Pdf tagging

Choices we made in ConTEXt

Introduction

Around 2023 tagged pdf became an issue, because some regulators started demanding them. Although we already supported it for a while,[1] it soon became clear that the standard was far from mature, that there were no tools that did something useful with these tags and that the waters were muddy. In 2025, when some laws started kicking in some European countries, with the USA following in 2026, one could start wondering where these demands really came from, who checked if they were reasonable and how they could be met, checked and imposed without stable tools, let alone a stable standard. Even when writing this (August 2025) we have no reliable way to see what is done with the tags although I have to admit that, because we have no access to Acrobat, I could be wrong.

For several years Mikael Sundqvist and I have been working on improved math rendering in LuaMetaTEX in the perspective of OpenType math, Unicode, user demands and detailed control. Faster machines and more memory are our friends here. As a side track we also looked into making math more accessible and our solution was to generate a verbose representation of a formula and embed that in the pdf file. Results of that effort were presented at BachoTEX 2024 where we also played with additional translations. In that perspective it also made sense to look into upgrading other tagged support in ConTEXt, and when we describe below sort of summarizes observations and experiences. At the ConTEXt 2025 meeting we presented the current state of affairs and had some discussions.

Accessibility in this perspective seems pretty much related to the ability to read a document from screen, using voice. The speech support that we tested in some other applications was kind of disappointing. Here we should notice that reading a text is not related to pdf. In fact, some pdf viewers just hook into the operating systems support for it and do a reasonable job. In that situation tagged reading can actually turn out to be worse. For instance, we were told that in some cases blind users have a well tuned system setup, play at a higher speed than normal, choose voices that match their taste, etc. A simple straightforward reading might be preferred over a confusing one based on tags and configuration options that accommodate various variants.

As far as we can see accessibility in the perspective of pdf doesn't come with well thought concepts of presenting it to users. And regulations are just that: imposing rules without much explanation. It's ConTEXt users who have to fulfill the demands who tell us that it's a bit of a gamble what is accepted. In the worst case a demotivating "just use this or that tool instead" is the answer one get when asking for clarification. A blind ConTEXt user on the mailing list explained that the documents that he produced with ConTEXt are mostly okay. He also mentioned that in nearly all the documents he encounters on the web the provided alternate texts are useless, which demonstrates that when demands are met that doesn't guarantee anything.

Logical structure in pdf is actually a rather old feature and version 1.3 already had it. Tagged pdf in the perspective of accessibility just builds upon this. Structure is a generic concept and also subjective. Consistency is a challenge and especially when we're talking of documents that have high visual demands or are artistic in nature it might not make sense to think structure. Not all documents are boring pages of sectioned and itemized laws, which from the perspective of governments can be a strong motive for accessibility.

We are aware of the fact that elsewhere in the TEX community there is being worked on tagged pdf but as demands, ways to structure, and internals of macro packages differ, solutions also will differ. Solutions likely depend on the field too. It makes a huge difference if you tag some article with limited structure and a small set of predictable structural components or instead a huge book with lots of complex structure and typographical solutions for presenting matters. With systems like TEX users might expect all to work out well magically.

At some point we might look into the more private tagging features, for instance to encode data (legal, financial, etc.) but as users already can have additional structural elements we can delay that. In this perspective we want to stress this: we start from structured ConTEXt and therefore need to work towards what pdf can do, not the reverse. We don't take pdf as starting

point and, apart from it making little sense, it would not help much either. We don't want to cripple ConTEXt or limit user input for a feature that has a good chance to be of temporary and limited importance; just look at other developments in documents rendering, informatics and technology in general.

When you look at the programs of user group meetings you will noticed that various attempts are made, seemingly independent, by suppliers of publishers[2] as well as for instance Plain TEX and LATEX world[3] where in the latter case user groups, Adobe, and the usa government financially support a multi-year project. It is not clear what the objectives of big tech and governments are but the fact that when one opens a reader (or browser for that matter) the rather intrusive suggestions coming from application of AI, combined with unknown scanning of ones document, and even worse the fact that your well structured content basically can be ripped off by anyone is not too motivating either. It often looks like regulations go beyond some initial (abstract) idealistic goal. It anyway makes that in ConTEXt we focus on what users want, and try to stay away from interference. We're not interested in supporting some viewer that every time you open a document asks you if it should make you a summary of something that you are supposed to understand by just reading.

We also don't feel the pressure of publishers but when articles are your main body of work, that might be different. But, anyway, that kind of support means that tagging is serious business. From the ConTEXt users we get signals that there are upcoming demands that have to be met, and sometimes it even sounds a bit panicking. It seems not enough to support only level one tagging, it has to be level two. The magic abbreviation is UA which stands for 'Universal Accessibility', where of course one can give that bombastic 'universal' classification some thoughts, but in this perspective both words can change meaning over time anyway. It anyway relates to validation of pdf documents in general, which introduces the 'A-4' standards that then relate to pdf 2. We just have to handle it, but based on testing by users, so it is for them that we describe part of the road to that goal.

Here we will discuss some observations and draw a few conclusions. We will explain how we are going to deal with this in a way that we hope is good enough for users and whoever put demands on them. If you came reading this far, you'll understand that we will not present a pretty and happy story so be prepared. We will not dive too deep into tagging in pdf, although you can be assured that we looked into and tested quite a bit. In that perspective we should mention that there are variants and even subjective takes where qualifications like 'well-formed' pop up, whatever that means as opinions can change over time. That is not a bandwagon we jump on. We like to mention that we rely on volunteers and free efforts to get things done in ConTEXt, so it has to be compensated by some fun, with in this perspective is hard.

As it is pretty subjective sometimes, I put my name on this wrap-up but the more general and objective observations and the work on this is very much the result of Mikael Sundqvist and I working on this together. We would not be where we are now with the update if it wasn't for extensive discussions, which gave us plenty good laughs but also sad observations. We did numerous tests, also with validation tools (to be) used at his university. For that we used real documents, think of 300+ page math books with thousands of formulas and much structure.

Starting from structure

Before we go into details it is important to notice that when we talk about tagging, we refer to adding directives to pdf documents in the perspective of accessibility. There are also tags in pdf that don't relate to that, but instead are more bound to applications. One can wonder to what extent the accessible ones are also application bound cq. derived, but if you read the documentation it follows that accessibility features are built on top of already present (old) features. Our experiences with pdf features beyond rendering are in the meantime such that we don't want to be too intrusive in the engine to deal with those demands. Instead we just use the already present structure related tagging, read: the features that drive the export. Unfortunately the pdf model is not the best match for that which means that we need to cheat every now and then. So be it: it's anyway better than to cripple ConTEXt's rather clean design for no reason also because one never know what surprises pdf brings in the future.

So, as intermezzo: when you turn on tagging in ConTEXt it will add structural directives. It will not relate to anything pdf. Only when you also enable some accessibility profile there will be role mapping. Even then you have a choice to what level. So, when below we discuss some issues, keep in mind that you can make choices. Also, in the ConTEXt distribution there is a document 'tagging-basic' that explains all this too but mixed with examples.

