MathML and other XML Technologies for Accessible PDF from LaTeX

Frank Mittelbach
frank.mittelbach@latex-project.org
LaTeX Project
Mainz Germany

Ulrike Fischer
fischer@troubleshooting-tex.de
LaTeX Project
Bonn Germany

David Carlisle
d.p.carlisle@gmail.com
LaTeX Project
Oxford UK

Joseph Wright
joseph@texdev.net
LaTeX Project
Ely UK

Abstract

In this paper we describe the current approach to using MathML within Tagged PDF to enhance the accessibility of mathematical (STEM) documents. While MathML is specified by the PDF 2.0 specification as a standard namespace for PDF Structure Elements, the interaction of MathML, which is defined as an XML vocabulary, and PDF Structure Elements (which are not defined as XML) is left unspecified by the PDF standard. This has necessitated the development of formalizations to interpret and validate PDF Structure Trees as XML, which are also introduced in this paper.

CCS Concepts

• **Software and its engineering** → *Open source model*; • **Applied computing** → **Format and notation**; **Document metadata**; • **Human-centered computing** → **Accessibility technologies**; • **Information systems** → **Document structure**.

Keywords

Accessibility, PDF/UA, Well Tagged PDF, LaTeX, Mathematics, XML

ACM Reference Format:
Frank Mittelbach, Ulrike Fischer, David Carlisle, and Joseph Wright. 2025. MathML and other XML Technologies for Accessible PDF from LaTeX. In *ACM Symposium on Document Engineering 2025 (DocEng '25), September 2–5, 2025, Nottingham, United Kingdom.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3704268.3748669

1 Introduction

Traditional mathematical notation requires specialist accessibility support to navigate the two-dimensional layout and to express the large set of symbols as speech or Braille. PDF 1.x accessibility standards treat mathematical formulae essentially as images. Accessibility is just provided by supplying a text string. Such a string does not allow navigation into subterms, and is usually unsuitable for Braille.

To address this, PDF 2.0 introduced the possibility of embedding MathML within the PDF, either as an associated (embedded) file attached to each formula, or as PDF Structure Elements, using tag names modelled on the standard MathML element names.

In this paper we discuss the current approaches taken to enhance PDF documents produced by LaTeX with MathML tagging, and more general issues relating to interpreting PDF Structure Elements as XML for validation or other purposes.

2 MathML in PDF

The PDF 2.0 accessibility standard, PDF/UA-2, mandates that all mathematical content is associated to MathML, either as an Associated File or tagged using the Structure Elements from the MathML namespace (which it does not define, other than by reference to the MathML Recommendation).

Using Associated Files to embed MathML is relatively simple, as the MathML may be produced by external conversion from the original sources (several such tools exist) and simply referenced by a property on the PDF Formula tag. If this is passed to an AT (Assistive Technology) system it will allow the mathematics to be spoken, or Braille generated. However currently not many PDF readers will recognize files embedded in this way, and as the MathML is just associated with the start of the formula, navigating the accessible version of the mathematics cannot be synchronised with the navigation of the visual rendering.

Recent releases of LaTeX have included the luamml package which, if LuaTeX is used as the typesetting engine, will convert the TeX mathematics to MathML during the typesetting phase. This can be used to create the Associated File and embed it during the compilation without external tools.

The luamml package also now allows the use of the second method of associating MathML, where each term of the expression is separately annotated with PDF Structure Elements using the MathML namespace.

The PDF 2.0 standard unfortunately just states that MathML PDF Structure Elements may be used, but it gives no definition of these elements. Despite some similarities (both PDF Structure Elements and XML encode annotated trees) there is no formal definition of how to construct these MathML elements, which by definition are XML, as PDF Structure Elements. The LaTeX Project have proposed an explicit mapping between PDF and XML elements documented at [6]. This has been agreed with the relevant working groups at the PDF Association and hopefully will form the basis of a future publication.

