(basic)

tagging

Contents

1 Introduction

In this document we describe one possible way to tag basic pdf files. With basic we mean simple, rather flat structure documents, like articles. This document is itself one such example. Our hope is that this simplified approach to tagging can be useful for the ConTEXt community, but we do not pretend that the tagging setup we suggest here will fit the more advanced structures often present in ConTEXt documents. On the other hand, you will hopefully see that you can set up your own mapping as you wish, so this document is at the same time a show case for a possible way to do that.

If you do not change the mappings in your file, most content will get mapped to `NonStruct`. The reason is that it is a somewhat safe tag, and even though it might not be the most informative tag to use, it will (very likely) lead to a document that validates. To use the mappings discussed in this document you need to enable tagging, for example by adding

```
\setupbackend[format=pdf/ua-2]
```

and then add

```
\setuptagging[state=start,preset=basic]
```

to get the specific setup.

Reading the various iso standards [ISO20; ISO23; ISO24] (we mainly include the references here to be able to test a basic bibliography setup in this document) is not always so easy, and sometimes wonder how the reasoning behind them were going. Their table(s) on permitted nesting of tags feels very arbitrary and not always logical. Some of them are claimed to be explained elsewhere than in the table (in another standard!), but even when reading the rules are not clear to us. Some of the possible tags do even seem less useful or extra complicated to handle, and we simply do not map to them.

Below we provide, section by section, examples and comments on the various types of content and tag we do have in mind for the so-called basic tagging. This serves the purpose of showing the reader on examples that we think work good enough. There is, however, no guarantee. If you start to nest these things or use some low-level hacking or so, you are on your own.

The rules we discuss here are defined in various files, depending on the type of mechanism. They are then collected (or linked to) in the file `lpdf-tag-imp-basic.lmt` This way, you as a user, can create your own file(s) and make your own mappings with your favorite tagging setup, in particular by using parts of the setups we provide here.

We will soon turn to the details, but let us mention that a few more details on reasoning and comments are available in Section 16. By the way, this version was validated by verapdf 1.29.90.

2 Some basics

The good news when it comes to tagging is that ConTEXt was built with structure in mind. It has for a very long time been possible to export to structured XML. Thus, for example, when one adds a section (using `\startsection` and `\stopsection`) one gets

```
<section>
  <sectioncaption>
    <sectionnumber>
```

```
      2
    </sectionnumber>
    <sectiontitle>
      Some basics
    </sectiontitle>
  </sectioncaption>
  <sectioncontent>
    ...
  </sectioncontent>
</section>
```

So, our problem has been to map each of these onto something that is safe and useful for the pdf accessibility standards. All our documents start with a `<document>` structure that is mapped to a `Document` structure type, and then at the next level sits `<documentpart>`, which we map to `Part`. These two has category `Document` and `Grouping`, respectively. Neither of them can contain content directly, so extra level(s) are needed. We only mention here that if those extra levels are not naturally present, we enforce (almost always) a `P` tag, that indeed can have content.

We can easily get an overview over this document by invoking

```
mtxrun --script pdf --check tagging-basic.pdf
```

That creates a file `tagging-basic-tagview.xml` with lots of data. If you have verapdf installed, it also runs a validation check on the file.

3 Headings

When we add a section we will get `<section>` mapped onto `Sect` and `<sectioncontent>` mapped onto `Div`. These are also in the `Grouping` category, and therefore do not directly contain content. We also get `<sectioncaption>` mapped onto some `Hn` and `<sectionnumber>` and `<sectiontitle>` mapped onto `Lbl`. Since we map `<sectioncontent>` to `Div`, we need some structure type below that can contain content. If we just type text, we get `P`. Thus, a single section can give rise to (here we indicate content in parentheses)

```
Document
  Part
    Sect
      H1
        Lbl (3)
        Lbl (Headings)
      Div
        P   (When we add...)
```

So far so good. When we start to nest sections and subsections and so on, we need to use tags that nest well. The `Div` can have `Sect`, so nesting sections should work.

We support a few levels of headings, and one is supposed to map them onto `H1`, `H2` and so on upto `H5`, see Table 1.

Heading	**Tag**
`part`	`H1`
`chapter, title`	`H2`
`section, subject`	`H3`
`subsection, subsubject`	`H4`
`subsubsection, subsubsubject`	`H5`

Table 1 Different heading mappings.

If you, as in this document, start at another level than `part`, it might be good to map your highest level to `H1`. In this document, where section is the highest level, we have done

```
\setuptagging[level=3]
```

which means that we map the third level onto `H1`. The interested reader can peek into `lpdf-tag-imp-basic-section.lmt`, to see how that was implemented.

4 Itemgroups

There are several types of lists available and they are all using the `itemgroup` mechanism. We add a bullet list with the code

```
\startitemize
  \startitem First item.  \stopitem
  \startitem Second item. \stopitem
\stopitemize
```

and the typeset output is shown below.