It all starts with structure. Now, we have to be honest, if you look at the average TEX encoded document you will probably notice that there will be some structure but that it can become ugly pretty easy. The worst are documents with no empty lines, as little spaces as needed, as many single or two character macros as one can get away with and hackery all over the place. In that perspective we are probably a bit obsessed because we

refuse to even look at an issue when a document source is not properly formatted and well structured. The fact that it is meant for a computer program to process it and therefore is unimportant doesn't impress us. You're welcome to use something different than ConTEXt. Just to give a simple example, compare:

```
\chapter[title=An example]

\startitemize
    \item one
    \item two
\stopitemize
```

with:

```
\startchapter[title=An example]

\startitemize
    \startitem one \stopitem
    \startitem two \stopitem
\stopitemize

\stopchapter
```

Both are valid but the second one has more structure. Okay, we can pretty well deal with the first one but there are cases where the derived structure is bit more of a pain to deal with than the explicit one. In retrospect the first alternative should never have been supported. It also has more overhead and it was the initial version, interestingly in a time when overhead mattered more than today. The sole reason that we didn't immediately discarded the less verbose version is simple: it was kind of custom in the TEX universe to minimize coding in the input. And yes, when you want flexible typesetting that can backfire which is why we quickly started discouraging it.

However, because in ConTEXt we always had structure deep down, adding support for tagged pdf was no big deal. It actually followed up on already present usage of structure, namely export of content resulting from rendering input, but for now we put that aside. Tagging and export are both very much backend related and at the input level and when typesetting little to nothing relates to it. This means that tagging should have no real consequences for the user coding a document, apart maybe from adding meaningful descriptions to images and optionally tagging paragraphs. However, when version 2 became the demand, and when the process of patching the standard, tools and interpretation started, we ended a period of adaptation, not so much of the way we did things, but more in trying to accommodate the new limitations and constraints, often for the worse. Over time different resources came available and it's not easy to figure out the logic in naming but we've been around since the start so we are accustomed to it. Probably the main issue users have to deal with are adding a description to images in which we distinguish between export and tagged pdf as they serve different purposes.

The focus on structure in ConTEXt comes from one of its main areas of application: education, and without structure, educating becomes pretty hard. As a result, when we are told to look at structure we automatically assume a certain logic. The most obvious are parts, chapters and sections. There can be itemized lists, tables and figures. All these are vary basic and presumably not open to confusions. Neither are descriptions, captions to floats, section heads, highlights and notes.

Some resources

Before we move on to pdf we need to mention some resources.[4] When pdf came around books were still printed and the pdf reference manual was a well printed book. From version 1.3 on the online versions became the resource and while the look and feel remained the same the number of pages grew. Around 2008 we'd arrived at the "pdf Reference sixth edition by Adobe" subtitles "Portable Document Format Version 1.7". For many years there was talk of a follow up but even when 2.0 showed up, it mostly was just 1.7 and this is what TEX engines stuck to. There has been no 1.8 and 1.9 to our knowledge. The main reference is referred to as "Part 1: PDF 1.7 2008-7-1". After that came "Part 2: PDF 2.0 2022" which in due time became an iso standard.

In addition to the 1.7 standard which served us well for a decade there were various documents, for instance related to forms (widgets) and JavaScript in pdf that were useful but not relate to what we discuss here.[5] Of course there were also documents that describe some of the technology involved like fonts.

Instead of upgrading the standard addenda showed up, like "Document management applications — Electronic document file format enhancement for accessibility — Part 2: Use of ISO 32000-2 (PDF/UA-2)" in 2024, as well as "ISO 19005-4:2020(E) Document management — Electronic document file format for long-term preservation — Part 4: Use of ISO 32000-2 (PDF/A-4)". The "TECHNICAL SPECIFICATION ISO/TS 32005 First edition 2023-07" subtitled " Document management — Portable Document Format — PDF 1.7 and 2.0 structure namespace inclusion in ISO 32000-2" is again an iso standard and overlaps with the main 2.0 standard. But we had to check it out anyway.[6]

Finally we mention the "Well-Tagged PDF (WTPDF) Using Tagged PDF for Accessibility and Reuse in PDF 2.0 2024" by the pdf association. There are predecessors like "Tagged PDF Best Practice Guide: Syntax For developers implementing ISO 14289-1 (PDF/UA) Version 1.0 (June 2019)" which targets 1.7 but got updates later on. It all shows that structure and tagging has a long

history in pdf and that there have been attempts to clarify the situation. But as we will see, it can still be confusing. What is interesting about the last mentioned document is that it identifies some issues and also suggests that applications should be tolerant and deal with it. But that is not what seem to happen when we progress to version 2, and one can wonder why that is.

So if we stick to numbers we have:

32000-1:2008	pdfversion 1.7	
32000-2:2020	pdfversion 2.0	
14289-1:2014	32000-1 UA-1	tagged pdf
TS 32005	32000-2	clarifications
14289-2:2024	32000-2 UA-2	tagged pdf
WTPDF 2024	2.0 and UA-2	recommendations

But you can find more at for instance the website of the pdf association, although it looks like there is overlap which is not uncommon, just look at our manuals that suffer from the same redundancy as side effect of developments. The main problem we see is that when you are an early adopter you now have to adapt again and not always for the better as you already have solutions in place. In practice it is the evolving validations and interpretations that spoil the game and it simply doesn't pay off to come back to this theme again and again (these are not projects but unpaid activities and there are more interesting typesetting related hobbies). But . . . back to work.[7]

The pdf approach

So, when we have to consider structuring the content in a pdf document you can imagine that when we already have structure in ConTEXt, that it should not be that hard to translate that into structure in a pdf file. And indeed, already long ago we could support it. But as said, nothing was done with it and when it became clear that something was going to be done with it things got weirdly confusing. For the record: we actually test tagging on educational documents and an occasional manual, if only because, unlike users on the mailing list, we don't encounter other (substantial in size and complexity) documents ourselves.

First of all, the basics of tagged pdf in the perspective of accessibility were laid down in pdf version 1.7 and somewhat extended in version 2.0. Then after a while, some found out that the definitions and descriptions were suboptimal so a process of patching started. Now, one can wonder where the proposed (or by now mandate) tags came from and looking at the list of possible tags and sub-tags show such a confusing mess that it likely comes from some application. If one really starts from structured documents, this is not what one comes up with. And then, when one starts from that confusing mess, trying to accommodate structured content in retrospect, it only makes things worse, because again a certain panic in the need to support it makes one look at the application at hand. This is all not the result of robust result or experience. One starts running around in circles. It is a bit like the early days of pdf when interactive features showed up that could not really be tested beforehand, and although TEX could easily adapt that also backfired a bit when successive versions of the pdf standards or interpretations of it changed, and/or the renderer was adapted. It's part of development but when downward compatibility is dropped it hurts the portability that gets advocated.

There are a few indicators that demonstrate this. For instance, there is support for so called `Ruby` structures that have three sub structures: `RB`, `RP` and `RT`. There is some detail there and actually there can be plenty more outside those three too. Then there is `Lbl` (mixed case), but where in a decent environment one can recognize a label, optional number and caption text, there is no such structure. In a similar fashion there is for instance `H3` that doesn't really encourage a number and title separation. The simple fact that rubies have detail and sectioning doesn't indicates that these tags don't really come from structured thinking but more from some application or specific intended usage. To this we can add the naming: we save characters on a list tag `L`, but waste plenty on the way more likely to be present Artifact. Some come straight from html, like `Table`, `Thead`, `TR` and `TD` but with different casing, and again a lot can be mixed in outside the suggested structure in this. An inconsistent naming like this is already a bad omen.

