On the road to Tagged PDF: About StructElem, Marked Content, PDF/A and Squeezed Bärs

Ulrike Fischer

Abstract

In this article I present two packages as part of the LaTeX Project's "Tagged PDF" effort:

`tagpdf` which contains the core code to create a tagged PDF and is used by the LaTeX team to test new code.

`pdfmanagement-testphase` which contains a large number of PDF-related commands and tools and installs a new management command for central PDF dictionaries.

I will show how to use these packages and the benefits they will bring for the average user, while also mentioning resulting incompatibilities and required changes in documents.

1 Introduction

At its core, PDF is a page and graphic-related format: it allows describing, very accurately, a page layout, the font sizes, font types, colors, placement of characters and graphical elements, but essentially it simply says where to put ink on the page. It is possible to extract text but one doesn't get the semantic meaning of such a text, the reading order can be unclear, and — with PDF produced by TeX — it can even be unclear where words begin and end.

However, PDF is also quite an extensible format. In the same way you can enhance HTML with Cascading Style Sheets (CSS) and JavaScript, you can enhance PDF in various ways: you can embed files with source code or other material, you can add alternative text to objects. And, most importantly for our purposes, you can *tag* it: with this you are adding a structural layer over the content (figure 1).

2 Some technical details

In the PDF format, every page is stored in a *page stream.* This stream contains various operators which select fonts, move around on the page, and insert images and text.

As described in a previous article [1], tagging requires first the writing of markers called *Marked Content* (MC) markers into the page stream which label and number all chunks of meaningful content.

The next step is to add various objects to the PDF. These objects contain references to each other as parent and kid objects and so form a tree. The numbered *Marked Content* chunks are leaf nodes in this tree. Figure 2 shows a simple tree created

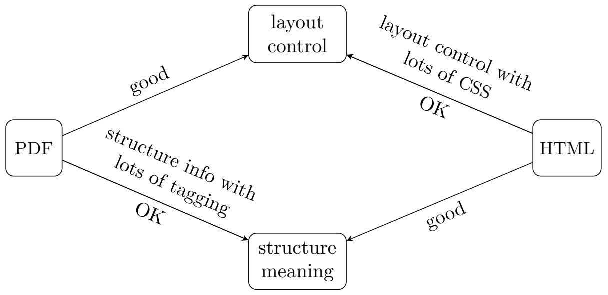

layout control
PDF
HTML
structure meaning
good
layout control with lots of CSS
OK
structure info with lots of tagging
OK
good

Figure 1: PDF vs. HTML, layout vs. structure

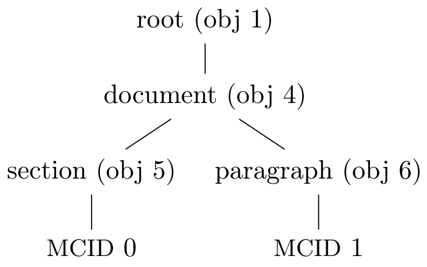


Figure 2: Simple structure tree

with such objects. The last step is to create various objects for cross-referencing and housekeeping, and to register them in the catalog so that a PDF viewer can find and use the structure.

3 Inspecting a tagged PDF

Checking the tags of a PDF can be done with Adobe Acrobat Pro; some free options are also slowly appearing. One of them is the online editor at `ngpdf.com`. It allows you to upload a tagged PDF, view the tag structure and view and also export an HTML representation of the PDF. If the math has an embedded associated file with the MathML representation, then the HTML will use MathJax to render it.

Unlike the copy & paste heuristic of PDF viewers, the site doesn't try to guess the reading order but shows you what you get from the tags and the structure tree. The HTML export demonstrates that tagging isn't only for screen readers. It gives you more options to *reuse* the PDF. The old saying that PDF is like scrambled eggs and that you can't get back an egg (a structured LaTeX document) from it is no longer true: You still don't get all the details back, but depending on the amount of tagging that's been added, you can restore quite a lot.

4 Amount of tagging: About (im-)perfection and standards

A document must be tagged in its entirety. You can't tag only a specific section or some table or some math to improve its export or copy & paste, even if the rest of the document already works satisfactorily.

But this still leaves you with quite a lot of freedom about the amount of tagging. If a paragraph contains some emphasized text you can tag it as an `Em`-structure, but you don't have to. This naturally means that some information will be lost, but quite apart from the fact that is often impossible anyway to mirror the intention of every detail of a layout into such a structure tree, tagging doesn't have to be perfect to be useful; it improves accessibility significantly even if merely headings are tagged for better navigation and paragraphs are shown in the correct reading order.

The same goes for standards: There exist various guides and standards which describe requirements for accessible and well-tagged PDF documents. They and the validators checking such standards are useful to set goals which we should aim to reach, but that it is often not (yet) possible to fulfill all aspects of such a standard should not stop you from creating more usable documents.

The end users of the documents we produce are not validators but intelligent humans, and intelligent humans can often handle imperfect documents surprisingly well, and in the end, their feedback is what will really matter.

