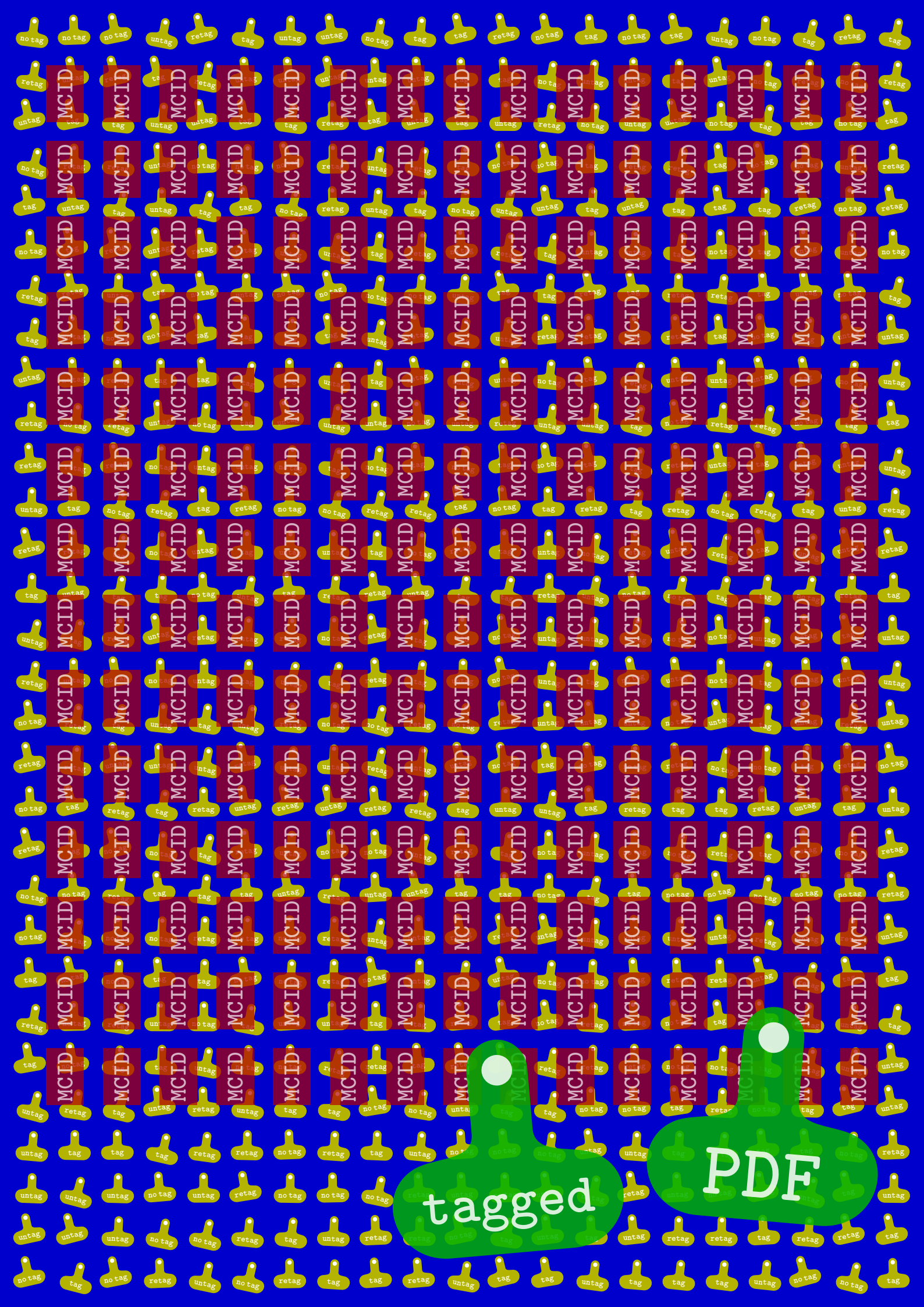
MCID
tag
no tag
retag
untag
tagged
PDF

1 Why do we tag

Around 2010 tagged pdf showed up in ConTEXt. Apart from demonstrating that it could be done it served little purpose because only full Acrobat could show a structure tree and in the more than a decade afterwards no other viewer did something with it. However for some users it was a necessity.

