Tagged PDF

Introduction

Occasionally users asked me if ConTEXt can produce tagged pdf and the answer to that has been: I'll implement it when I need it. However, users tell me that publishers show an increasing demand for tagged pdf files, although one might wonder what for, except maybe for accessibility. Another reason for not having spent too much time on it before, is that the specification was not that inviting.

At any rate, when I saw Ross Moore[1] presenting tagged math at TUG 2010, I decided to look up the spec once more and see if I could get into the mood to implement tagging. Before I started it was already clear that there were a couple of boundary conditions:

- Tagging should not put a burden on the user but users should be able to do the tagging themselves.
- Tagging should not slow down a run too much; this is no big deal as one can postpone tagging till the last run.
- Tagging should in no way interfere with typesetting, so no funny nodes should be injected.
- Tagging should not make the code look worse, neither the document source, nor the low level ConTEXt code.

And of course implementing it should not take more than a few days' work, certainly not during an exceptionally hot summer.

You can 'google' for one of Ross's documents (like `DML_002-2009-1_12.pdf`) to see how a document source looks at his end using a special version of pdfTEX. However, the version on my machine didn't support the primitives shown, so I could not see what was happening under the hood. Unfortunately it is quite hard to find a properly tagged document so we have only the reference manual as starting point. As the pdfTEX approach didn't look that pleasing anyway, I just started from scratch.

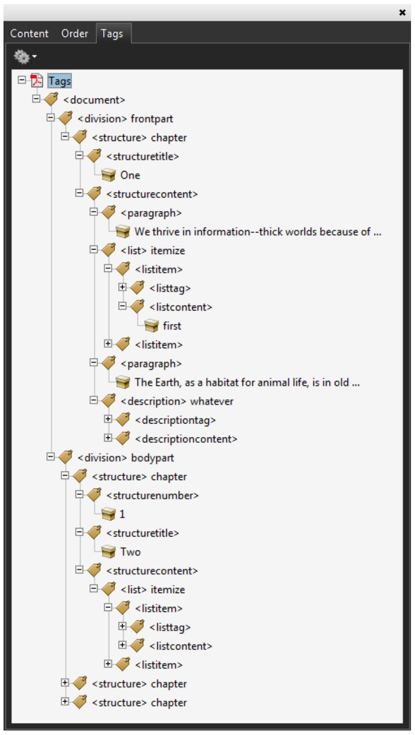
Tags can help Acrobat Reader when reading out the text aloud. But you cannot browse the structure in the no-cost version of Acrobat and as not all users have the professional version of Acrobat, the fact that a document has structure can go unnoticed. Add to that the fact that the overhead in terms of bytes is quite large as many more objects are generated, and you will understand why this feature is not enabled by default.

Implementation

So, what does tagging boil down to? We can best look at how tagged information is shown in Acrobat. Figure 1 shows the content tree that has been added (automatically) to a document while figure 2 shows a different view.

In order to get that far, we have to do the following:

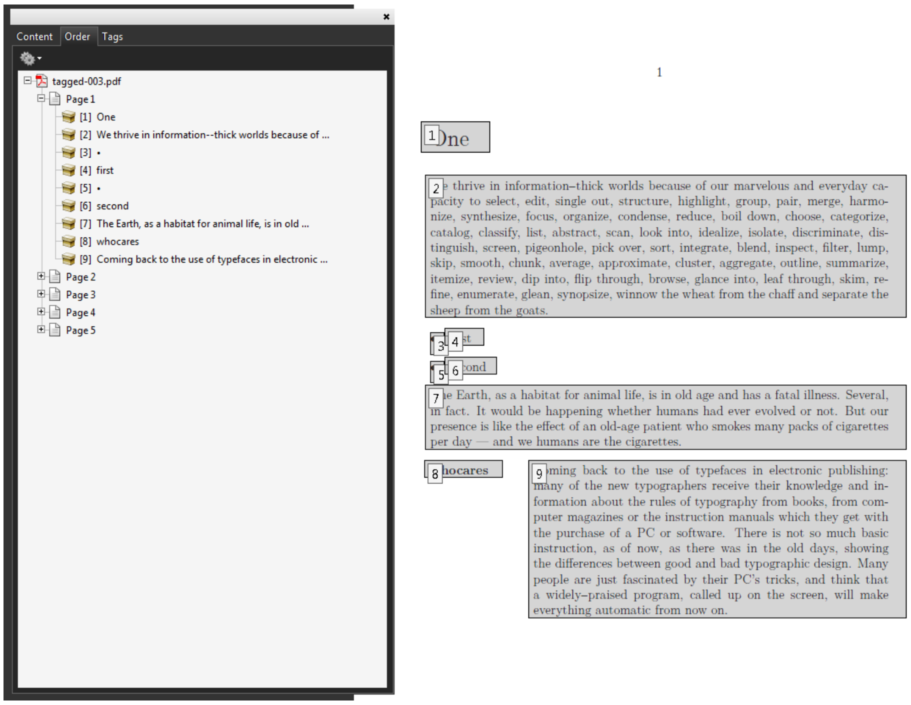
- Carry information with (typeset) text.
- Analyse this information when shipping out pages.



