Ongoing efforts to generate "tagged PDF" using pdfTeX

Ross Moore

Mathematics Department, Macquarie University, Sydney, Australia
ross@maths.mq.edu.au

Abstract. Recently PDF has been accepted as a standard for production of electronic documents, as ISO 32000-1:2008, with an acronym of PDF/UA (for "Universal Accessibility"). The second draft ISO 32000-2:2009 is to include specifications for including MathML tagging of mathematical environments and expressions. This talk presents a report on work-in-progress, aimed at:

(a) developing the primitive commands for pdfTeX needed to support the production of fully tagged PDF documents;
(b) writing appropriate TeX and LaTeX macros to make effective use of the new primitives;
(c) authoring changes to internal LaTeX structures to use these macros automatically at appropriate places within the existing code-base for LaTeX.

This is work that is being undertaken together with Hàn Thế Thành, author of pdfTeX [2], who has added some new primitive commands to an experimental version of this software tool.

1 Background

In July 2008, Adobe's PDF Reference 1.7 [1] became ISO 32000 [4]. Since 2005, the PDF Reference 1.4 has served as the basis for ISO 19005 [3], as an archival format for technical documents. Both of these standards rely heavily on "Tagged PDF", so that not only is the content displayed at the highest quality, but also its structure is provided, allowing for selective extraction of content and "reflow" on small-screen devices (such as a PDA or modern mobile-phone), and screen-reading perhaps in alternate languages. Work is under way on revision of ISO 19005, called PDF/A-2, to accommodate extra features introduced with PDF 1.5, 1.6 and 1.7. Furthermore, in November 2008 it was agreed that a revised ISO 32000-2 should include tagging of the structure of mathematical expressions and formulae, using MathML tags. It may take as long as 2–3 years before these updated standards are released in their final form.

TeX and LaTeX remain *de facto* standards for technical documents, particularly those having a large amount of mathematical content, though other methods are starting to gain significant usage. Whilst PDF is the main output

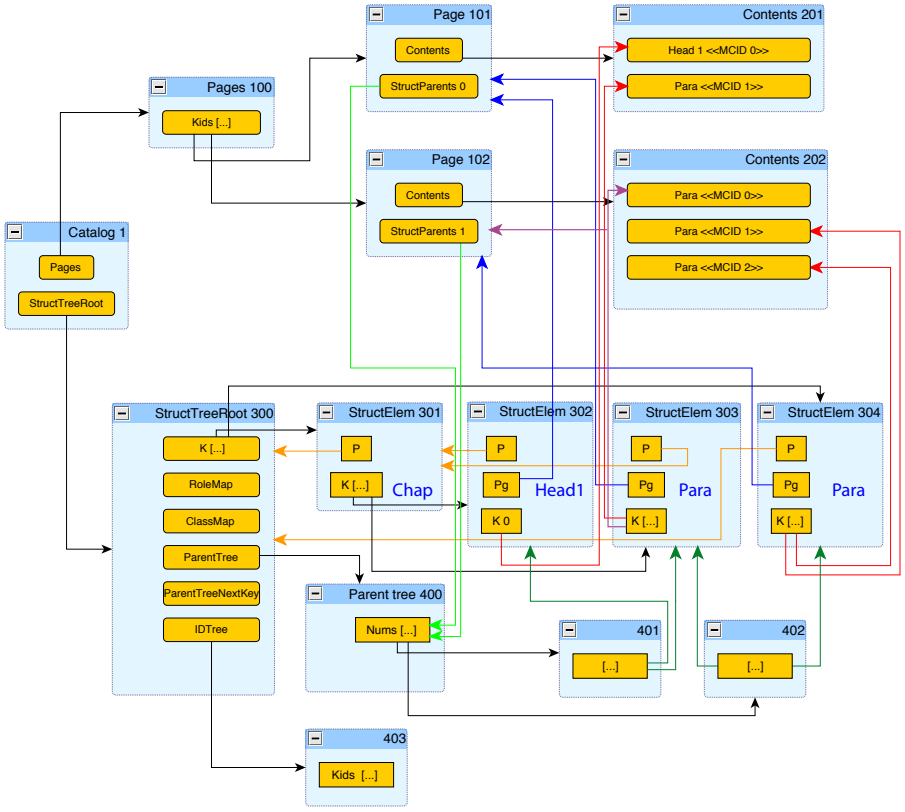

Fig. 1. Interleaving of structure and content tagging within a 2-page PDF document, structured as a heading and two paragraphs. (based on an example in [1])

format for LaTeX, there have been no automated methods to include the structure and content tagging that the above standards require. Work being undertaken here is aimed at providing this missing support when pdfTeX [2] is used as the PDF-producing software. As well as requiring new *primitive* commands to mark content and build the structure trees that are needed for this tagging, a large amount of the LaTeX codebase will need to be revised to take advantage of the new features. In this paper and associated talk, examples are shown of work done so far, towards this aim.