What can go into what is actually a puzzle in itself. Every tag has a specification (a list of other tags) that is flagged. Most can be used zero of more times, some only one and often there is an indicator telling the reader to look at (one of) the specification(s) to which of course we now have to add addenda, either of not themselves documents. And of course, it being an iso standard, not every developer can access them easily (open source and free software developers normally don't have the resources) and as these documents evolve they are secret anyway. So there is a delay in supporting the changes.

Because of the many possible but far from consistent possibilities to nest tags we found it pretty hard to come up with a mapping from ConTEXt structure onto pdf structure. It is a process of trial and error, one that needs to be repeated when a validation get an update because of a fix or because the standard became more standard. We're not sure how an application reads the text: will it use the alternative text, will it speak the optional title, does it indicate a sectioning level. etc. For a moment consider this in the perspective of regulations: what is okay today, is invalid tomorrow! It

can be due to clarification, so that a generator can do a better job, or it can be due to expected (or encountered) limitations in consuming software. To give an example of the latter: the generic H (head) tag is deprecated (which in this context means that it triggers a validation error) in favor of H1 and friends, and the argument is that it's hard for applications to deal with nesting. This is kind of curious because there is a crazy amount of weird (inconsistent) nesting possible with all these tags. The main warning here is that users should really try to imagine what a tagged document will present you. If it really matters you might need to adapt a document.

So what structure do we have in ConTEXt and what tags do we find in pdf? Let's start with the later. The colors in the table have the following meaning:

obsolete	These seem to be declared obsolete so we can best not use them. There has been changes between both versions.
useless	We found that these are confusing, have strange nesting constraints or are just useless.
internal	We use these only internally so don't map onto them.
possible	These can be used, although we might change that classification.

This will be explained later with some examples. Here we show a rather long list, and the size will give you an idea of what we're dealing with. In case you think it's a waste of paper, consider the fact that you have to pay a hefty price per page for a standard that put each row on a nearly empty page and then have to check it again and again as you try to implement it. Browsing a few pages like the following might make you appreciate the fact that a macro package tries to do that work for you.

Annot	Annot Art Artifact BibEntry Code Div Em FENote Figure Formula Lbl Link NonStruct Note Quote Ruby Sect Span Strong Sub Warichu content item
Art	Annot Artifact Aside BlockQuote Caption Code Div DocumentFragment FENote Figure Form Formula Hn Index L Lbl Link NonStruct Note P Part Private Sect TOC Table Title
Artifact	Annot Art Artifact Aside BibEntry BlockQuote Code Div Document DocumentFragment Em FENote Figure Form Formula Hn Index L LBody LI Lbl Link NonStruct Note P Part Private Quote RB RP RT Reference Ruby Sect Span Strong Sub TBody TD TFoot TH THead TOC TOCI TR Table Title WP WT Warichu content item
Aside	Annot Art Artifact BlockQuote Code Div Document DocumentFragment FENote Figure Form Formula Hn Index L Lbl Link NonStruct Note P Part Private Reference Sect TOC Table content item
BibEntry	Annot Artifact Div Em FENote Figure Lbl Link NonStruct Note P Part Private Reference Span Strong content item
BlockQuote	Annot Art Artifact BlockQuote Code Div Document DocumentFragment FENote Figure Form Formula Hn Index L Lbl Link NonStruct Note P Part Private Reference Sect TOC Table Title content item
Caption	list is irrelevant
Code	Annot Artifact BibEntry Div Em FENote Link NonStruct Note Part Private Reference Span Strong content item
Div	inherited
Document	Annot Art Artifact Aside BlockQuote Code Div Document DocumentFragment FENote Figure Form Formula Hn Index L Link NonStruct Note P Part Private Sect TOC Table Title
Document-Fragment	Annot Art Artifact Aside BlockQuote Code Div Document DocumentFragment FENote Figure Form Formula Hn Index L Link NonStruct Note P Part Private Sect TOC Table Title
Em	Annot Artifact BibEntry Code Em FENote Figure Form Formula Lbl Link NonStruct Note Private Quote Reference Ruby Span Strong Sub Warichu content item
FENote	Annot Art Artifact Aside BlockQuote Code Div Em FENote Figure Form Formula L Lbl Link NonStruct Note P Part Private Quote Reference Ruby Sect Span Strong Sub Table Warichu content item
Figure	Annot Art Artifact Aside BibEntry BlockQuote Code Div Em FENote Figure Form Formula Hn Index L Lbl Link NonStruct Note P Part Private Quote Reference Ruby Sect Span Strong Table Warichu content item
Form	Artifact BibEntry Caption Div FENote Lbl NonStruct Note Private content item
Formula	Annot Art Artifact Aside BibEntry BlockQuote Code Div Em FENote Figure Form Formula Hn Index L Lbl Link NonStruct Note P Private Quote Reference Ruby Span Strong Sub Table

	Warichu content item
H	list is irrelevant
Hn	Annot Artifact BibEntry Code Em FENote Figure Form Formula Lbl Link NonStruct Note Private Quote Reference Ruby Sect Span Strong Sub Warichu content item
Index	Annot Artifact Caption Div FENote Figure Formula H Hn L NonStruct Note P Part Private Reference Sect Table
L	Artifact L LI NonStruct Private
LBody	Annot Art Artifact Aside BibEntry BlockQuote Code Div Em FENote Figure Form Formula Hn Index L Link NonStruct Note P Part Private Quote Reference Ruby Sect Span Strong Sub Table Warichu content item
LI	Artifact Div LBody Lbl NonStruct Private content item
Link	Annot Art Artifact BibEntry Code Div Em FENote Figure Formula Lbl NonStruct Note Private Quote Reference Ruby Sect Span Strong Sub Warichu content item
NonStruct	list is irrelevant
Note	Annot Art Artifact Aside BibEntry BlockQuote Code Div Em FENote Figure Form Formula Index L Lbl Link NonStruct Note P Part Private Quote Reference Ruby Sect Span Strong Sub Table Warichu content item
P	Annot Artifact BibEntry Code Em FENote Figure Form Formula L Lbl Link NonStruct Note Private Quote Reference Ruby Span Strong Sub Table Warichu content item
Part	list is irrelevant
Private	Annot Art Artifact Aside BibEntry BlockQuote Caption Code Div Document DocumentFragment Em FENote Figure Form Formula H Hn Index L LBody LI Lbl Link NonStruct Note P Part Private Quote RB RP RT Reference Ruby Sect Span Strong Sub TBody TD TFoot TH THead TOC TOCI TR Table Title WP WT Warichu content item
Quote	Annot Artifact BibEntry Code Em FENote Figure Form Formula Lbl Link NonStruct Note Private Quote Reference Ruby Span Strong Sub Warichu content item
RB	Annot Artifact Em Form Link NonStruct Private Quote Reference Span Strong Sub
RP	Annot Artifact Em Form Link NonStruct Private Quote Reference Span Strong Sub
RT	Annot Artifact Em Form Link NonStruct Private Quote Reference Span Strong Sub
Reference	Annot Artifact BibEntry Em FENote Figure Lbl Link NonStruct Note Private Span Strong content item
Ruby	NonStruct Private RB RP RT content item
Sect	Annot Art Artifact Aside BlockQuote Caption Code Div DocumentFragment FENote Figure Form Formula H Hn Index L Lbl Link NonStruct Note P Part Private Sect TOC Table Title
Span	Annot Artifact BibEntry Code Em FENote Figure Form Formula Lbl Link NonStruct Note Private Quote Reference Ruby Span Strong Sub Warichu content item
Strong	Annot Artifact BibEntry Code Em FENote Figure Form Formula Lbl Link NonStruct Note Private Quote Reference Ruby Span Strong Sub Warichu content item
Sub	Annot Artifact BibEntry Code Em FENote Figure Form Formula L Lbl Link NonStruct Note Private Quote Reference Ruby Span Strong Warichu content item
TBody	Artifact NonStruct Private TR
TD	Annot Art Artifact BibEntry Code Div Em FENote Figure Form Formula Hn Index L Lbl Link NonStruct Note P Private Quote Reference Ruby Sect Span Strong Table Warichu content item
TFoot	Artifact NonStruct Private TR
TH	Annot Art Artifact BibEntry Code Div Em FENote Figure Form Formula Hn Index L Lbl Link NonStruct Note P Private Quote Reference Ruby Sect Span Strong Table Warichu content item
THead	Artifact NonStruct Private TR
TOC	Artifact NonStruct Part Private TOC TOCI
TOCI	Artifact Div Lbl NonStruct P Private Reference TOC
TR	Artifact NonStruct Private TD TH
Table	Artifact Caption NonStruct Private TBody TFoot THead TR
Title	Annot Artifact Aside BibEntry Code Div Em FENote Figure Form Formula L Lbl Link NonStruct Note P Part Private Quote Reference Ruby Span Strong Table Warichu content item

```
WP          Annot Artifact Em Form Link NonStruct
            Private Quote Reference Span Strong
            Sub
WT          Annot Artifact Em Form Link NonStruct
            Private Quote Reference Span Strong
            Sub
Warichu     NonStruct Private WP WT content item
```