Content Order Tags

```
Tags
  <document>
    <division> frontpart
      <structure> chapter
        <structuretitle>
          One
        <structurecontent>
          <paragraph>
            We thrive in information--thick worlds because of ...
          <list> itemize
            <listitem>
              <listtag>
              <listcontent>
                first
            <listitem>
          <paragraph>
            The Earth, as a habitat for animal life, is in old ...
          <description> whatever
            <descriptiontag>
            <descriptioncontent>
    <division> bodypart
      <structure> chapter
        <structurenumber>
          1
        <structuretitle>
          Two
        <structurecontent>
          <list> itemize
            <listitem>
              <listtag>
              <listcontent>
            <listitem>
      <structure> chapter
      <structure> chapter
```

Figure 1. A tag list in Acrobat.



Content Order Tags

```
tagged-003.pdf
  Page 1
    [1] One
    [2] We thrive in information--thick worlds because of ...
    [3] •
    [4] first
    [5] •
    [6] second
    [7] The Earth, as a habitat for animal life, is in old ...
    [8] whocares
    [9] Coming back to the use of typefaces in electronic ...
  Page 2
  Page 3
  Page 4
  Page 5
```

1

1 One

2 e thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

3 4 st

5 6 cond

7 he Earth, as a habitat for animal life, is in old age and has a fatal illness. Several, in fact. It would be happening whether humans had ever evolved or not. But our presence is like the effect of an old-age patient who smokes many packs of cigarettes per day — and we humans are the cigarettes.

8 hocares

9 oming back to the use of typefaces in electronic publishing: many of the new typographers receive their knowledge and information about the rules of typography from books, from computer magazines or the instruction manuals which they get with the purchase of a PC or software. There is not so much basic instruction, as of now, as there was in the old days, showing the differences between good and bad typographic design. Many people are just fascinated by their PC's tricks, and think that a widely–praised program, called up on the screen, will make everything automatic from now on.

Figure 2. Acrobat showing the tag order.

- ▫ Add a structure tree to the page.
- ▫ Add relevant information to the document.

That first activity is rather independent of the other three and we can use that information for other purposes as well, like identifying where we are in the document. We carry the information around using attributes. The last three activities took a bit of experimenting mostly using the “Example of Logical Structure” from the pdf standard 32000-1:2008.

This resulted in a tagging framework that uses explicit tags, meaning the user is responsible for the tagging:

```
\setupstructure[state=start,method=none]

\starttext

\startelement[document]

    \startelement[chapter]
        \startelement[p] \input davis \stopelement \par
    \stopelement

    \startelement[chapter]
        \startelement[p] \input zapf \stopelement \par
        \startelement[whatever]
            \startelement[p] \input tufte \stopelement \par
            \startelement[p] \input knuth \stopelement \par
        \stopelement
    \stopelement

    \startelement[chapter]
        oeps
        \startelement[p] \input ward \stopelement \par
    \stopelement

\stopelement

\stoptext
```

Since this is not much fun, we also provide an automated variant. In the previous example we explicitly turned off automated tagging by setting `method` to `none`. By default it has the value `auto`.

```
\setupstructure[state=start] % default is method=auto

\definedescription[whatever]

\starttext

\startfrontmatter
    \startchapter[title=One]
        \startparagraph \input tufte \stopparagraph
        \startitemize
            \startitem first \stopitem
            \startitem second \stopitem
        \stopitemize
        \startparagraph \input ward \stopparagraph
        \startwhatever {Herman Zapf} \input zapf \stopwhatever
    \stopchapter

\stopfrontmatter
```

```
\startbodymatter
    ..................
```

If you use commands like `\chapter` you will not get the desired results. Of course these can be supported but there is no real reason for it, as in MkIV we advise using the `start-stop` variant.