5 Tagging with the `tagpdf` package

I wrote the `tagpdf` package (`ctan.org/pkg/tagpdf`) around four years ago. At that time I was already convinced that tagging at a reasonable scale can't be done in an external package, but that changes would be required in LaTeX itself. If we want a large number of documents to be tagged, "normal" tagging shouldn't need expert skills and fragile patches but should be supported by LaTeX directly.

So the goals of `tagpdf` were

- to develop the basic tagging code for the kernel,
- to provide examples and tests,
- to identify LaTeX problems, and
- to develop stable solutions which can be eventually integrated into LaTeX.

This means that `tagpdf` wasn't written as a standard user package, but rather primarily as a research and development tool and will eventually vanish again. As such

- it requires the newest LaTeX code, sometimes even LaTeX-dev,
- it can still change,
- it concentrates on basic commands, and
- it doesn't patch other packages.

Listing 1: Tagging commands

```
\tagstructbegin{tag=P}%
 \tagmcbegin{tag=P}...text...\tagmcend
 \tagstructbegin{tag=Link}
  \tagmcbegin{tag=Link}...text...\tagmcend
 \tagstructend
  \tagmcbegin{tag=P}...text...\tagmcend
  %pagebreak: new mc-chunk
  \tagmcbegin{tag=P}...text...\tagmcend
\tagstructend
```

`tagpdf` defines a number of commands, but the two core command pairs are these: First, the commands to mark the MC-chunks. As they split the text into small labeled chunks these commands should be used linearly: `\tagmcbegin ... \tagmcend`.

Secondly, there are the commands to mark the structure. They are typically nested: `\tagstructbegin ... \tagstructend`.

Listing 1 shows a simple example of their use. It is important to note that if there is a page break in the middle of a text chunk, the MC-marker must be closed before the break and reopened after the break. This is easy to do manually but rather challenging if it has to be done automatically, at least with other engines than LuaLaTeX. Frank Mittelbach's companion talk discusses this.

Listing 2 shows the source code of a small but complete document using `tagpdf`. In general, tagging works best with LuaLaTeX. To use other engines, you need the code mentioned in Frank's talk, or mark up all page breaks manually. Neither LaTeX & dvips nor XeLaTeX support real space characters.

There are a few important points to note here:

- The document should be compiled twice, sometimes more. Be aware that you don't necessarily get a rerun message yet, even if one is needed.
- `tagpdf` requires the `pdfmanagement-testphase` package. I will say more about this below.
- `tagpdf` is still a normal package and can be loaded with `\usepackage`, but as a first step towards full integration into LaTeX itself it can also be loaded with the `testphase` key.
- Tagging must be activated explicitly. This can be done as in the example with the `activate` key, and also with the `\tagpdfsetup` command from the package (see the documentation).
- The example shows lots of tagging commands around the section, but none at all around the paragraph or the link.

This last point reflects the state of the project: With the help of the work done over the last months

Listing 2: A full example

```
\RequirePackage{pdfmanagement-testphase}
\DeclareDocumentMetadata{pdfversion=2.0,
  testphase={tagpdf}, activate=tagging}
\documentclass{article}
\usepackage{hyperref}
\pagestyle{empty}
\begin{document}
\tagpdfparaOff
\tagstructbegin{tag=H1}\tagmcbegin{tag=H1}
\section{A section}
\tagmcend\tagstructend\tagpdfparaOn

A paragraph with some text and a link to
\url{https://www.latex-project.org}

Another paragraph \ldots
\end{document}
```

on the new paragraph hooks and PDF management, paragraphs and links can be automatically tagged, but sectioning commands still need either manual tagging or patches. It will be the next task to automate this too, which will require some months' work. You may wonder why it takes so long to insert what amounts to two lines of code. The first reason is that the code must be added to commands which are used, patched and redefined by many classes and packages. If we simply change them to support tagging, this would break these classes and packages. So we need a proper change strategy here. The second reason is that as we are obliged to change the sectioning commands, we want to use the opportunity to modernize and improve them.

6 PDF management

Tagging writes many objects and other code into the PDF. This requires proper tools. "Proper tools" means, above all, abstracted, backend-independent tools provided directly by LaTeX. When I started with `tagpdf` there was nearly nothing available. You either had to use some external package or a primitive command. By now the situation has changed dramatically, and for the better: The LaTeX format now includes the L3 programming layer (`expl3`) code and loads backend files from `l3backend`. This means that even in small documents, commands to write PDF objects, to set the PDF version, and to use colors are available.

The new PDF management support, currently in the external package `pdfmanagement-testphase`, extends this set of tools. It offers commands to write the MC-markers, to embed files (needed to add associated files to a structure), to create links which have suitable hooks for the automatic tagging, commands to create form fields with built-in tagging support, and more.

But the PDF management does more: In a PDF there are a number of central dictionaries to which packages shouldn't write directly, to avoid clashes with other packages. Until now, packages had no way to avoid such problems apart from testing for potentially problematic packages, but the new PDF management will resolve this problem: It offers a command which allows to write to the dictionaries in a managed way. The command exists in two versions, for LaTeX 2_ε code and for `expl3` code: `\PDFManagementAdd` and `\pdfmanagement_add:nnn`, respectively, which together we'll call `pdfmanagement` for short. Details on how to use the command can be found in the documentation `l3pdfmanagement`.

