an option for grouping equations by containing multiple equation-blocks. As with equation-block, equation-figures can be numbered. Furthermore, the numbering of an equation-figure can be separate from the numbering of equation-blocks contained within an equation-figure.

$$\mathcal{L}^{-1}\left[F(s)G(s)\right] = \int_0^t f(t-z)g(z)\,dz \quad (8.1)$$

$$= \int_0^t g(t-z)f(z)\,dz \quad (8.2)$$

Laplace Transform, with acceptable functions f(t) and g(t).

```
<equation-figure>
  <title>Laplace transform, with acceptable functions
    <equation-inline>
      <mathml>...</mathml>
    </equation-inline>
    ...
  </title>
  <equation-block>
    <equation-number/>
    <mathml>...</mathml>
  </equation-block>
  <equation-block>
    <equation-number/>
    <mathml>...</mathml>
  </equation-block>
</equation-figure>
```

Equation Numbering

DITA 1.3 accommodates the numbering of equations with the `<equation-number>` element. Authors have the choice of specifying that an equation number should be auto-generated by leaving `<equation-number>` empty or by explicitly providing the equation number for a given formula by providing content for the `<equation-number>`; e.g. `<equation-number>3.1.2</equation-number>`. If an equation number is explicitly provided, it is recommended that only the equation number be listed so that any punctuation for demarcation purposes be added by the DITA processor for the sake of consistency.

$$R_{k,j} = \begin{cases} R^*_{k,j} = h\left(\dfrac{\theta^*}{\sigma^2_{k,j}} Q'(0)\right), & \text{if } 0 < \theta^* < \sigma^2_{k,j} \\ 0, & \text{if } \theta^* \geq \sigma^2_{k,j} \end{cases} \quad (3.6.1)$$

```
<equation-block>
  <equation-number>
     3.6.1
  </equation-number>
  <image href="eqn1752.png"/>
</equation-block>
```

The DITA 1.3 specification allows for some flexibility on the part of the DITA processor, as it should. The only guidance it provides is with the statement, "common practice is to present the equation number to the right (in left-to-right reading order) of the equation, centered vertically within the vertical extent of the block equation." When DITA processors provide support for equation numbering, hopefully content architects will be given wiggle room to customize how the equation number is generated and formatted.

MathML in DITA

MathML, the W3C standard for encoding mathematical information as XML, is a natural fit for DITA XML, but as with any organizational decision, one should be aware of the pros and cons of moving to MathML for mathematical content before making the leap from a different storage format.

Advantages of MathML

Since MathML is XML, by using MathML you'll derive many of the same benefits for mathematics as you derive from the rest of your content being in XML: separation of style and content, localization, and reuse. In addition, MathML is required for accessibility purposes and is the format of choice for searchability of your content.

MathML Is XML

First and foremost, MathML allows the separation of the style of your equations from the content, giving you the opportunity to make document-wide changes to the formulae with XSLT or other stylesheets. Applying the style of the equations during the output process allows for professional-looking documents in all delivery formats, in which the formulae can match the surrounding text. For example, some organizations prefer to use sans serif fonts for web output but serif fonts for print. Considering that mathematical expressions are generally an integral part of the document, further explaining information introduced in the text, the mathematical expressions will often need to be in the same font for alphanumeric characters as used in the main body of the text, in this case sans serif fonts on the web and serif fonts in print.

Secondly, MathML eases your localization burdens. While math is a universal language and semantically one expression will carry the same meaning in one spoken language as another, the notation to express a given formula may vary from region to region. For instance, abbreviations for trigonometric functions will look different depending on where you are. Though tan() is generally used for the tangent function in the United States, variants in other locales include tn(), tag(), and tang(). Again, a stylesheet can be used to easily localize the notation in your mathematical expressions. Furthermore, if your mathematical expressions include words that need to be translated, the use of MathML can speed the translation process because only the words in the formulae will need to be translated rather than requiring a re-creation of an image with the new word in place.