However despite the preliminary nature of this mapping specification, there is already a working agreement between implementations that MathML encoded as PDF Structure Elements in this way will be recognized by AT systems. In particular NVDA screen reader and Adobe Acrobat PDF viewer may be used with files encoded in this way, and support in the Foxit PDF Reader is being developed.

These recent developments in LaTeX, together with parallel developments in screen reader applications (NVDA and MathCAT), PDF readers (Foxit) and coordination and clarification on the PDF specifications within the PDF Association Technical Working Groups, mean that there is now a complete workflow available for accessible technical or mathematical documents. Starting from a LaTeX source, one may generate well tagged PDF in which the document structure, including mathematical formulas, are exposed in an accessible way to screen readers, Braille displays and other accessibility tools.

2.1 MathML Attributes

The encoding of MathML attributes causes particular difficulties. Some attributes such as `id` correspond to PDF Properties although the PDF Property to encode a unique identifier is `ID` so even here there has to be some name mapping. But more general attributes are encoded using the PDF Structure Attributes, which are quite unlike XML attributes, having names consisting of two parts, an *Owner* and a *Name*, and a value that may reference arbitrary PDF Structures. In common with most other XML vocabularies MathML attributes are unprefixed and not in a namespace. However by agreement with the PDF Technical Working Groups, the representation in PDF is using the MathML namespace. This complicates the mapping and the full details are in the web page referenced above.

2.1.1 PDF Properties. PDF Structure Elements (SE) have a small number of pre-defined Properties. The most important in this context being ID and Lang. These are directly mapped to XML attributes, but with some name changes so the resulting XML is consistent with existing schema for XHTML and MathML. The full list is given by the following table

SE Property	XML Attr.	SE Property	XML Attr.
AF	af	Lang	lang
ActualText	actualtext	Phoneme	phoneme
Alt	alt	PhoneticAlphabet	phonetic-alphabet
C	class	R	revision
E	expanded	T	title
ID	id		

2.1.2 PDF Structure Attributes. In addition to the Properties shown in the previous section, PDF Structure Elements may have *Attributes*. In many cases the attribute value is a simple value such as a string or a number that may be directly encoded in XML, but arrays of values are also possible, and in general a PDF Structure Attribute may reference any PDF Object including binary streams that have no natural encoding as an XML attribute.

However for practical reasons, these Structure Attributes are modelled as XML attributes by the following mapping.

- A PDF Structure Attribute with Owner *owner* and Name *name* is represented by the XML attribute `owner:name` where the prefix `owner:` is associated to the namespace URI `http://iso.org/pdf/ssn/`*owner*.
- PDF 2.0 introduced an extension mechanism to allow systems to add new attributes. If the owner is the special Keyword NSO (Namespace Owner) then a namespace URI is supplied in an additional NS property. The following convention for representing attributes in no-namespace has been agreed with the PDF UA Technical Working Group.
  If the Owner is NSO then the attribute is represented in XML by an attribute in the namespace given by the NS property *unless* the NS property references the same namespace object as the parent element, in which case an attribute in no-namespace is used. This allows a MathML element such as `<mo lspace="2pt">` to be represented by a Structure Element mo having NS property referencing the MathML namespace, and attribute lspace with O (owner) NSO and NS referencing the same PDF namespace object as the mo element.
- The value of the PDF Structure Attribute must be linearized into an XML compatible string and represented as the value of the XML attribute.

2.2 Intent

One of the main novel features in MathML 4 [1] is a new attribute `intent` which allows the MathML generation system to add annotations.

A main motivating example is the notation $|x|$ which may denote the absolute value of x or the cardinality (size) of x or several other meanings depending on context.

This may now be encoded as

```
<mrow intent="absolute-value($x)">
  <mo>|</mo> <mi arg="x">x</mi> <mo>|</mo>
</mrow>
```

As well as disambiguating the meaning, the intent system allows AT systems to provide localised readings of the formula. Also (unlike earlier approaches of providing alternative text strings, as used for PDF 1.x) this does not interfere with Braille production. In a Braille display you need to generate a compact string corresponding directly to the notation `|x|`. The Braille for the English language "alt-text" "`a-b-s-o-l-u-t-e-v-a-l-u-e o-f x`" would be impossibly verbose.