Section 2 gives an idea of how tagging in PDF works, indicating the complexity of the extra structures that PDF-producing software needs to provide; Figure 1 gives a schematic view of these structures, within a document having a quite simple structure. On the other hand, MathML tagging generally requires a much deeper structure tree. In Section 3 an example (see Figure 2) is presented, showing how the MathML tagging is represented, using new TeX primitives and within the PDF, such that it can then be faithfully exported to XML.

2 Tagging PDF documents

Figure 1 indicates the extra structures that need to be created when producing "Tagged PDF". The upper half of the image shows the kinds of object that are needed to display a PDF file as a series of pages. These kinds of objects include:

(i) page content streams, which consist of the low-level commands to select fonts and place text on the page — the blue boxes headed as 'Contents ...';
(ii) an indexing object for each page — the blue boxes headed as 'Page ...';
(iii) an indexing object, headed 'Pages' that acts as a parent for the collection of 'Page ...' objects;
(iv) the previous object is a child of the 'Catalog', which is the root node for the complete document structure.

With tagged PDF there is also a *Structure Tree* whose root node 'StructTreeRoot' is another child of the 'Catalog'. This is itself the root node for a tree of objects headed as 'StructElem ...', which describe the abstract structure of the textual content of the document. Each 'StructElem' node has both references to its children, and a back-pointer to its parent node within the Structure Tree.

To define the content that is encompassed within the structure, there need to be references from the nodes of the Structure Tree to specific locations within the 'Contents' streams. The locations are indicated by the (round) rectangles, with arrows indicating how the structure relates to these. Extra arrows point from structure nodes to 'Page' nodes, which help identify where the content can be seen; that is, on which page does it (mostly) occur.

A second tree is linked-to from the 'StructTreeRoot'; this is called the 'ParentTree', containing a node for each physical page. These nodes are each an array of references to all the 'StructElem' nodes that have content on the corresponding page. There is a link from each 'Page ...' object to the 'ParentTree', which allows the corresponding node to be easily located.

Finally a third 'IDTree' is an optional feature. Each 'StructElem' node can be given a unique name. The 'IDTree' acts as the root node for a tree built up to include arrays of these names, each paired with a pointer to the corresponding 'StructElem' node. This possibility of associating names to structure is for the benefit of Application software that produces or manipulates PDF files. It can use whatever naming scheme it likes to facilitate access to the kinds of structured objects that it needs to work with.

Figure 1 is based on an example in the PDF Reference document [1]. The structure it represents consists of a document section (chapter) having a heading and a paragraph stretching across two pages, together with another paragraph.

3 MathML tagging within a PDF document

Figure 2 shows the effect of having a piece of mathematics tagged (using MathML syntax) within a PDF document. The middle part of the image shows how the page would appear within an Adobe Reader, or (in this case) Acrobat, browser.

This view is partly obscure by Acrobat's 'Order Panel', which displays the tagging of a mathematical expression, with an `<mrow>` selected. The corresponding content is highlighted with rectangles back in the browser view. On the left side we see the result of an 'Export to XML 1.0' action, writing the tagged contents out into a text file. This export has included the mathematical symbols using UTF8 format, so the correct Unicode Plane 1 "Mathematical Alphanumerics" are shown within a text editor that supports the full Unicode range.

This example was produced using an experimental version of pdfTEX. The fonts being used are from the Computer Modern family, which are the standard fonts that have traditionally been used with TEX and LATEX. Mappings to Unicode Plane 1 characters are achieved using the LATEX package mmap.sty, described within a recent TUGboat article [5].