As with the rest of your DITA content, you can take advantage of re-use of mathematical expressions that are stored in MathML. Since MathML is a standard, it has been implemented in a wide variety of applications including computer algebra systems, such as Mathematica and Maple, document systems, and educational software. If your team is using engineering software in its workflow, it's possible that the MathML can be auto-generated. If so, generating the MathML may save content authors some time. Additionally, should you need to consider a different venue for your content (such as, turning a journal article into a presentation), chances are high that you will be able to re-use the MathML created originally.

Accessibility

At some point you may face government regulations enforcing the accessibility of your content for audiences with special needs. MathML is the encoding format of choice for the accessibility community because MathML marks up every single component of a mathematical expression allowing assistive technology, including screen readers and braille translation software, to navigate and translate the expression to aid comprehension for blind, low vision, and learning disability audiences. This is why accessibility standards such as DAISY, NIMAS, and PDF/UA require mathematical information be encoded as MathML. If you are using MathML in your DITA source, you will be better prepared to meet any mandates for accessible content.

Searchability

As a text format, MathML generally allows better search and replace functionality than some other formats, especially images. Research is currently underway for better math search capabilities that would allow a search for *a*+*b* to not only turn up results with *b*+*a*, but also *x*+*y*, and so on. Improved search of mathematical expressions could have a range of applications, from use as an educational tool to finding similar patterns between remote disciplines. The detailed markup of MathML appears to be the path forward towards improved search functionality, allowing canonicalization of expressions for comparison purposes. See *MathWebSearch* (search.mathweb.org) for further information. By using MathML in your content now, you can not only take advantage of basic text search today, but also prepare your content for improved math search and other developments.

Disadvantages of MathML

While moving to MathML for mathematical content does offer a number of solid benefits, it may require a paradigm shift in the way organizations approach mathematical content. In particular, content teams seem to have difficulties with inconsistencies between rendering engines and with the shift in thinking required for authoring MathML.

Differences in Rendering Engines

The cost of the flexibility that MathML offers as discussed in the previous section is that MathML is often handed off to different rendering engines. On the upside, when a mathematical expression is rendered at the time the document is loaded, the rendering engine can take into account the environment (font settings, background and foreground colors, pixel density, and so on) and can display the equation so that it matches the surrounding document. This provides for a seamless user experience with high quality output. On the downside, however, your content is subject to the characteristics of whatever rendering engine by which your content is being processed. In other words, you lose some control over the output.

If you have any experience in web design, you are familiar with this phenomenon: your web content may look slightly different depending on which browser a web page is loaded in. This is often true of MathML, as well. Though the MathML specification provides rules for rendering engines in some situations, it leaves a lot of room for interpretation. Therefore, what looks correct in one MathML rendering engine might look different in another.

One example of this is the stretchy arc used in geometry: $\overset{\frown}{ABC}$. In Unicode, one can find a few codepoints that could potentially be used for the arc character, but, due to font limitations, most rendering engines will only support the stretching of one of those arcs. The MathFlow rendering engine will only stretch Unicode character U+02322, while the MathJax rendering engine will only stretch Unicode character U+023DC. This issue can be addressed by pre-processing the MathML before it is loaded into one or the other rendering engine, but this is the sort of problem that you may have to prepare for when adopting MathML.

In addition to the general rendering discrepancies, there is also the matter of which version of MathML the rendering engines support. Most rendering engines support MathML 2.0, but some, such as Apple iBooks, only support a subset of the MathML 2.0 standard. Developers are still in the midst of adding support for MathML 3.0, which became a W3C recommendation a few years ago, and most are adding support for MathML 3.0 in stages. MathML 3.0 introduced some long-awaited features including new features for elementary math notation, line breaking, and right-to-left languages. To give you an idea of the range of MathML version support, at the time of writing, the popular open-source MathML rendering engine jEuclid, which is not under active development, only supports MathML 2.0. MathJax has some support for MathML 3.0's elementary math notation and for most of MathML 3.0's linebreaking functionality. Finally, the MathFlow Windows rendering engine has extensive support for MathML 3.0.