It will be clear that this kind of automated tagging brings with it a couple of extra commands deep down in ConTEXt and there (of course) we use symbolic names for tags, so that one can overload the built-in mapping.

```
\setuptaglabeltext[en][document=text]
```

As with other features inspired by viewer functionality, the implementation of tagging is independent of the backend. For instance, we can tag a document and access the tagging information at the TEX end. The backend driver code maps tags to relevant pdf constructs. First of all, we just map the tags used at the ConTEXt end onto themselves. But, as validators expect certain names, we use the pdf rolemap feature to map them to (less interesting) names. The next list shows the currently used internal names, with the pdf ones between parentheses.

construct (Span), delimited (Quote), delimitedblock (BlockQuote), description (Div), descriptioncontent (Div), descriptionsymbol (Span), descriptiontag (Div), division (Div), document (Div), float (Div), floatcaption (Caption), floatcontent (P), floattag (Span), floattext (Span), formula (Div), formulacontent (P), formulaset (Div), formulatag (Span), image (P), item (Li), itemcontent (LBody), itemgroup (L), itemtag (Lbl), link (Link), list (TOC), listcontent (P), listdata (P), listitem (TOCI), listpage (Reference), listtag (Lbl), margintext (Span), margintextblock (Span), math (Div), merror (Span), mfrac (Span), mi (Span), mn (Span), mo (Span), mover (Span), mpgraphic (P), mroot (Span), mrow (Span), ms (Span), msqrt (Span), msub (Span), msubsup (Span), msup (Span), mtext (Span), munder (Span), munderover (Span), paragraph (P), register (Div), registerentries (Div), registerentry (Span), registerpage (Span), registerpagerange (Span), registerpages (Span), registersection (Div), registersee (Span), registertag (Span), section (Sect), sectioncontent (Div), sectionnumber (H), sectiontitle (H), sort (Span), subformula (Div), subsentence (Span), synonym (Span), table (Table), tablecell (TD), tablerow (TR), tabulate (Table), tabulatecell (TD), tabulaterow (TR), verbatim (Code), verbatimblock (Code), verbatimline (Code), verbatimlines (Code).

So, the internal ones show up in the tag trees as shown in the examples but applications might use the rolemap which normally has less detail.

Since we keep track of where we are, we can also use that information for making decisions.

```
\doifinelementelse{structure:section}           {yes} {no}
\doifinelementelse{structure:chapter}           {yes} {no}
\doifinelementelse{division:*-structure:chapter} {yes} {no}
\doifinelementelse{division:*-structure:*}       {yes} {no}
```

As shown, you can use `*` as a wildcard. The elements are separated by `-`. If you don't know what tags are used, you can always enable the tag related tracker:

```
\enabletrackers[structure.tags]
```

This tracker reports the identified element chains to the console and log.

Special care

Of course there are a few complications. First of all the tagging model sort of contradicts the concept of a nicely typeset document where structure and outcome are not always related. Most TEX users are aware of the fact that TEX does not have space characters and does a great job on kerning and hyphenation. The tagging machinery on the other hand uses a rather dumb model of strings separated by spaces.[2] But we can trick TEX into providing the right information to the backend so that words get nicely separated. The non-optimized function that does this looks as follows:

```
function injectspaces(head)
    local p
    for n in node.traverse(head) do
        local id = n.id
        if id == node.id("glue") then
            if p and p.id == node.id("glyph") then
                local g = node.copy(p)
                local s = node.copy(n.spec)
                g.char, n.spec = 32, s
                p.next, g.prev = g, p
                g.next, n.prev = n, g
                s.width = s.width - g.width
            end
        elseif id == node.id("hlist") or id == node.id("vlist") then
            injectspaces(n.list,attribute)
        end
        p = n
    end
end
```