The reason for our classification not only comes from our lack of imagination, but is also driven by testing if a predictable validation success is possible. Also, we had to start somewhere in order to be able to fill in gaps and experiment which what is currently accepted, which is less than before. So the above is a bit fluid: the picture might become different over time, but we hope that you get the idea. We don't want to end up in lame excuses like "You're not supposed to nest this structural element into that one". We also don't want to risk that what works in a simple article fails on a book. Getting a nice looking consistent document is already hard enough.

What we classify as **useless** is often the result of the simple fact that it is flagged as "consult the standard for usage" and if you ever looked at the standard you will understand that this is not what we ask users to do. The concept of what tags can be children or parents is also complicated by the fact that there can be very small differences in these lists that indicate some logic but as well defeat it.

Although it is still a valid version one tag, we don't even mention `H` any more; with `HN` (or `Hn`) single capital followed by a digit is meant. So how useful are the plenty left over? In order to satisfy the validation tool one has to be sure not to mix these tags the wrong way. For that purpose we provide a few variant mappings: the safe built-in default, a basic mapping and a more detailed one. Users can start from that.

Matching both models

Now, before we look (again) at the way ConTEXt sees structure, we already can say that most of the previously shown colorful table makes little sense so we will condense this table later on to what is usable. We mainly have to make sure that the endpoints, that is where the content sits, conforms to what the standard expects. When in the table above you see **content item** that means that there can be some real rendered text. So, if for some reason we're not yet there in terms of structure, we need to inject a terminal structure element, most likely a `P` or `Span` and maybe a `Div`, although one should not confuse that one with the generic html cousin. Oh, and `Sub` is not a subscript! There is a use for `Sect` but it's not a section like `section`. Just keep this in mind.

Another important pitfall is in the concept of paragraphs. If you look at a typeset document, you will for sure recognize them as such: a blob of typeset text. But if you look closely you can see that often there is a mix in of elements: quotations, emphasized words, code, math. So we have a mix. In TEX a paragraph is what starts with some content, and runs till we see `\par` (or equivalent empty line). One can say that it runs from `\parindent` (with `\everyparbegin` and `\everypar`) upto `\par` (with `\everyparend` and `\wrapuppar`).[8] Add some grouping and anchored content and you will see the complication. Also, think of for instance explicit breaks, local boxes, margin notes, footnotes, and the mix becomes more mixed.

Users can explicitly tag a paragraph with:

```
\startparagraph
some text ... what a mix
\stopparagraph
```

but that's an extra amount of work not many users appreciate an dit doesn't always work well with other structure either. So, why can't TEX (read: ConTEXt) do that for us? It actually does that but keeping track of what sits in the same paragraph doesn't solve the endpoint problem. When we use the above commands, we add an explicit level while lack of them forces us to decide if what we just encountered actually is one. Heuristics are fine but fighting them when they are wrong is hard. There is no real mixed content element in pdf so we just assume a paragraph even if in fact a combination of those end points plus some others make a real one. It doesn't matter much because we're just talking tagged text, we don't need it for exporting the original to some other format.

An interesting one is `NonStruct` as it is permitted everywhere but in the end not acceptable as endpoint, that is: at some moment in 2025 the validation tool started complaining about it. But don't worry, we can create an endpoint inside. After all, who cases about yet another level of structure and the additional bloat is small compared to the already bloated file. So, in the perspective of ConTEXt the `NonStruct` tag is (now) a placeholder. Interesting is that using `Artifact` instead works okay while it is less meaningful. This is again a reason why we end up with more `P` roles than we like which in itself is tricky because one really needs to do some structural guess work to make sure they don't get nested. The inline, mixed and display content model in pdf is pretty weak, limited and unforgiving.

There is also this somewhat confusing `ua1` vs `ua2` issue. Tags are defined in the first (version 1.7) namespace and some are also defined in the second (version 2.0) namespace. Among the ones only in `ua1` is `Quote`. And one can mix namespaces, and even add one own as long as in the end there is some known tag involved.

The reason (we think) is that `Quote` is still the same while `BlockQuote` has an adapted interpretation. We are not that interested in possible differences because in the end a `BlockQuote` is not what we think it is: where in a ConTEXt document one immediately thinks of `quotation` versus `quote`, in pdf the bigger thing can be a complete document that of course itself can have `BlockQuotes` than then can have documents etc. What was the reason for ditching `H`, wasn't the argument 'complexity' (here we used `\quote`)? For the moment we don't block it, just like we keep `FENote` around but you are warned.

Again the table is quite some pages but it also indicates how much potential (rich) structure one has to map onto something limited as pdf offers. Imagine that you would have to do that manually every time, just because mapping is a bit guesswork. This also shows how important it is that you structure your document well and consistently!