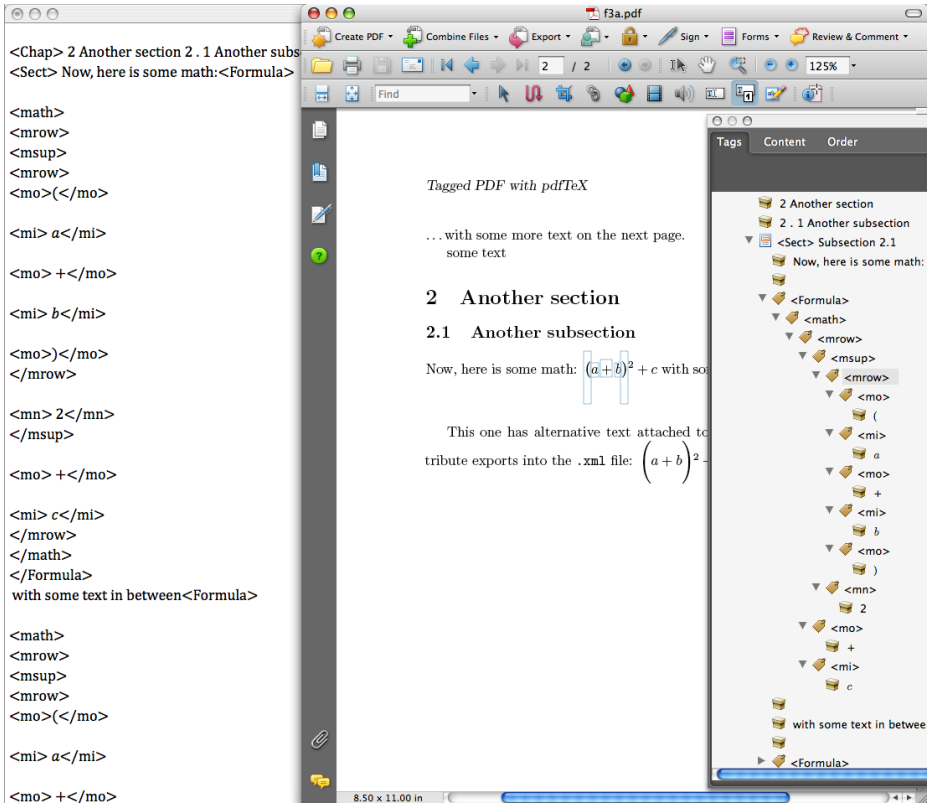

Fig. 2. MathML tagging within a PDF document

The LATEX coding below shows part of what was used to produce the tagging of mathematics shown in Figure 2. It shows how to use new primitive commands `\pdfstructelem`, `\pdfstartmarkedcontent` and `\pdfendmarkedcontent`.

```
\pdfstructelem attr{/S /Formula} 3 27
\pdfstructelem attr{/S /math} 27 28
\pdfstructelem attr{/S /mrow} 28 29
 \pdfstructelem attr{/S /msup} 29 30
  \pdfstructelem attr{/S /mrow} 30 31
   \pdfstructelem attr{/S /mo} 31 32
    \pdfstartmarkedcontent attr{/ActualText(\050)
      /Alt(, open bracket, )} 32 {mo}\biggl( \pdfendmarkedcontent
   \pdfstructelem attr{/S /mi} 31 33
    \pdfstartmarkedcontent attr{/Alt( alpha )}noendtext
       33 {mi}\alpha \pdfendmarkedcontent
   \pdfstructelem attr{/S /mo} 31 34
    \pdfstartmarkedcontent attr{/Alt( plus )}noendtext
       34 {mo}+ \pdfendmarkedcontent
    ...
```