Here we squeeze in a space (given that it is in the font which it normally is when you use ConTEXt) and make a compensation in the glue. Given that your page sits in box 255, you can do this just before shipping the page out:

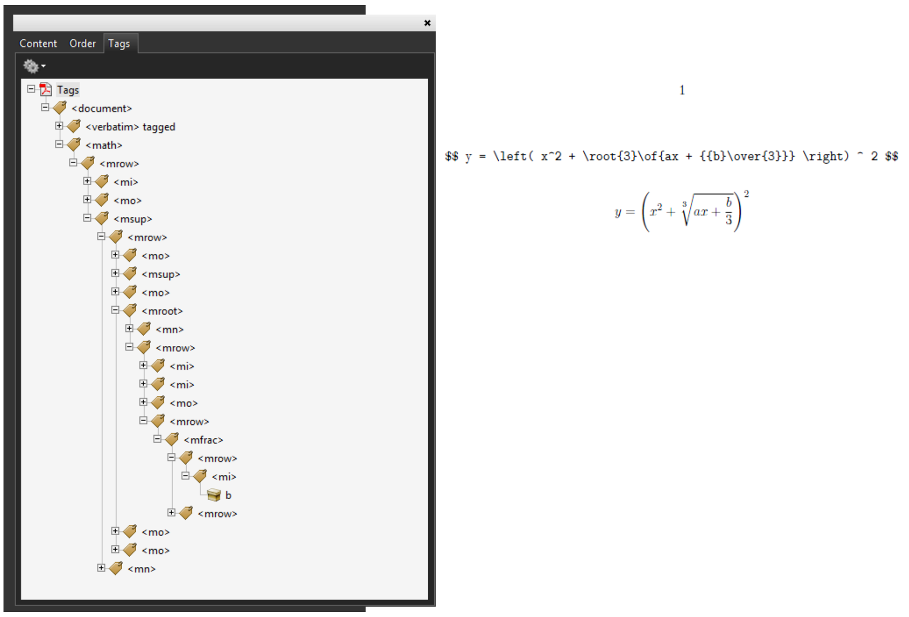
```
injectspaces(tex.box[255].list)
```

Then there are the so-called suspects: things on the page that are not related to structure at all. One is supposed to tag these in a special way to prevent the built-in reading equipment from getting confused. So far we could get around them simply because they don't get tagged at all and therefore are not seen anyway. This might well be enough of a precaution.

Of course we need to deal with mathematics. Fortunately the presentation MathML model is rather close to TEX and so we can map onto that. After all we don't need to care too much about back-mapping here. The currently present code is rather experimental and might get extended or thrown out in favour of inline MathML. Figure 3 demonstrates that a first approach does not even look that bad. In future versions we might deal with table-like math constructs, like matrices.

This is a typical case where more energy has to be spent on driving the voice of Acrobat but I will do that when we find a good reason.

As mentioned, it will take a while before all relevant constructs in ConTEXt support tagging, but support is already quite complete. Some screen dumps are included as examples at the end.



```
Tags
  <document>
    <verbatim> tagged
    <math>
      <mrow>
        <mi>
        <mo>
        <msup>
          <mrow>
            <mo>
            <msup>
            <mo>
            <mroot>
              <mn>
              <mrow>
                <mi>
                <mi>
                <mo>
                <mrow>
                  <mfrac>
                    <mrow>
                      <mi>
                        b
                    <mrow>
            <mo>
            <mo>
          <mn>
```

1

```
$$ y = \left( x^2 + \root{3}\of{ax + {{b}\over{3}}} \right) ^ 2 $$
```

$$y = \left(x^2 + \sqrt[3]{ax + \frac{b}{3}} \right)^2$$

Figure 3. Experimental math tagging.

Conclusion

Surprisingly, implementing all this didn't take that much work. Of course detailed automated structure support from the complete ConTEXt kernel will take some time to get completed, but that will be done on demand and when we run into missing bits and pieces. It's still not decided to what extent alternate representations and alternate texts will be supported. Experiments with the reading-aloud machinery are not satisfying yet but maybe it just can't get any better. It would be nice if we could get some tags being announced without overloading the content, that is: without using ugly hacks.