element	nature	pdf	basic
block	display	NonStruct	
break	display	Artifact	
citation	inline	NonStruct	Artifact
cite	inline	NonStruct	
combination	display	NonStruct	Table
combinationcaption	mixed	NonStruct	TD
combinationcontent	mixed	NonStruct	TD
combinationpair	display	NonStruct	TR
construct	inline	NonStruct	
constructcontent	inline	NonStruct	
constructleft	inline	NonStruct	
constructright	inline	NonStruct	
crumpy	mixed	NonStruct	
delimited	inline	NonStruct	Artifact
delimitedblock	display	NonStruct	BlockQuote
delimitedcontent	inline	NonStruct	NonStruct
delimitedsymbol	inline	NonStruct	NonStruct
description	display	NonStruct	Sect
descriptioncontent	mixed	NonStruct	Div
descriptionsymbol	inline	NonStruct	NonStruct
descriptiontag	mixed	NonStruct	Lbl
displaymath	display	NonStruct	
division	display	NonStruct	
document	display	Document	
documentpart	display	Part	
dummy	mixed	Artifact	
float	display	NonStruct	Aside
floatcaption	mixed	NonStruct	Div
floatcontent	mixed	NonStruct	Div
floatlabel	inline	NonStruct	Lbl
floatnumber	inline	NonStruct	Lbl
floattext	mixed	NonStruct	P
formula	display	NonStruct	Div
formulacaption	mixed	NonStruct	Artifact
formulacontent	display	NonStruct	Formula
formulalabel	mixed	NonStruct	NonStruct
formulanumber	mixed	NonStruct	Lbl
formulaset	display	NonStruct	Div
highlight	inline	NonStruct	Artifact
ignore	mixed	NonStruct	
image	mixed	Artifact	Figure
inlinemath	inline	NonStruct	
item	display	NonStruct	LI
itembody	display	NonStruct	NonStruct
itemcontent	mixed	NonStruct	LBody
itemgroup	display	NonStruct	L
itemhead	display	NonStruct	NonStruct
items	display	Table	
itemsymbol	inline	TD	
itemsymbols	mixed	TR	
itemtag	mixed	NonStruct	Lbl
itemtext	inline	TD	
itemtexts	mixed	TR	
label	mixed	NonStruct	
line	mixed	NonStruct	
linenumber	inline	NonStruct	
lines	display	NonStruct	
linetext	inline	NonStruct	
link	inline	Link	
list	display	NonStruct	L
listcontent	mixed	NonStruct	LBody
listdata	mixed	NonStruct	NonStruct
listitem	display	NonStruct	LI
listpage	mixed	NonStruct	Lbl
listtag	mixed	NonStruct	Lbl
listtext	inline	NonStruct	NonStruct
m	mixed	Artifact	Artifact
maction	display	Span	
marginanchor	inline	NonStruct	
margintext	inline	NonStruct	Aside
margintextblock	inline	NonStruct	
math	inline	NonStruct	Formula
mc	mixed	Span	
mdelimited	display	Span	
mdelimitedstack	display	Span	
mdigits	mixed	Span	
merror	mixed	Span	
metadata	display	NonStruct	
metavariable	mixed	NonStruct	
mextensible	display	Span	
mfenced	display	Span	
mfrac	display	Span	
mfraction	display	Span	
mfractionstack	display	Span	
mfunction	mixed	Span	
mfunctionstack	display	Span	
mi	mixed	Span	
mid	inline	NonStruct	
mmultiscripts	display	Span	
mn	mixed	Span	
mo	mixed	Span	
mover	display	Span	
mpgraphic	mixed	Artifact	
mprescripts	mixed	Span	
mroot	display	Span	
mrow	display	Span	
ms	mixed	Span	
msqrt	display	Span	
mstack	display	Span	
mstacker	display	Span	
mstackerbot	display	Span	
mstackermid	display	Span	
mstackertop	display	Span	
msub	display	Span	
msubsup	display	Span	
msup	display	Span	
mtable	display	mtable	
mtd	display	mtd	
mtext	mixed	Span	

mtr	display	mtr	
munder	display	Span	
munderover	display	Span	
munit	mixed	Span	
narrower	display	NonStruct	Div
navigation	mixed	Artifact	
navigationaction	display	Artifact	
navigationbutton	mixed	Artifact	
navigationmenu	display	Artifact	
navigationmenuitem	display	Artifact	
navigationpage	inline	Artifact	
number	mixed	NonStruct	
ornament	inline	NonStruct	
p	mixed	P	P
paragraph	mixed	P	
placeholder	inline	NonStruct	
private	mixed	NonStruct	
pubfld	inline	NonStruct	
publication	mixed	NonStruct	
publications	display	NonStruct	
quantity	inline	NonStruct	
reference	inline	Artifact	
register	display	NonStruct	
registercontent	mixed	NonStruct	
registerentries	display	NonStruct	
registerentry	display	NonStruct	
registerlocation	inline	NonStruct	
registerpage	mixed	NonStruct	
registerpagerange	mixed	NonStruct	
registerpages	mixed	NonStruct	
registersection	display	NonStruct	
registersee	mixed	NonStruct	
registerseparator	inline	NonStruct	
registertag	mixed	NonStruct	
section	display	NonStruct	Sect
sectionblock	display	NonStruct	NonStruct
sectioncaption	display	NonStruct	Div
sectioncontent	display	NonStruct	Div
sectionnumber	mixed	NonStruct	Lbl
sectiontitle	mixed	NonStruct	Lbl
sorting	inline	NonStruct	
sub	inline	NonStruct	
subfloat	mixed	NonStruct	Div
subformula	display	NonStruct	
subparagraph	mixed	P	
subsentence	inline	NonStruct	
subsentencecontent	inline	NonStruct	
subsentencesymbol	inline	NonStruct	
subsup	inline	NonStruct	
sup	inline	NonStruct	
synonym	inline	NonStruct	
t	mixed	Span	Span
table	display	Table	
tablebody	display	TBODY	
tablecell	mixed	TD	
tablefoot	display	TFOOT	
tablehead	display	THEAD	
tableheadcell	mixed	TH	
tablerow	display	TR	
tabulate	display	Table	
tabulatebody	display	TBODY	
tabulatecell	mixed	TD	
tabulatefoot	display	TFOOT	
tabulatehead	display	THEAD	
tabulateheadcell	mixed	TH	
tabulaterow	display	TR	
text	inline	Span	
textdisplay	display	NonStruct	
unit	inline	NonStruct	Artifact
userdata	display	NonStruct	
verbatim	inline	NonStruct	Code
verbatimblock	display	NonStruct	Div
verbatimline	mixed	NonStruct	Sub
verbatimlines	display	NonStruct	Code
verse	display	NonStruct	
versecontent	inline	NonStruct	
verseseparator	inline	NonStruct	
versetag	inline	NonStruct	
vspace	display		

For all these elements there is also a **namespace**, which is always set to context except in the case of the last one, **vspace**. For brevity, this column is removed from the table (red.)

Structure often looks similar across applications: sectioning, itemization, descriptions, tables, highlights, quotations are rather common in documents. It is for that reason a bit weird that pdf tags are so inconsistent. Take for instance sectioning. The `Part` and `Sect` are not really for sectioning, there one has to use `Hn` with the `n` being a digit. Think html here. For itemize like tags there are possibilities to indicate that we have a numbered or unnumbered list of items, but for sectioning ... indeed we don't have such attributes. So how do we tag them? Let's look at how it ends up in xml speak:

```
<section>
  <sectioncaption>
    <sectionnumber> 1 </sectionnumber>
    <sectiontitle>  Hi There </sectiontitle>
  </sectioncaption>
  <sectioncontent>
    We're here!
  </sectioncontent>
</section>
```

Inside section content we can have nested sections. Although the number can carry a level, say `1.2.1` the level actually is implied from the nesting. However, we have what we call 'some detail'. For instance a specific level can be detailed as 'chapter' and we put that in the pdf tag specifier at hand as title. Let's entertain you now with some pdf:

```
20 0 obj
<< /K [ 21 0 R 27 0 R ] ... /S /section /T
  (chapter) >>
endobj
21 0 obj
<< /K [ 22 0 R  23 0 R 25 0 R ] ... /S
  /sectioncaption >>
endobj
22 0 obj
<< /K [ 1 ] ... /S /reference >>
endobj
23 0 obj
<< /K [ 24 0 R ] ... /S /sectionnumber >>
```

```
endobj
24 0 obj
<< /K [ 2 ] ... /S /paragraph >>
endobj
25 0 obj
<< /K [ 26 0 R ] ... /S /sectiontitle >>
endobj
26 0 obj
<< /K [ 3 ] ... /S /paragraph >>
endobj
```

This only shows the caption but that already is a quite extensive chain of tags with as endpoints paragrapghs that map onto P. That last endpoint is automatically injected because the number and title are both NonStruct by default and so in there we can safely use a P. It's a robust way to support arbitrary content: we always have to make sure that the endpoint validates. The reference is there as anchor for hyperlinks (a mess in itself) and that ends as a pretty safe Artifact. Let's cross our fingers that the standard and validator keeps up on that. The basic variant maps the titles in H pdf tags and one can optionally set the outer mist level.