6.1 Incompatibilities

This new `pdfmanagement` support replaces the five primitives `\pdfinfo`, `\pdfcatalog`, `\pdfpageattr`, `\pdfpagesattr` and `\pdfpageresources` and the analogous commands of the other engines and backends. *Replace* truly means *replace* here: every package which uses those primitives is incompatible with the new `pdfmanagement`. For this reason we built a safety net around its use:

- The code has not been added directly to the kernel, but is offered now as an external testphase package.
- For the central management command, there is an explicit activation trigger command: `\DeclareDocumentMetadata`.
- We already wrote a number of replacements for incompatible packages and drivers, for example `hyperref`, `transparent`, `pdflscape`.
- We wrote a number of "firstaid" patches for packages which will stay until the packages adapt their code, e.g., `pgf` and `hyperxmp`.
- We notified various package and class maintainers and asked them to adjust their code.

But we can see problems only with packages that are on CTAN and in the TeX distributions. For the various house classes and packages of universities and journals we need the help of users to report problems. *So please test and check!*

6.2 Benefits

The new `pdfmanagement` is not only good for tagging. The rewriting and checking of PDF related code and packages was also used to modernize, correct and extend various other features. I would like to mention a few improvements.

Listing 3: New hyperref options

```
\hypersetup{href/protocol=https://,
            href/urlencode}
\hrefurl{www.köln.de}{Köln}
\url[format=\textsc]{www.köln.de}
```

Listing 4: Rotating of float pages

```
\begin{figure}[p]
\PDFManagementAdd{ThisPage}{Rotate}{90}
... landscape picture ...
\end{figure}
```

6.2.1 Colors and link options for hyperref

When the new pdfmanagement is used, hyperref (ctan.org/pkg/hyperref) uses a new, generic driver. This driver uses better default colors, has keys to change to other color schemes, and styles like linkcolors can now be changed at any time in the document. The support for non-ASCII links has been extended: links can be input in UTF-8 and hyperref will convert them internally into the needed “percent” encoding. It is possible to preset a protocol, most likely `https://`, which is then prefixed to all links. There is also a hook system which allows packages to add support to differentiate internal links by concepts such as citation, acronym, glossary and similar. See listing 3.

6.2.2 Rotations of float pages

With the pdfmanagement command it is possible to rotate float pages; depending on the engine this requires one or two compilations. See listing 4.

6.2.3 Support for PDF standards

The pdfmanagement bundle also contains a module for PDF standards. Currently it mostly handles A-standards. When such a standard is requested the code will, for example, include a color profile and register it in the PDF catalog. It will also enable verification tests that other packages can use in their code; for example, the new hyperref driver uses this to suppress JavaScript actions. This replaces the key pdfa which is no longer used by the new driver. XMP-metadata can be added by loading the hyperxmp package (ctan.org/pkg/hyperxmp). See listing 5.

6.2.4 Preserving links

As part of the work, a modernized version of the pax package (ctan.org/pkg/pax) from Heiko Oberdiek has been written. The new package is called newpax

Listing 5: Specifying a PDF standard

```
\RequirePackage{pdfmanagement-testphase}
\DeclareDocumentMetadata{pdfstandard=A-2b}
\documentclass{article}
\usepackage{hyperxmp}
\usepackage{hyperref}
....
```

(ctan.org/pkg/newpax), and it supports including PDFs inside another PDF without losing the links. It works with more backends and engines and, unlike pax, it doesn’t require an external Java application, instead using LuaTEX.

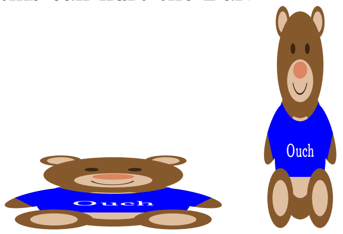
6.2.5 New form field code

Finally I would like to mention the code for form fields, which has been completely rewritten.

The new code is currently in a testphase package called l3pdffield-testphase, which is part of the pdfmanagement-testphase bundle. It is not yet decided if it will be part of hyperref like the old code or stay in a separate package.

One important change in this code is that it has been adapted to PDF 2.0. This means that it uses “appearances” in various places for the look of the fields (appearances are small graphics called “form XObjects”, stored in the PDF). Using appearances means that radio buttons or checkbox fields are no longer restricted to characters but can contain arbitrary images, e.g., animals from the tikzlings package (ctan.org/pkg/tikzlings).

But be careful! Appearances are squeezed by the PDF viewer into the dimensions of the form field, and this can hurt the Bär!



References

[1] U. Fischer. Creating accessible pdfs with LATEX. *TUGboat* 41(1):26–28, 2020. https://tug.org/TUGboat/tb41-1/tb127fischer-accessible.pdf

◇ Ulrike Fischer
LATEX project team
Mönchengladbach
Germany
ulrike.fischer (at) latex-project.org