LaTeX can already add these `intent` annotations in some cases but longer term the expectation is that packages providing commands such as `\abs` will be able to specify intents to use for tagging contexts as part of the LaTeX command definition.

One important use of `intent` that is already automatically added by LaTeX is the encoding of displayed equations. A display such as

$$a + b = c \tag{3}$$
$$c = d \tag{4}$$

is encoded in MathML as an `mtable` element, the same element used for all table constructs such as matrices. LaTeX will now generate `<mtable intent=":system-of-equations">` which allows AT systems to announce this as a system of equations, not as a 2-by-2 matrix. Similarly the equation numbers (3) and (4) are marked with `<mtd intent="equation-number">`, which allows each line of the display to be announced as "line with label ..." rather than the "(3)" being read as part of the mathematical data of the formula.

The result of LaTeX embedding this MathML with additional `intent` annotations is that the above display is read by the screen reader (using NVDA and MathCAT from Acrobat or Foxit PDF viewers) as

```
line 1 with label 3; a plus b is equal to c
line 2 with label 4; c is equal to d
```

Note this requires no additional markup from the document author.

The display structurally is a three column table: one column for each side of the equation, and an extra, first, column for the equation number. If the `intent` of the `mtable` is changed to `:matrix` the reading is

```
the 2 by 3 matrix;
row 1; column 1 3; column 2 a plus b; column 3 is equal to c
row 2; column 1 4; column 2 c; column 3 is equal to d
```

If neither `:system-of-equations` nor `:matrix` is specified as an `intent`, the reading relies on the underlying AT system to infer the correct structure. In practice, it is recognized as a system of equations, but the equation labels are not announced and the equation numbers are read as part of the mathematical expression, which can be very confusing for the reader.

See [8] for a list of all the core properties currently being developed by the W3C Math Working Group for MathML 4. MathML 4 is not yet a final W3C recommendation; however, these properties are already implemented by MathCAT and can be demonstrated when reading PDF files from Acrobat or Foxit PDF readers.

2.3 Hiding displayed content from AT

Conceptually to tag a mathematical expression you just need to tag each term with appropriate MathML, so $x + y$ might be encoded in XML as: `<math><mi>x</mi><mo>+</mo><mi>y</mi></math>`.

In the PDF stream the MathML elements are represented by Structure Elements but the text content is not directly added to the Structure Tree, it is represented by references to the content stream used for the typeset display, so called *Marked Content Sequences*.

This can cause difficulties for square roots and other constructs where visual components correspond to the markup and not just the character data. An expression such as $\sqrt{x}$ will have items in the content stream used for typesetting corresponding to a $\sqrt{}$, to a rule $\overline{}$ and to x. As the MathML is `<msqrt><mi>x</mi></msqrt>` only the x should be referenced from the Structure Tree. The other items all need to be marked as *Artifact* in the content stream so that they are not used in the AT rendering. For a large range of expressions LaTeX can automatically detect such cases and add appropriate tagging, but due to the expressive nature of the TeX input, this is hard to get right in all cases and further improvements are expected in this area. Similar issues arise with large delimiters, e.g.,

$$X = \begin{pmatrix} a \\ b \end{pmatrix}$$

The generated MathML for the large left parenthesis needs to be `<mo>(</mo>` with a standard "(" character, however, there is no such character in the content stream that can be referenced. The PDF content stream will have several glyphs making up parts of the extendable character stacked vertically. So it is necessary to both hide these glyph parts and to provide a standard character. This is achieved by referencing the content stream from the mo Structure Element as normal, but marking the Content Stream with an ActualText Property of "(". ActualText is a standard PDF Property that causes the marked content to be replaced by the specified "(" in most uses other than rendering, in particular when generating content for AT or for cut-and-paste from the PDF.