Tuning by users

We only provide a generic setup, accompanied by what we call a basic setup that users can enable. From that users can roll out their own final mapping, just because they know what they (have to) deliver. The default setup is as follows:

```
section        = {
    namespace = "context", nature = "display",
      pdf = "NonStruct"
}
sectioncaption = {
    namespace = "context", nature = "display",
      pdf = "NonStruct"
}
sectionnumber  = {
    namespace = "context", nature = "mixed",
      pdf = "NonStruct"
}
sectiontitle   = {
    namespace = "context", nature = "mixed",
      pdf = "NonStruct"
}
sectioncontent = {
    namespace = "context", nature = "display",
      pdf = "NonStruct"
}
```

so we play very safe. One can however overload that mapping and how to do that is explained elsewhere in manuals.

To give an idea:

```
section        = { pua = "ua2", pdf = "Sect" }
sectioncaption = { pua = "ua2", pdf = "Div"  }
sectiontitle   = { pua = "ua2", pdf = "Span" }
sectionnumber  = { pua = "ua2", pdf = "Span" }
sectioncontent = { pua = "ua2", pdf = "Div"  }
```

In that same setup file we also deal with the Hn mapping:

```
sectioncaption = {
  criterium = "parents",
  mapping   = {
    ...
    chapter = { pua = "ua2", tag = "
      section_title_2", pdf = "H2" },
    title   = { pua = "ua2", tag = "
      subject_title_2", pdf = "H2" },
    ...
  },
}
```

The real basic section variant definition is a bit more complex. The level key in \setuptagging can be used to set the first relevant level, so if needed one can start \section at H1 but of course it depends on the document and how it fits in a hierarchy. The user can decide if section 1.2.3 gets announced by the reading application as head level 3 or 1. In that respect the generic H is a loss.

But we deliberately leave it to the user to load one or more such overloads because only the user know what is really the intention. However, the basic setup

```
\setuptagging[preset=basic]
```

could work ok. We only permit (support) a subset of pdf tags. We start with general tags. In the next tables we still show tags that we tested, in red, but in the end we just rejected them.

Hn	Annot BibEntry Code Em Figure Form Formula Lbl Link NonStruct Note Private Quote Reference Ruby Sect Span Strong Sub Warichu content item
Aside	Annot Art Code Div Document Document-Fragment Figure Form Formula Hn Index L Lbl Link NonStruct Note P Private Reference Sect TOC Table content item
Code	Annot BibEntry Div Em Link NonStruct Note Private Reference Span Strong content item
Figure	Annot Art Aside BibEntry Code Div Em Figure Form Formula Hn Index L Lbl Link NonStruct Note P Private Quote Reference Ruby Sect Span Strong Table Warichu content item
Formula	Annot Art Aside BibEntry Code Div Em

	Figure Form Formula Hn Index L Lbl Link NonStruct Note P Private Quote Reference Ruby Span Strong Sub Table Warichu content item
Hn	Annot BibEntry Code Em Figure Form Formula Lbl Link NonStruct Note Private Quote Reference Ruby Sect Span Strong Sub Warichu content item
Sect	Annot Art Aside Caption Code Div DocumentFragment Figure Form Formula H Hn Index L Lbl Link NonStruct Note P Private Sect TOC Table Title
Sub	Annot BibEntry Code Em Figure Form Formula L Lbl Link NonStruct Note Private Quote Reference Ruby Span Strong Warichu content item

Followed by the neutral ones:

Div	inherited
Span	Annot BibEntry Code Em Figure Form Formula Lbl Link NonStruct Note Private Quote Reference Ruby Span Strong Sub Warichu content item
P	Annot BibEntry Code Em Figure Form Formula L Lbl Link NonStruct Note Private Quote Reference Ruby Span Strong Sub Table Warichu content item
Private	Annot Art Aside BibEntry Caption Code Div Document DocumentFragment Em Figure Form Formula H Hn Index L LBody LI Lbl Link NonStruct Note P Private Quote RB RP RT Reference Ruby Sect Span Strong Sub TBody TD TFoot TH THead TOC TOCI TR Table Title WP WT Warichu content item

These relate to styling:

Em	Annot BibEntry Code Em Figure Form Formula Lbl Link NonStruct Note Private Quote Reference Ruby Span Strong Sub Warichu content item
Strong	Annot BibEntry Code Em Figure Form Formula Lbl Link NonStruct Note Private Quote Reference Ruby Span Strong Sub Warichu content item

Lists have multiple tags and (surprise) have some structure:

L	L LI NonStruct Private
LI	Div LBody Lbl NonStruct Private content item
LBody	Annot Art Aside BibEntry Code Div Em Figure Form Formula Hn Index L Link NonStruct Note P Private Quote Reference Ruby Sect Span Strong Sub Table Warichu content item

Of course we have (modeled after html) tables:

Table	Caption NonStruct Private TBody TFoot THead TR
THead	NonStruct Private TR
TFoot	NonStruct Private TR
TBody	NonStruct Private TR
TR	NonStruct Private TD TH
TD	Annot Art BibEntry Code Div Em Figure Form Formula Hn Index L Lbl Link NonStruct Note P Private Quote Reference Ruby Sect Span Strong Table Warichu content item
TH	Annot Art BibEntry Code Div Em Figure Form Formula Hn Index L Lbl Link NonStruct Note P Private Quote Reference Ruby Sect Span Strong Table Warichu content item

And a specialized:

Ruby	NonStruct Private RB RP RT content item
RB	Annot Em Form Link NonStruct Private Quote Reference Span Strong Sub
RP	Annot Em Form Link NonStruct Private Quote Reference Span Strong Sub
RT	Annot Em Form Link NonStruct Private Quote Reference Span Strong Sub

and:

Warichu	NonStruct Private WP WT content item
WP	Annot Em Form Link NonStruct Private Quote Reference Span Strong Sub
WT	Annot Em Form Link NonStruct Private Quote Reference Span Strong Sub

If we omit the unsupported (red) tags we get our final, smaller subset. We also leave out the ruby and warichu tags. Showing the table ones makes little sense too, so in the end the repertoire that users can consider for alternative mappings is small.