And of course, code like this is never really finished if only because pdf evolves. Also, it is yet another nice test case and torture test for LuaTEX and it helps us to find buglets and oversights.

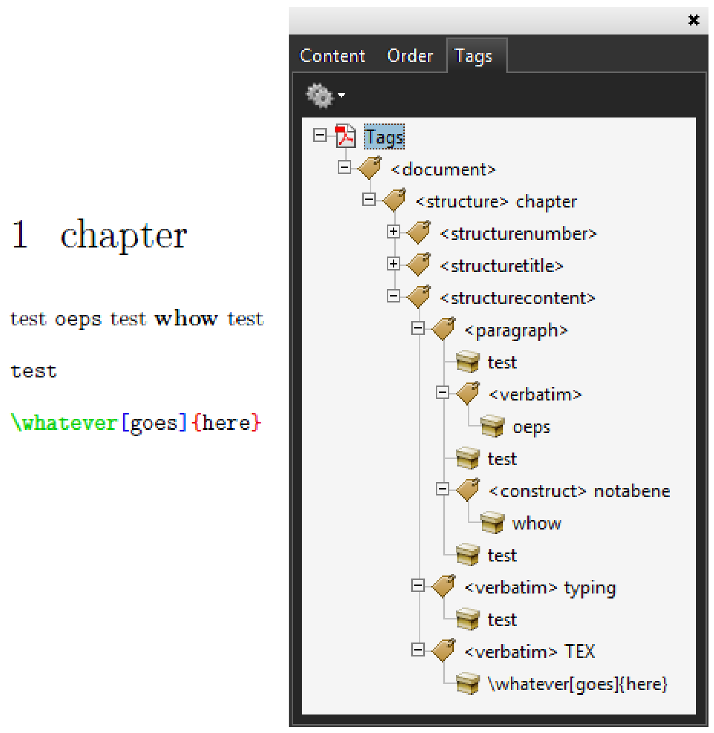
Some more examples

In ConTEXt we have user definable verbatim environments. As with other user definable environments we show the specific instance as comment next to the structure component. See figure 4. Some examples of tables are shown in figure 5. Future versions will have a bit more structure. Tables of contents (see figure 6) and registers (see figure 7) are also tagged. (One might wonder how useful this is.) In figure 8 we see some examples of floats. External images as well as MetaPost graphics are tagged as such. This example also shows an example of a user environment, in this case:

```
\definestartstop[notabene][style=\bf]
```

In a similar fashion, footnotes (figure 9) end up in the structure tree, but in the typeset document they move around (normally forward when there is no room).