The primitive `\pdfstructelem` requires two numbers specifying a unique identifier for the structure node being created, preceded by the identifier of its parent node, and attributes including the type of tag. Leaf nodes, constructed with `\pdfstartmarkedcontent`, require the identifier of the parent structure node. Their attributes can include `/Alt` text to be read by a screen-reader, and an `/ActualText` alternative for text-extraction. The kind of node for a mathematical symbol agrees with its parent structure node, (e.g., `/mi`, `/mo` or `/mn`). This is followed by the TEX coding to produce a visual representation, terminated by `\pdfendmarkedcontent`. Part of the PDF content stream resulting from this coding is given below, showing how the tagging is interspersed with positioning and font-changing commands, and the font characters themselves.

```
1 0 0 1 70.69 -23.949 cm
/mo <</MCID 15  /ActualText(\050) /Alt(, open bracket, )>>BDC
1 0 0 1 0 17.036 cm BT
/F1 9.9626 Tf/F18 1 Tf( )Tj/F1 9.9626 Tf [(\040)]TJ ET EMC
1 0 0 1 7.887 -17.036 cm
/mi <</MCID 16  /Alt( alpha )>>BDC BT
/F11 9.9626 Tf/F18 1 Tf( )Tj/F11 9.9626 Tf [(\013)]TJ ET EMC
1 0 0 1 6.41 0 cm
/mo <</MCID 17  /Alt( plus )>>BDC
1 0 0 1 2.214 0 cm BT
/F8 9.9626 Tf/F18 1 Tf( )Tj/F8 9.9626 Tf [(+)]TJ ET EMC
```

This kind of coding, directly in pdfTEX primitives, is really only useful for testing and "proof of concept" examples, such as Figure 2. Any mistake in the numerical identifiers can result in a broken PDF that may appear to render properly, but nevertheless crashes Acrobat due to a malformed structure tree.

Handling those numerical identifiers and parent relationships is something better done using an extra layer of LATEX macros, as in the coding example below. A `\taginlinemath` macro sets up an enclosing `/Formula` structure tag. Presentation MathML structure is specified using `\tagmathbranch`. MathML content tags are associated with TEX source using `\tagmathbleaf`, which has

an optional argument for spoken text. A variant `\tagmathaleaf` accommodates `/ActualText` replacements for large delimiters and extended constructions which require more than one glyph to display a single symbol.

```
\taginlinemath{%
  \tagmathbranch{msup}{\storePDFparentID
   \tagmathbranch{mrow}{%
    \tagmathaleaf[, open bracket, ]{mo}{/stretchy /false
      /minsize(1.2em) /maxsize(1.2em)}{\050}{\bigl(}%
    \tagmathbleaf[ alpha ]{mi}{}{\alpha}%
    \tagmathbleaf[ plus ]{mo}{}{+}%
    \tagmathbleaf[ beta ]{mi}{}{\beta}%
    \tagmathaleaf[, close bracket, ]{mo}{/stretchy /false
       /minsize(1.2em) /maxsize(1.2em)}{\051}{\bigr)^{%
      \adjustendcontent \tagmathbleaf[ all squared, ]{mn}{}{2}%
     }%end of ^
    }% </mo>
   }% </mrow>
  }% </msup>
  ...
```

In the above examples, the MathML tagging has been coded by hand to get working LaTeX source. Ultimately such markup, that interweaves MathML tagging with TeX code, needs to be generated automatically. This will require new coding structures called from modified expansions for existing LaTeX internal commands and environments (as used with paragraphs, headings, etc.), as well as with mathematical environments. For math the proposed strategy is to write the LaTeX source of a complete environment to disk, run a 3rd-party MathML converter to generate the tagging, then read the result back into the running job, merging the two coded views of the same piece of mathematics. Any external MathML converter could be used, provided it can be run as a command-line program using TeX's `\write18` facility. Alternatively, if a MathML version is already available for a piece of LaTeX source, then this could be used instead.

References

1. Adobe Systems Inc.; PDF Reference 1.7, November 2006. http://www.adobe.com/devnet/pdf/pdf_reference.html
2. Hàn Thế Thành; Thesis — pdfTeX, published as: TUGboat, 21:4, (2000). http://www.tug.org/TUGboat/Contents/contents21-4.html
3. ISO 19005-1:2005; Document Management — Electronic document file format for long term preservation — Part 1: Use of PDF 1.4 (PDF/A-1). http://www.iso.org/iso/catalogue_detail?csnumber=38920
4. ISO/DIS 32000; Document management — Portable document format (PDF 1.7), July 2008. http://www.iso.org/iso/catalogue_detail?csnumber=51502
5. Moore, Ross R.; Advanced features for publishing mathematics, in PDF and on the Web. TUGboat, 29:3, (2008), pp. 464–473. http://www.tug.org/TUGboat/Contents/contents29-3.html
6. PDF/UA Universal Accessibility; websites at http://pdf.editme.com/pdfua and http://www.aiim.org/Standards/article.aspx?ID=27861.