2.4 Validation of the MathML

As PDF Structure Elements are embedded in the PDF document and do not correspond to any existing markup system such as SGML or XML, there is no standard way to validate the structure in a PDF document. PDF validators will not validate MathML embedded as an Associated File, other than checking that the reference is valid. If MathML is encoded using MathML Structure Elements then it will either be classed as invalid (as the system does not recognize MathML) or the whole MathML expression may be accepted even if the MathML encoding is incorrect as no checking of the MathML structure is done. To address this issue the LaTeX Project have utilised the mapping from PDF Structure Trees to XML discussed in [6] and implemented (using the Lua interface to PDF files provided by LuaTeX) a tool, `show_pdf_tags`, to extract an XML representation of the Structure Tree of any Tagged PDF file. XML (Relax NG) Schema have been developed capturing (as far as possible) the constraints described in the PDF standards and including the standard MathML Schema that is described in the MathML Recommendation [1]. Currently the Schema for MathML Core [9] is used, as this is the version of MathML implemented in web browsers, and includes the additional attributes for accessibility such as `intent` discussed above. The tool and the Relax NG Schema are described in more details in the next section.

3 Relax NG Schema for PDF Structure Trees

Expressing a PDF Structure Tree as XML has many benefits in addition to the original motivation of checking the embedded MathML in mathematical documents. Having the structure in a standard format such as XML allows it to be validated or queried by a far wider set of tools.

The extraction lua script `show_pdf_tags` and the Relax NG Schema are all available as Open Source software on GitHub [5], and are also currently provided as an online service [4] which allows any Tagged PDF file to be uploaded. The XML representation of the Structure Tree is then shown in the browser along with the result of validating the XML with Relax NG Schema.

The structure of the Schema mostly follows the containment rules in ISO 32005 [2] (including proposed updates). The elements are in the "`pdf2`" namespace (`http://iso.org/pdf2/ssn`) as specified by the PDF 2.0 standard. The schema ensures that if PDF 2 elements such as Em are used in the PDF 1.x Schema the element must have a *role mappping* to a standard PDF 1.x Structure Element such as Span element to meet the requirements of the relevant accessibility standard PDF/UA-1. The schema also allows validation of the standard PDF Structure Attributes (which are not mentioned in the containment rules in ISO 32005). For example, PDF Structure

Attributes include TextAlign with Owner Layout, which is represented in XML with a Layout namespace and a Schema fragment

```
attribute Layout:TextAlign
   { "Start" | "Center" | "End" | "Justify"}?,
```

which allows both the attribute name and attribute value to be validated. Currently all standard Structure Attributes that take an enumerated list of values are represented in the same way.

Attributes that take more general values such as lengths or color expressions do not currently have the values constrained by the Schema, typical examples would be

```
attribute Layout:TextDecorationColor {text}?,
attribute Layout:TextDecorationThickness {text}?,
```

This allows the attribute name to be validated, but any value is currently allowed by the Schema. As PDF lengths are just expressed as floating point numbers they could potentially be directly validated in the Relax NG Schema by replacing "text" by a reference to a numeric type. Similarly, as Relax NG may constrain values with regular expressions, it would be possible to at least constrain color expressions to invalidate clearly invalid expressions although this is not attempted by the current Schema.

3.1 Extension Schema for Custom Elements

Tagged PDF documents may utilize custom Structure Elements defined by the generating application and identified by a namespace URI specified by the authoring system. Any such custom element must have a *role mapping* to another element, and these Role maps must eventually resolve to one of the Standard Structure elements from PDF 1.7, PDF 2.0 or MathML namespaces.

The role map system allows Tagged PDF documents to use custom tags targeted at specific systems, but with an automatic fallback mechanism so they may be used by systems that do not implement the custom tags.

A particular use of custom elements is a set of elements being developed for PDF generation from LaTeX. These closely model the structure inherent in the source document, while allowing graceful fallback to the more generic Standard Structure Elements defined by PDF. A schema that encodes the current set of elements used is already available and described in [6].