Hans Hagen
Pragma ADE, Hasselt, The Netherlands
pragma@wxs.nl



1 chapter

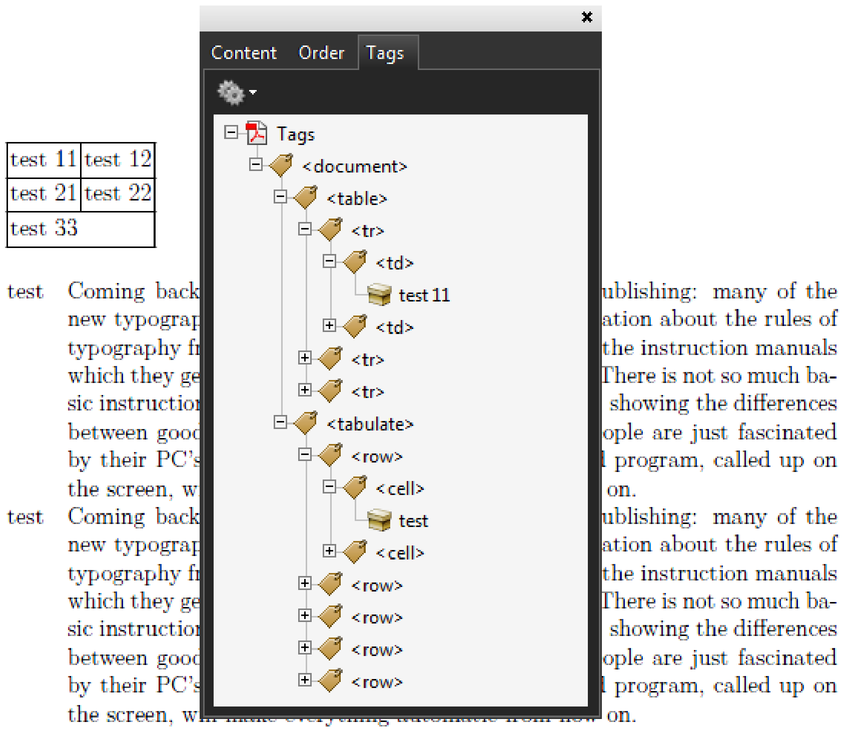
test oeps test **whow** test

test

```
\whatever[goes]{here}
```

```
⊟ Tags
  ⊟ <document>
    ⊟ <structure> chapter
      ⊞ <structurenumber>
      ⊞ <structuretitle>
      ⊟ <structurecontent>
        ⊟ <paragraph>
            test
          ⊟ <verbatim>
              oeps
            test
          ⊟ <construct> notabene
              whow
            test
        ⊟ <verbatim> typing
            test
        ⊟ <verbatim> TEX
            \whatever[goes]{here}
```

Figure 4. Verbatim, including dedicated instances.



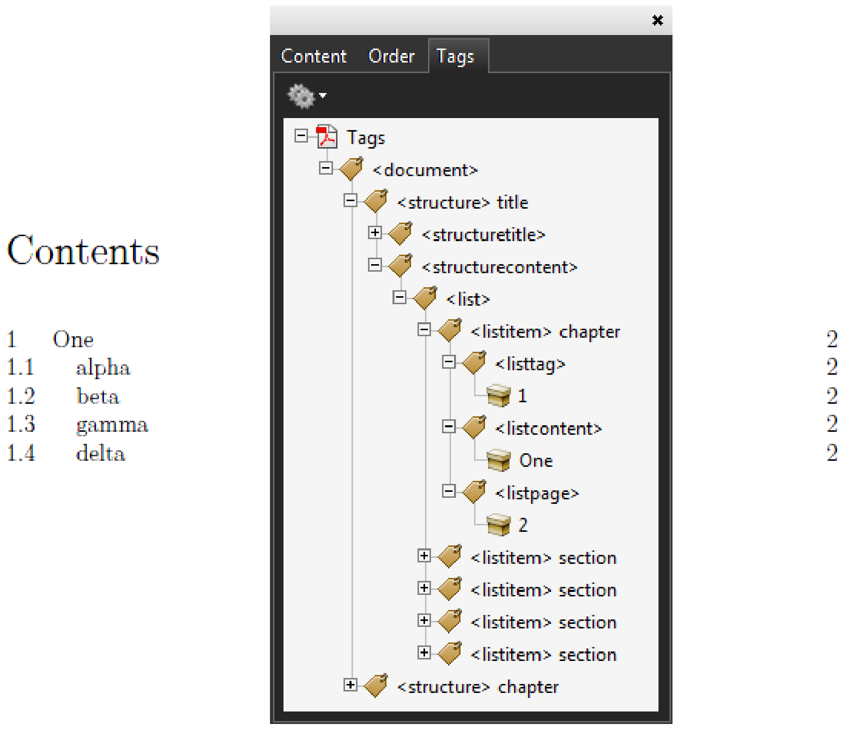
test 11	test 12
test 21	test 22
test 33	

test Coming back … ublishing: many of the new typograp… ation about the rules of typography fr… the instruction manuals which they ge… There is not so much basic instructio… showing the differences between good… ople are just fascinated by their PC's… program, called up on the screen, w… on.

test Coming back … ublishing: many of the new typograp… ation about the rules of typography fr… the instruction manuals which they ge… There is not so much basic instructio… showing the differences between good… ople are just fascinated by their PC's… program, called up on the screen, w… on.

```
⊟ Tags
  ⊟ <document>
    ⊟ <table>
      ⊟ <tr>
        ⊟ <td>
            test 11
        ⊞ <td>
      ⊞ <tr>
      ⊞ <tr>
    ⊟ <tabulate>
      ⊟ <row>
        ⊟ <cell>
            test
        ⊞ <cell>
      ⊞ <row>
      ⊞ <row>
      ⊞ <row>
      ⊞ <row>
```

Figure 5. Natural tables as well as the tabulate mechanism is supported.