- First item.

- Second item.

This gets mapped into

```
L
  LI
    Lbl (•)
    LBody (First item.)
  LI
    Lbl (•)
    LBody (Second item.)
```

We can indeed use simple lists, bulleted or with numbers or characters. It is assumed that items are added with `\startitem` and `\stopitem`, not just `\item`. It also works to nest lists.

1. Here is some text in the numbered list. This item also contains a second list.

    - First item in the nested list.

    - Second item in the nested list.

    - Third item in the nested list.

2. Here is some more text in the second item of the numbered list.

5 Floats

We mention the two most common type of float elements, tables and figures, but in principle other floats will work in the same way.

5.1 Tables

We have already used a table in this document, see Table 1. That table was entered with

```
\startplacetable
  [title=Different heading mappings.,
   reference=table:headings]
  \starttabulatehead
    \FL
    \NC {\bf Heading} \NC {\bf Tag} \NC \NR
    \ML
  \stoptabulatehead
  \starttabulate[|T|T|]
    \NC part                          \NC H1 \NC \NR
    \NC chapter, title                \NC H2 \NC \NR
    \NC section, subject              \NC H3 \NC \NR
    \NC subsection, subsubject        \NC H4 \NC \NR
    \NC subsubsection, subsubsubject \NC H5 \NC \NR
    \LL
  \stoptabulate
\stopplacetable
```

We do not show all details here, but we typically get

```
Aside
  Div
    Table
      TR
        TD (Heading)
        TD (Tag)
      ...
  Div
    Lbl (Table)
    Lbl (1)
    P   (Different heading mappings.)
```

and so on. The `Aside` and the two `Div` come from the floating mechanism while the `Table`, `TD` and `TR` come from the tabulate.

5.2 Graphics

With the code

```
\startplacefigure
  [title={This is the caption of the figure.},
   reference=figure:cow]
  \externalfigure
    [cow]
    [width=5cm,
     alternativetext=A Dutch cow!]
\stopplacefigure
```

we get a floating element as in Figure 1.

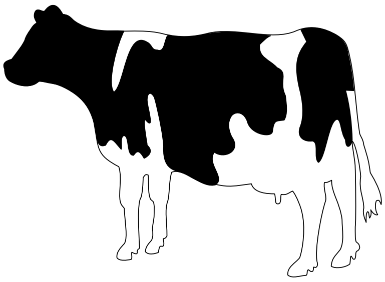

Figure 1 This is the caption of the figure.

The `alternativetext` adds a `Alt` tag on the graphic (that is tagged as a `Figure`). This alternative text will be taken verbatim, so no math or other stuff should go there. The structure in the pdf file is similar to the one for the table, but `Table` is instead `Figure`, so we leave it out.

MetaPost content is also working. It uses its own tags, but we can again add an alternative text, this time with `alternativetext`.

```
\startplacefigure
  [title=A MP figure]
  \startMPcode
    [alternativetext=A circle]
    draw fullcircle
      scaled 3cm
      withpen pencircle scaled 3
      withcolor darkred ;
  \stopMPcode
\stopplacefigure
```

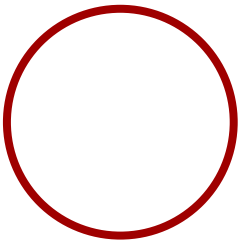

Figure 2 A MP figure

6 Descriptions

Mathematical theorems and similar environments use the description mechanism.

We define an instance by

```
\defineenumeration
  [theorem]
  [text=Theorem,
   alternative=serried,
   title=yes,
   width=fit]

\starttheorem
  [title=Pythagoras]
  In a right triangle, the square of the hypotenuse
  is equal to the sum of the squares of the legs.
\stoptheorem
```

Theorem 1 (Pythagoras) In a right triangle, the square of the hypotenuse is equal to the sum of the squares of the legs.

The structure we get is

```
Sect
  Lbl (Theorem 1 (Pythagoras))
  Div
```

```
    P (In a right ...)
```

As we have explained earlier, inside the `Div` we need a tag that have content, and in this case the `P` is enforced. This happens automatically when `Div` is the endpoint. When we generate the structure XML file, these enforced paragraphs are seen as `<p>`.

7 Notes

There are several types of notes possible, footnotes[1] is a probably the most common one. They do in fact also use the description structure. In `lpdf-tag-imp-basic-description.lmt` we give an example on how to remap to `FENote`. If you have any problems with footnotes, it might be a good idea to use endnotes instead, since then they are not done as inserts, and probably less fragile.

8 Code

We have already shown code several times. For inline code `like this`, done with `\typ`, `\type`, or similar, we use a `Code` tag.

When typesetting a block of code with `\starttyping` and `\stoptyping`, like

```
Here is some code
Here is more code
```

we do get

```
Div
  Code
    Sub (Here is some code)
    Sub (Here is more code)
```

9 Quotations

We can use `\quotation` which results in “quotation”, or `\quote` which results in ‘quote’. We get here `NonStruct`. That is perhaps

---

[1] Like this one.

not optimal, but the reason is that the `Quote` tag that is probably meant for this is quite limited.