Authoring MathML

Teams have a number of options for generating MathML. One could use a text editor and manually write the MathML. One could write TeX/LaTeX and convert it to MathML. A standalone equation editor such as MathType or MathMagic could be used with conversion to MathML. Most likely, however, you will use a dedicated MathML editor, such as MathFlow. No matter how you choose to create your MathML, equation authors will need to keep in mind the hierarchical nature of MathML, its rules, and its limitations. Though MathML editors generally hide the MathML as much as possible to ease the transition to MathML, just as with moving to any XML authoring system, users of MathML may need to adjust how they think about the mathematical expression to better work with MathML.

For example, some equation editors allow an author to create a sub- or superscript that is not attached to a base. This feature often is employed when an author is trying to stagger a sub- and superscript on the same base, e.g. x_m^n. A MathML authoring system, on the other hand, cannot allow someone to simply place the *n* in the preceding example in the superscript position without a base. This would create an invalid MathML string. The author must understand that the *n* superscript must be attached to either the base *x* or the base x_m. Until an author understands minor rules such as this one, the authoring experience, because it is different, can be a frustrating one. Some training may be necessary to ease the transition.

The DITA 1.3 MathML Specialization

The new MathML specialization in DITA 1.3 simply standardizes what many organizations have already put into use: a DITA `<mathml>` element that is derived from `<foreign>` serves as a container for MathML. If the MathML is entered directly into the DITA document, the use of a namespace prefix is required on the MathML string to help with parsing of the document by DITA processors. Furthermore, the use of named entities (such as, α for the Greek alpha character) is discouraged when MathML is directly included. In fact, the DITA 1.3 schemas ignore the entity definitions in the MathML schemas, thus named entities will cause validation and parsing errors. It's better to use numeric references (e.g., α for alpha) unless you are referencing an external document.

Alternatively, the `<mathml>` element can contain a `<mathmlref>` DITA reference that either references the location of an external non-DITA document with MathML in it directly using `@href` or indirectly using `@keyref`. Note that the external document can either contain a single MathML string with the MathML `<math>` element as its root, or it can contain a number of MathML strings. In the latter case, you can identify the string to include with the URI hash fragment identifier, referencing the XML ID of a MathML `<math>` tag.

XML Referenced	DITA 1.3
EquationLibrary.xml	**@href**

`<?xml version="1.0" encoding="UTF-8"?>`
`<library>`
`  <math`
`  id="equation_01"`
`  xmlns="http://www.w3.org/1998/Math/`
`MathML">`
`  ...`
`  </math>`
`  <math`
`  id="equation_02"`
`  xmlns="http://www.w3.org/1998/Math/`
`MathML">`
`  ...`
`  </math>`
	`<equation-block>`
	`  <mathml>`
	`  <mathmlref`
	`  href="EquationLibrary.xml#equation_02"`
	`  />`
	`  </mathml>`
	`</equation-block>`

XML Referenced	DITA 1.3
`</library>`	

`<keydef`
`  keys="Equation_02"`
`  href="math/EquationLibrary.xml#Equation_02"`
`  format="mathml"`
`/>`
	@keyref
	`<equation-inline>`
	`  <mathml>`
	`  <mathmlref keyref="Equation_02"/>`
	`  </mathml>`
	`</equation-inline>`

Whether including the MathML directly or referencing an external document, the MathML string must start with the MathML `<math>` element.

MathML Output

As of the writing of this article, the DITA Open Toolkit (DITA-OT) does not yet support MathML, but MathML support is expected to be added soon. If you cannot wait for a DITA-OT release with official support for MathML, you can use the *DITA For Publishers* (http://dita4publishers.sourceforge.net/) open source project as a model for your customization of the DITA-OT. DITA For Publishers is an extension of the DITA-OT developed by Eliot Kimber primarily for publishers who use DITA in document workflows. The most recent version of DITA For Publishers does include a MathML specialization and supports MathML to HTML output using MathJax (see below). The DITA For Publishers MathML specialization, however, was developed prior to DITA 1.3's equation specializations. Thus, some updates will need to be made to the DITA Open Toolkit plug-in before it can be used with the standardized specializations.