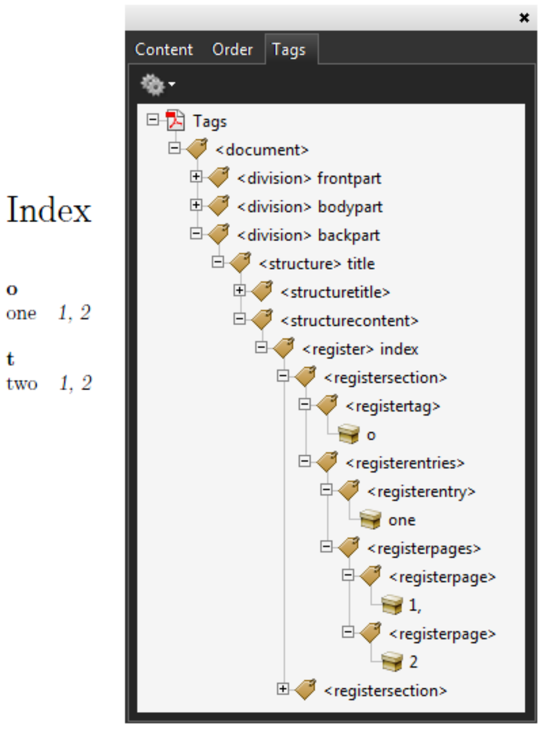


Contents

1	One	2
1.1	alpha	2
1.2	beta	2
1.3	gamma	2
1.4	delta	2

```
Content  Order  Tags
Tags
  <document>
    <structure> title
      <structuretitle>
      <structurecontent>
        <list>
          <listitem> chapter
            <listtag>
              1
            <listcontent>
              One
            <listpage>
              2
          <listitem> section
          <listitem> section
          <listitem> section
          <listitem> section
    <structure> chapter
```

Figure 6. Tables of content with specific entries tagged.