Similarly, for block quotes, we can type

```
\startquotation
  This quote contains two paragraphs.

  You just read the first one, this is the second one.
\stopquotation
```

which typesets as

> “This quote contains two paragraphs.
>
> You just read the first one, this is the second one.”

Here we use `BlockQuote`, that seems to work better than `Quote`. The NVDA screen reader did not read the quotes, neither for the inline nor the displayed versions. It did not matter if we tagged them as `Artifact` or `Lbl`.

10 Contents

We felt that the `TOC` and `TOCI` tags are difficult to handle. Some bogus reference type element are needed and it quickly gets messy. We therefore ended up mapping the table of contents onto an ordinary list. Thus, the structure we get is

```
L
  LI
    Lbl   (1)
    LBody (Introduction)
    Lbl   (2)
```

11 Languages

Language switches are in principle handled. On the todo is to add a `language=...` for more mechanisms in ConTEXt.

12 Mathematics

We tag inline formulas such as $1 + 2 = 3$ (this was typed `\m{1 + 2 = 3}`). In the pdf file we find back this:

```
<!-- 1 plus 2 equals 3 -->
<math>
  <mrow>
    <mn>1</mn>
    <mo>+</mo>
    <mn>2</mn>
    <mo>=</mo>
    <mn>3</mn>
  </mrow>
</math>
```

The comment just shows what goes into the `Alt` tag and the `<math>` is put into a `Formula` tag.

For displayed formulas, we type as usual

```
\startformula
  a^2 + b^2 = c^2
  \numberhere[equation:Pythagoras]
\stopformula
```

to get the typeset results

$$a^2 + b^2 = c^2 \tag{1}$$

In the pdf file, this gets the structure

```
Div
  Formula
  Artifact (()
    Lbl (1)
  Artifact ())
```

To Hans: Can we disable the Alt for formulas and just rely on the XML somehow? If so, it should be mentioned here.

13 Margin material

We can put simple stuff in the margin, as can be seen in the example.

In the pdf file, this gets the `Aside` structure, which can contain content, so it seems we are fine.

This is just a small margin note.

We add some math $|a| = \sqrt{a^2}$ in a new paragraph.

14 User elements

It is possible for the user to define their own tagging elements. This can better be done in one of the `lmt` files. We have given one example in the `lpdf-tag-imp-basic-whatever.lmt` file, where we point `whatever` onto the `P` structure element. Thus, with

```
\startelement[whatever]
  This is a test using the whatever element.
\stopelement
```

This is a test using the whatever element.

and this is indeed mapped to `P`.

15 Bibliography

We place the bibliography in this document with `\placelistofpublications`. There is some kind of tagging support for bibliography, but we consider it just as a list, as we do for the table of contents. The bibliography list items are put into the `LI` tag, and hence mapped to `Lbl`. The rest of the publication fields are put into the `LBody` tag. They go under the ConTEXt types `publication` and `pubfld` tags, both mapped to `NonStruct`, so they inherit the `LBody` properties. Enough talking. Here comes the bibliography itself:

[ISO20] ISO, *Document management — Portable document format — Part 2: PDF 2.0*, International Standard, no. ISO 32000-2:2020 (Geneva, Switzerland, 2020).

[ISO23] ISO, *Document management — Portable Document Format — PDF 1.7 and 2.0 structure namespace inclusion in ISO 32000-2*, Technical Specification, no. ISO/TS 32005:2023 (Geneva, Switzerland, 2023).

[ISO24] ISO, *Document management applications — Electronic document file format enhancement for accessibility — Part 2: Use of ISO 32000-2 (PDF/UA-2)*, International Standard, no. ISO 14289-2:2024 (Geneva, Switzerland, 2024).

16 Some final words

If you decide (or are forced) to use tagging, then you should also be aware that it adds overhead. For a book we tested on, runtime doubled, and the size of the file was also much larger. This will quickly interrupt the work flow. One way to avoid this overhead while working is to add the tagging setups in a mode. So, perhaps do something like

```
\startmode[tagging]
\setupbackend
  [format=pdf/ua-2]

\setuptagging
  [state=start,
   preset=basic]
\stopmode
```

Then tagging is by default off, and you need to do

```
context --mode=tagging file.tex
```

if you want it enabled.

If you have interaction enabled in your document, as we have in this one, you will by default get validation errors. The reason is that the standard requires a few `Link` and `Reference` tags added here and there, always pointing to “real content”. If you want those elements added add

```
\setuptagging
  [option=interaction]
```

Remember, however, that if that breaks validation, you are on your own, and the best choice is probably to disable interaction. Or to live with those validation errors specific for the links. In one of our test documents the table of contents was added inside a MetaPost graphic. There the solution was to move it into an overlay.