Artifact	Artifact Aside BlockQuote Code Div FENote Figure Formula Hn L LBody LI Lbl NonStruct P Part Private Sect Span Sub content item
Aside	Artifact BlockQuote Code Div FENote Figure Formula Hn L Lbl NonStruct P Part Private Sect content item
BlockQuote	Artifact BlockQuote Code Div FENote Figure Formula Hn L Lbl NonStruct P Part Private Sect content item
Code	Artifact Div FENote NonStruct Part Private Span content item
Div	inherited
FENote	Artifact Aside BlockQuote Code Div FENote Figure Formula L Lbl NonStruct P Part Private Sect Span Sub content item

Figure	Artifact Aside BlockQuote Code Div FENote Figure Formula Hn L Lbl NonStruct P Part Private Sect Span content item
Formula	Artifact Aside BlockQuote Code Div FENote Figure Formula Hn L Lbl NonStruct P Private Span Sub content item
Hn	Artifact Code FENote Figure Formula Lbl NonStruct Private Sect Span Sub content item
L	Artifact L LI NonStruct Private
LBody	Artifact Aside BlockQuote Code Div FENote Figure Formula Hn L NonStruct P Part Private Sect Span Sub content item
LI	Artifact Div LBody Lbl NonStruct Private content item
P	Artifact Code FENote Figure Formula L Lbl NonStruct Private Span Sub content item
Private	Artifact Aside BlockQuote Code Div FENote Figure Formula Hn L LBody LI Lbl NonStruct P Part Private Sect Span Sub content item
Sect	Artifact Aside BlockQuote Code Div FENote Figure Formula Hn L Lbl NonStruct P Part Private Sect
Span	Artifact Code FENote Figure Formula Lbl NonStruct Private Span Sub content item
Sub	Artifact Code FENote Figure Formula L Lbl NonStruct Private Span content item

But this is just one of the aspects to deal with. Let's look at how we deal with linking section titles with tables of contents. For this it is good to realize that hyperlinks or hypertext is something that started getting implemented in applications that showed text in the 80's. Around the 90's TEX systems could cope with that but it was very much dependent on the viewer.

The generic dvi output format can contain directives (specials) that register source and target (link and reference) and we used that for instance to create interactive documents that were extensively hyperlinked and could be watched in for instance dviwindo. When pdf and Acrobat showed up the ConTEXt backend could easily support similar features these, simply because we already had the infrastructure in place. However, it must be noted that there are two fundamentally different ways to link a source and target: by name or by page, and in both cases the source and target can have an areas that determines the place to click and the place to zoom into. To what extend this is supported depended on the technology and state of that technology as within systems it evolves.

In pdf the main page stream has nothing in it that relates to this, so called (link) annotations and (named) destinations are defined independently and one can think of them sitting on their own layer. They come with area specifications. However, when tagged pdf showed up we suddenly got a relationship with the content stream (pretty much what was the case in dviwindo). So, as part of validation we now suddenly have a boundary condition that these sources and targets are to be related to some structural element. That puts a burdon on a system that uses more sophisticated methods: it has to become more stupid and restricted. It basically means that when you are normally making documents that prvide advanced navigation, read: are accessible already for ages, you now have to operate in a constrained environment.

One way out is to make two documents: an original rich one, and a less advanced conforming accessible one. But does the laws that get imposed on soem users permit that? Do they permits distributing a two documents instead of one, each with a dedicated usage profile?

So, to the linking, consider the next. When we have a section head therefore can be a number and in most cases we have a title. Here we are traditional in the sense that we don't have some nice design, picture or additional stuff.

1.1	Section title

1.1.1	Subsection title

From a tables of content, the traditional view is that number (if used), trhe title (which can be the same or a bit different, depending on needs) and a page number, although that is often irrelevant in an interactive version.

1.1	Section title	12

1.1.1	Subsection title	13

If we talk hyperlinking: clocking on a section number, section head or combination can bring us to the table of contents, or maybe to the next chapter. In the table of contents each of the three element sor the whole can be a link. That is a design decision too!

In tagged pdf the concept is different. First of all, there is no real concept of a section head. There is this `H1` but a distinction between elements is fuzzy at most. Is the number a `Lbl` or maybe a `Link`, but what if it is absent? And does `Link` mean that we should wrap the lot in that one?

In the table of contents it gets weirder. The iso standard gives an example where the title of the table

of contents is a P, so not a real header, maybe because there is no number? Then in the entries the number is a Lbl, the title a Reference, and the page number also a Reference (with a dotted line being a NonStruct and not an Artifact). Actually there are wrappers: a (sub) list is a TOC an the entries (here the threesome) are TOCI's. Now, that is indeed some structure but it leaves aside that somehow Reference is a title or page number.

From that one can actually decide that tagged pdf is not about real structure at all! It's about marking something that might eventually be used somehow. Maybe we should not output an advanced structure but just the simplest possible marking. Because predicting what will be done with it, based on more than a decade of lack of use, is hard. Maybe just make titles Link and Reference and just forget about the rest. Of course one can use these two elements as wrappers but that also sort of gives a weird structure. Compare:

```
<section> <!-- H2 -->
  <sectioncaption>
    <sectionnumber> </sectionnumber>
    <sectiontitle> </sectiontitle>
  </sectioncaption>
  <sectioncontent>
  </sectioncontent>
</section>
```

```
<section> <!-- H2 -->
  <Link>
  <sectioncaption>
    <sectionnumber> </sectionnumber>
    <sectiontitle> </sectiontitle>
  </sectioncaption>
  </Link>
  <sectioncontent>
  </sectioncontent>
</section>
```

```
<section> <!-- H2 -->
  <sectioncaption>
    <sectionnumber> </sectionnumber>
    <Link>
      <sectiontitle> </sectiontitle>
    </Link>
  </sectioncaption>
  <sectioncontent>
  </sectioncontent>
</section>
```

And similarly (lists actually can have various structure but we show one here):

```
<list>
  <listitem>
    <listtag> </listtag>
    <listtext> </listtext>
    <listpage> </listpage>
  </listitem>
</list>
```

with becomes something like this, depending on how one designs the document and interaction:

```
<list> <!-- TOC -->
  <listitem> <!-- TOCI -->
    <listtag> </listtag> <!-- Lbl -->
    <Reference>
      <listtext> </listtext>
    </Reference>
    <Reference>
      <listpage> </listpage>
    </Reference>
  </listitem>
</list>
```

So we can end up with injecting additional Link and Reference elements while actually using attributes makes more sense. Imagine that you have to manipulate such a structure. When you're willing to use arguments that a nested H is complex, I suppose that is easy? In the meantime various patches and clarifications indicate that the two 'toc related' tags are no longer to be used and that list can replace them, so in the end html's use of tables and lists for most objects has infected pdf. This made us decide to ignore these tags. In a similar fashion it seems that BlockQuote fades to the background so we can as well ditch that one too. It's a troublemaker anyway as on the one hand it permits whole sub-documents but has troubles with spans.

Checking your result

The built in mapping is as robust as possible. You can load the basic variant and play with that, maybe copy some and mess around. There is a simple helper that can enlighten you,

```
mtxrun --script pdf --structure --details
                                  --save yourfile
```

will produce an xml file that only serves the purpose of checking. We might or might not extend it and it's not perfect either. But it helped us a bit in figuring out the basic set. The xml tree embeds the xml blobs so that can be a bit confusing. The links and references and such are our way to support at least dumb interaction.

The verapdf checker can be applied with the next command, and we report a summary. You can combine both commands in one call but be aware of the validation taking time.

```
mtxrun --script pdf --validate yourfile
```

If you have an advanced document, you should really

consider making untagged (lean and mean) version and a maybe somewhat well-tuned less demanding version that you tag. Just serve the audience as best as you can.

From the perspective of TEX we have to face reality. This is yet another standard that complicates our live. Where TEX was the first to handle math well, the community had to bow to Microsoft for a first update. We catched up with LuaMetaTEX but that is very much a ConTEXt thing and the rest of the TEX world sticks to what has been there for decades. We saw the same with Unicode where we're now stuck with alphabets with gaps that makes our live miserable and and doesn't really help with accessibility and easy cut'n'paste either. Then there is MathML where the best we can do is catch up with what browsers expect from us and as with tagging often limitations determine the direction. So, last but not least, the TEX community also lost the battle for structure and it shows in pdf tagging. We can do it but it's far from pretty and effective.