PDF Output

Most users of DITA and the DITA-OT will use one of three XSL-FO engines when producing PDF output: Antenna House Formatter, Apache FOP, or RenderX XEP. Antenna House's Formatter is the only one of these three XSL-FO processors that has native support for MathML. If you use Apache FOP or RenderX XEP, you will likely want to use one of the independent MathML rendering engines discussed below to pre-process the MathML before converting your DITA to XSL-FO and PDF. The jEuclid MathML rendering engine does have an Apache FOP plug-in, but it requires a version of Apache FOP that is now several years old.

HTML-based Outputs

Some browsers include native support for MathML, and because e-readers are often built on browser technology, some e-readers also will display MathML. A classic example is Apple's Safari browser and iBooks e-reader application; both use the WebKit rendering engine, which has limited support for MathML, and, therefore, both can display the MathML in your web/ebook content.

Unfortunately, not all browsers and e-readers are so advanced. Notably Internet Explorer and Chrome fail to render MathML. For maximum coverage you'll need to implement a different solution other than relying on native support for MathML. You have the option of 1) converting the MathML to an image during the DITA conversion process using one of the independent rendering engines described below, 2) setting up MathML-to-image conversion on the web server for processing of the MathML at load time, or 3) using the Javascript MathML rendering library, *MathJax* (mathjax.org).

MathJax is a free open-source library for rendering MathML in all modern browsers. To use MathJax with your content, you merely need to link your web pages to the MathJax library, either installed on your own server or hosted on the MathJax CDN site. Using MathJax allows you to leverage the benefits of using MathML as previously discussed: by serving the equations as MathML your mathematical content will remain searchable and accessible to audiences with special needs. The primary complaint about MathJax is its speed, or lack thereof, especially on pages with a large number of equations. The MathJax Consortium is working to address

this complaint and in the latest release (version 2.5) has reported speed improvements of up to 40% over the previous version.

Independent MathML Rendering Engines

You do have a few options to choose from should you need to use an independent MathML rendering engine. The one you choose will likely depend on your requirements and resources.

fMath

The *fMath* (www.fmath.info) rendering engine is a free application under active development. It's primarily intended for use on a web server, with libraries in Java, Javascript, Flash, and C# producing PNG or Flash output. fMath does include some support for MathML 3.0.

jEuclid

The *jEuclid* (jeuclid.sourceforge.net) rendering engine is a free open-source Java application that is not being actively developed. For this reason, it does not include any support for MathML 3.0. jEuclid can produce JPG, PNG, and BMP output files, and with the help of extension libraries can also produce SVG, PDF, and SWF files among other file formats. jEuclid can be used as a command line utility or can be accessed through its API.

MathFlow Equation Composer

The *MathFlow Equation Composer* (dessci.com) is available as Java and Windows libraries, and, like jEuclid, can be run from a command line or through an API. The Windows Equation Composer rendering engine was used as the benchmark for testing of the MathML 3.0 test suite. It is the most advanced MathML rendering engine available, but it is also a commercial application. The MathFlow Equation Composer can produce GIF, PNG, and EPS output.

Conclusion

The new math specializations in DITA 1.3 are a welcome addition to DITA for organizations that have mathematics in their content. Having a standard way in which to insert formulae will shorten DITA implementation time and provide more portability for your DITA content. The math specializations are separated into two specializations that can be used in conjunction: the equation specialization provides a general method, regardless of the format of the equation, for typesetting equations in either block or inline style and providing the option of adding captions and equation numbers, while the MathML specialization allows for adding MathML to your DITA XML document.

Most organizations will choose to adopt the MathML format for equations because of the benefits it offers. Since MathML is XML it has the same advantages as XML: MathML allows separation of content from style, is cheaper to localize, and is a standard that can be re-used in other applications. In addition, MathML broadens your audience with its use in accessibility standards and its searchability. However, organizations that adopt MathML may also have to contend with inconsistent rendering of expressions and the need to train authors.

Though the DITA-OT does not currently have support for MathML, a number of tools are available to handle the MathML in your content, including Antenna House Formatter and MathJax, as well as independent rendering engines fMath, jEuclid, and MathFlow Equation Composer. This author expects that support for MathML will continue to grow as organizations and software vendors recognize the benefits to be gained from adding it to XML and DITA workflows.