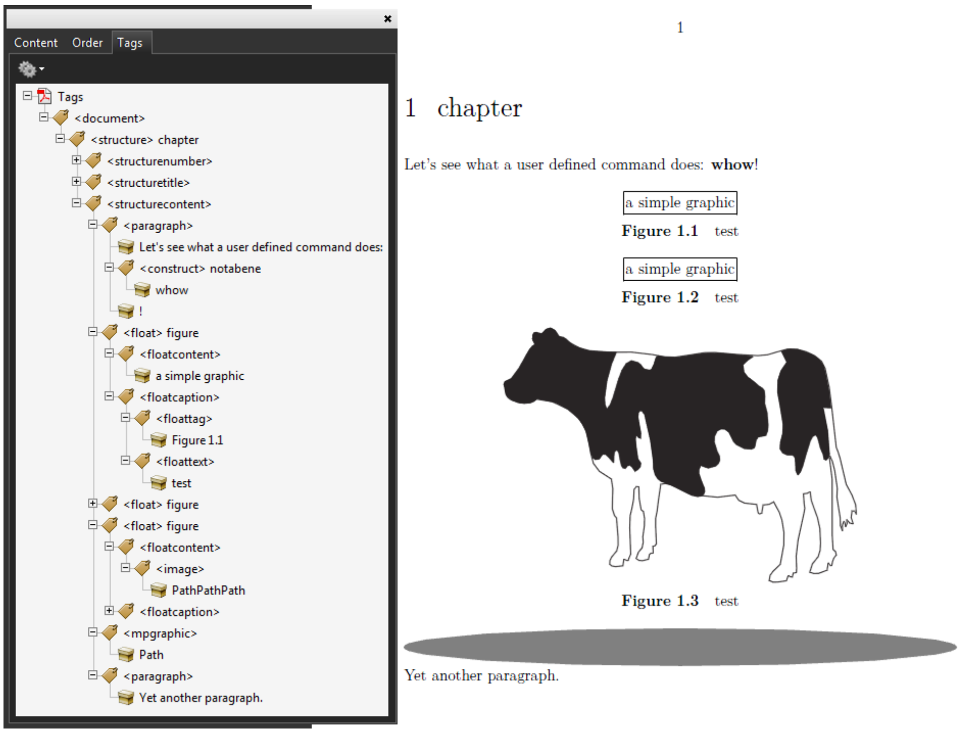
Index

o
one *1, 2*

t
two *1, 2*

```
Content  Order  Tags
Tags
  <document>
    <division> frontpart
    <division> bodypart
    <division> backpart
      <structure> title
        <structuretitle>
        <structurecontent>
          <register> index
            <registersection>
              <registertag>
                o
              <registerentries>
                <registerentry>
                  one
                <registerpages>
                  <registerpage>
                    1,
                  <registerpage>
                    2
            <registersection>
```

Figure 7. A detailed view of registered is provided.



```
Content   Order   Tags
Tags
  <document>
    <structure> chapter
      <structurenumber>
      <structuretitle>
      <structurecontent>
        <paragraph>
          Let's see what a user defined command does:
          <construct> notabene
            whow
          !
        <float> figure
          <floatcontent>
            a simple graphic
          <floatcaption>
            <floattag>
              Figure 1.1
            <floattext>
              test
        <float> figure
        <float> figure
          <floatcontent>
            <image>
              PathPathPath
          <floatcaption>
        <mpgraphic>
          Path
        <paragraph>
          Yet another paragraph.
```

1

1 chapter

Let's see what a user defined command does: **whow**!

a simple graphic

Figure 1.1 test

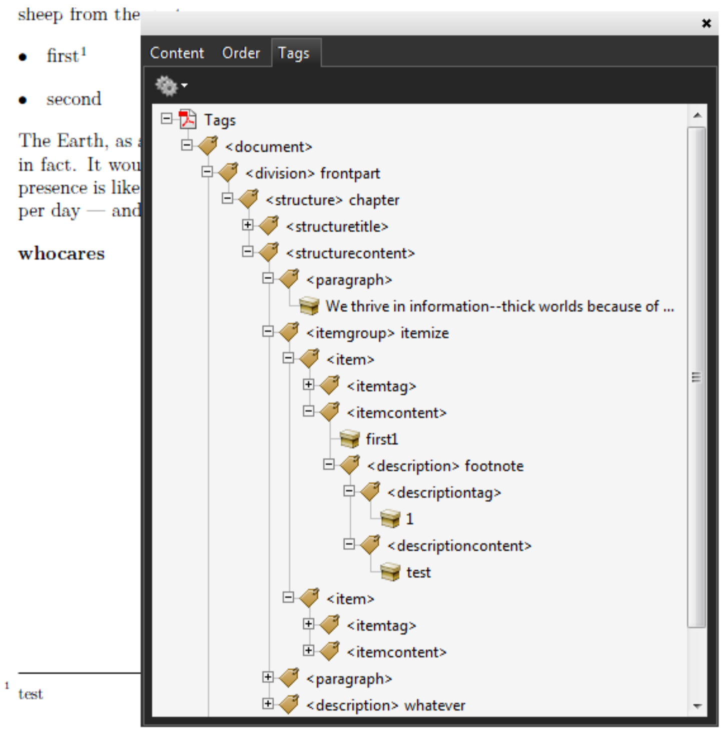
a simple graphic

Figure 1.2 test

Figure 1.3 test

Yet another paragraph.

Figure 8. Floats tags end up in text stream. Watch the user defined construct.



sheep from the

- first[1]
- second

The Earth, as
in fact. It wou
presence is like
per day — and

whocares

[1] test

```
Content   Order   Tags
Tags
  <document>
    <division> frontpart
      <structure> chapter
        <structuretitle>
        <structurecontent>
          <paragraph>
            We thrive in information--thick worlds because of ...
          <itemgroup> itemize
            <item>
              <itemtag>
              <itemcontent>
                first1
                <description> footnote
                  <descriptiontag>
                    1
                  <descriptioncontent>
                    test
            <item>
              <itemtag>
              <itemcontent>
          <paragraph>
          <description> whatever
```

Figure 9. Footnotes are shown at the place in the input (flow).