As the PDF specifications provide no mechanism to specify the constraints on element nesting that should apply to these custom elements, existing PDF validators almost all rely on the role mapping and just validate the structure implied by the Standard Structure Elements. Using XML to represent the PDF Structure Tree as described here makes available standard validation technologies such as DTD, W3C Schema and Relax NG Schema. These are all designed to easily specify languages that are extended by addition of new elements to an existing vocabulary.

A simple example in the LaTeX collection of Tagged PDF files [3] is a tagged version of the (American Standard Version) Bible. This uses custom Structure Elements: Testament, Book, Chapter, Verse. Each of these elements is *Role mapped* to the generic Sect Standard Structure element for sectioning. While the PDF may be validated< using standard PDF tools such as VeraPDF [7], the validation would still have succeeded if the nesting of the custom elements had been incorrect and, for example, a Book was nested inside a Chapter, because after role mapping these are all Sect.

Conversely it is easy to validate the extracted XML representation against a `latex-bible` schema that first imports the general LaTeX schema, then defines the element Chapter to have a title, possibly some preliminary text, and then a sequence of Verse, so an attempt to nest a Book within a Chapter would lead to a validation error.

4 Summary and Outlook

With the ability to represent mathematical content as MathML in the PDF output, LaTeX is now capable of producing accessible STEM documents in an automated fashion, in particular without the need for any post-processing activities.

This is an important achievement, because only through automation that does not require undue extra effort by the document author (who often does not see accessibility as an important goal but as an enforced burden) there is a chance that most, if not all, STEM documents become accessible from the outset.

While full automation that generates a reasonable and usually accurate representation is important as a basis, there are limits to what can be achieved in this way. This is similar to LaTeX offering reasonable default structures for describing documents. It is clear that this is not enough to represent the structural nuances of all conceivable documents. LaTeX therefore offers ways to define new and more granular structures. In a similar way, the current automatic production of accessible documents needs augmentation to allow the document authors to adjust, correct, or extend the interpretation of the LaTeX source, especially if enhanced with author-defined structures, but also if, for example, the default interpretation of a mathematical construct does not reflect the usage in the author's field.

Related to this necessity for extended structures, the interpretation of PDF Structure Trees as XML and the ability to validate such trees using an extensible grammar, for the first time, gives the possibility of specifying and validating custom tagging structures in PDF, independently of whether they are generated by LaTeX.

Researching which accessibility improvements are necessary and developing from that data a concise and easy to use interface will be among the tasks to further improve PDF accessibility for STEM documents generated from LaTeX.

References

[1] David Carlisle (Ed.). 2025. *Mathematical Markup Language (MathML) Version 4.0*. https://www.w3.org/TR/mathml4/
[2] ISO 2023. *ISO/TS 32005:2023* (1st ed.). ISO. https://iso.org/en/contents/data/standard/04/58/45878.html PDF 1.7 and 2.0 structure namespace inclusion. https://iso.org/en/contents/data/standard/04/58/45878.html.
[3] LaTeX Project Team. 2024. WTPDF / PDF/UA-2 Examples by the LaTeX Project. https://github.com/latex3/tagging-project/discussions/72.
[4] LaTeX Project Team. 2025. PDF Structure Tree Display and Validation. https://texlive.net/showtags.
[5] LaTeX Project Team. 2025. The `show_pdf_tags` Tool. https://github.com/latex3/pdf_structure.
[6] LaTeX Team. 2025. Interpreting a PDF Structure Tree as XML. https://github.com/latex3/tagging-project/discussions/789
[7] veraPDF consortium. 2024. Industry Supported PDF/A Validation. https://verapdf.org.
[8] W3C Math Working Group. 2025. MathML Document Repository – Core Intent Property List. https://w3c.github.io/mathml-docs/intent-core-properties/.
[9] W3C Math Working Group. 2025. MathML Document Repository – Schema for MathML Core. https://github.com/w3c/mathml-schema/blob/main/xsd/mathml4-core.xsd.

Received 3 June 2025; accepted 10 July 2025