As an aside: it makes no sense to tag for instance ConTEXt manuals, if only because the audience is assumed to be able to see and read those manuals. Interesting is that a blind user explained us that he's quite capable of not only dealing with that but also appreciates the (error) messages. This made those who listened to the impressive explanations of the workflow of writing books aware of the fact that we should not underestimate nor try to think solutions that we cannot really experience. In that perspective it is indeed way more interesting to try to make optimized versions for specific audiences.

We should also realize that (speaking in 2025) when it takes more than a decade for a standard to be taken serious and then being adapted in anon upward compatible way, that tools for accessing these documents didn't show up and/or are under development and that rendering elsewhere (also in the TEX community) still seem to be work-in-progress, we're not that much in a hurry. It might be that all those efforts are triggered by regulations but then we're talking chickens and eggs. We'll see how all this evolves. We've seem similar developments with for instance OpenType where we were among the early adopters but standards and fonts also took time to mature. We're not talking science (research and development) here, more business driven evolution. Examples of the former are original TEX, MetaPost and Lua that all were and are pretty much originating in academia and subjected to testing and revision before using and releasing, a more academic approach so to say. They became rather solid standards that way.

When we talk about accessibility tagging is one aspect, another is the text itself. Here Unicode helps out. The text stream effectively is stream of shapes (glyphs) and the font resources map these onto one or more characters. The page stream uses indices in an array with shapes. Likely an 'A' becomes an `A` but an 'fi' ligature expands to an `f` and `i`. The engine turns spaces in the input (document source) into glue so here the backend has to do some juggling to get spaces back (or one at some point can massage the node list to that effect). In the transition from eight bit engines to wide Unicode ones quite some research and effort went into analyzing what glyph names mapped onto what character so that e.g. pdfTEX could inject the right mapping.[9]

Interactivity

In ConTEXt for decades we provided features that permit one to make highly interactive documents and that related pretty much to accessibility. In tagged pdf links, references, destinations, annotations is just broken and it shows in validation: reporting depends on the version of the validation. After spending many hours on testing, discussing, trying to find a workaround, we decided that it makes no sense to cripple the document with a persistent imperfection and also deprive the user from this experience. We no longer support that bit of the standard unless one explicitly enables it with `option=interaction`. In the check one still gets some feedback but details are only provided with `--interaction`.[10]

Interactivity is already tricky enough, given various levels of support in (open source) viewers. It is also already bad enough that some (sub)standards drop handy basic named actions for no obvious reason. We don't want to add more loss to that. If you really want to serve users who have troubles with this you should spend some time on making a targeted accessible version. And when it comes to demands from organizations or governments you can always post an non-interactive version of the document.

Some final words

So why being so critical? Isn't pdf in general nice? Sure when one looks at the rendering all is fine. When one had to adapt to successive interactive features is't less so, as that evolved, stuff came and went, and much is not pretty, But when we're talking of commercial interest and large companies and partners with filled pockets one expects better when it comes to for instance the discussed tagging. So maybe we're just disappointed and should not really expect better. An example: it was really easy to embed audio and video in a pdf file and what better way to serve an audience. But it was replaced by later obsoleted flash and complex media features, that, no surprise, are not supported by open source viewers, in a time when web browsers and other applications can do this well.

Of course one can wonder how history will look back at this. At this moment there are plenty technologies and inventions to look back on, and one can be impressed by what people could achieve. But much got lost and likely for good reasons. We're not talking master pieces here so maybe all will fade away so maybe keep versions of your document around that lack the bloat of tagging. Or even better: spend some time on making a really nice accessible alternative, it being html, pdf or a combination. Best play safe instead of going weird. Make sure that your documents can survive and keep an untagged version alongside.

Figuring out and accommodating all this is no fun. It is somehow expected from free tools like TEX to support all this and it's just because we can compensate dealing with these messes with more interesting endeavors that we can motivate ourselves. The results are just not interesting enough. It's in the end not difficult but it takes way more time than one is willing to spend on it, mainly due to fuzzy and complex standards that keep evolving, limited resource and tools not being ready. But as long as users are happy with the results, even if they are 'the best we can do', we are.

We are lucky that on the mailing list we have patient users who are willing to test, update, test again, and so on, because we ourselves don't have the usage patterns that they have nor are we confronted with tagging demands. So we dedicate all this effort to them! We also expect useful input from visually disabled users and will happily meet their demands! But we'll do so in the context of mixing beauty and usability, preferably with a multiple output solution.

So how are ConTEXt users to proceed? We advise them to just make nice looking well structured documents and forget about tagging. When in the final stage there is demand, a first attempt can be:

```
\setuptagging[state=start]
```

This will permits advanced (llm driven) search machines to pick up the structure. When it's really needed one can try:

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

and hope for the best. The possibility to pass a `preset` of mappings can improve matters. Have a look at the sample document in the ConTEXt distribution. And use the `mtxrun --script pdf` tool with the `--structure` or `--check` option.

In any case post the non tagged version, maybe the just tagged one, and then the officially tagged variant. There is no reason to pollute the future with uselessly tagged bloated documents when computers can figure it out anyway. It will likely take years before we know how this all works out.

Hans Hagen

1. See TugBoat Volume 31 (2010), Tagged PDF in ConTEXt, Hans Hagen, https://tug.org/TUGboat/tb31-3/tb99hagen.pdf
2. Publishers out source conversion from TEX input to for instance pdf and those doing it are basically just consumers of open sourced engines and macro packages. Some might add own technology but that is then mostly a secret. They adapt to tagged pdf as requirement. It's a sort of closed usage of open source.
3. Here demands mostly come from for instance educational and maybe governments. We can't speak for hidden commercial usage, for instance in workflows.
4. We use the original names here, including capitalization and so.
5. It looks like for instance Okular has some JavaScript support but relatively simple code that works in Acrobat fails there, but I keep an eye on it.
6. These standards don't come cheap and mention a price based on the number of pages. It might be why these tag tables are so inefficiently typeset.
7. At the 2025 ConTEXt meeting we had presentations and discussion about tagged pdf and how to proceed. Afterwards someone pointed me to https://netz-barrierefrei.de/en/awful-pdf.html which pretty much sums up the situation we're in: The awful Accessibility of pdf Documents, (1) pdf is not designed for accessibility, (2) Semantics must be grafted on, (3) Too expensive, too complicated, too closed, (4) Accessible pdf is a money-burning machine, (5) Adobe has done too little, (6) pdf-UA – a closed society, (7) Adobe is to blame, (8) The failure of the pdf Association, (9) pdf is a historical misstep; one can agree or disagree on the various points when one can use TEX and friends but there is a truth on some of this. But let's leave politics aside here.
8. Some of these primitives are specific to LuaMetaTEX.
9. When tagged pdf showed up Ross Moore became the leading LATEX expert on this matter and tagging in general. At some point we discussed this and ran and analyzed thousands of fonts just to find out that there is a lot of variation. It was an interesting and sort of fun experience. At meetings he also reported regularly on experiences with tagging math, which has it own quirks.
10. We found some bug reports on the web but answers to that were hardly promising and concepts and interpretations seem to evolve. Not much of a standard.