In 2024 we picked up on tagging because due to regulations (especially in higher education) demands for tagged pdf in the perspective of accessibility popped up. We will not go into details here but just mention that we want to make sure that users can meet these demands.

As of now (2024) we have little expectations when it comes to tagging. The ongoing discussions about how to tag, how to interpret the specification, what to validate, and what to expect from applications are likely to go on for a while, so the best we can do is keep an eye on it and adapt when needed. If we have opinions, these will be exposed in other documents (and articles).

We can also notice that the standard is less standard as things change, part as side effect of clarification (which tells us something) but also because it looks like some applications have problems with it. Working on this is disappointing and dissatisfying, but often we have a good laugh about this mess, so we try to keep up (adapt) anyway.

Hans Hagen
Mikael Sundqvist

2 Tagging text

As mentioned in the introduction, we need to satisfy validators that are imposed on those working in education (often via web interfaces with little information on what actually gets checked, it's business after all). It is not that hard to fool them and make documents compliant, so that is what we can do anyway. It is also possible to let these tools do some auto tagging but our experiments showed that this is a disaster. So, we end up with a mix of relatively rich tagging that we feel good with. When we're a decade down the road we expect that with a little help from large language models a decent verbose tagging is better than a crappy suboptimal one.

One reason for tagging is that it could permit extraction but there are better solutions to that: if there is something shown in a table or graphic, why not add the dataset. We currently add MathML and bibTEX blobs but more can become possible in the future (this also depends on user demand).

Another application is reflow but when that is needed, why not go html or distribute different output. When accessibility is the target one has to wait till more is clear how that is actually supposed to work. Often the recommendations are to use Arial, little color, simple sectioning etc, so that gives little reason to use pdf at all.

All that said, we assume that pdf level 2 is used, if only because it looks like validators aim for that. Also, if you find pre level 2 documents produced elsewhere, often tagging is so bad or weird that one can as well ignore it.

You should realize that tagging is actually rather old, it was already available in the early version. You can enable tagging with:

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

This will give you basic tagging: directives in the page stream combined with a structure tree. You can also decide to add mapping to pdf specific tags. This is done with:

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

The first command ensures that the right data ends up in the pdf file, and the second one enables tagging. As long as you're working on a document you can comment these commands which saves you some runtime and give way smaller files.

We don't want to cripple proper structure ConTEXt support by the limitations introduced in pdf version 2, but we do offer users some control, as long as it does not backfire. Due to the fluid situation (around 2024) we delegate some choices to the user.

By default we use robust mapping (read: not sensitive for limitations in nesting pdf specific tags cf. checkers) but you can say this:

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

The basic mapping is defined in `lpdf-tag-imp-basic*` files and get and of course define your own. The basic set works for a few complex documents we tried but has to work around limitations. We have no clue how that works out in the long term because (as mentioned) validation, standardization and whatever relates are fluid. We still provide:

```
\setuptagging [preset=mkiv]
```

you get the mapping used in MkIV but that one fails level 2 validation. In due time we might explain (re)mapping here in more detail. There are various features controlled by directives but these mostly serve(d) testing.

3 Interactivity

By default we ignore references and links in the mapping. This because we found it to be confusing, unreliable, limited and (some searching indicated) open to interpretations and changes. It doesn't bring anything to the table anyway. In ConTEXt users can make advanced interactive documents; it was always a strong point and for for some users a reason to use this system. If you really want it to be taken into account, you can do this:

```
\setuptagging [option=interactivity]
```

You might need to change your document if validation fails, for instance because you add buttons and such in constructs that no one could envisage.

4 Tagging math

Tagging math at level 2 is still experimental but works as follows. Instead of tagging the atoms and structures, as we do in level 1, we generate a MathML attachment and put a so called actual text on the math structure node. This text can be spoken by reading machinery. The MathML is not that rich but we can enable more detail when needed. However, given the way (presentational) MathML evolved we are somewhat pessimistic. Instead of adding a few more elements that would help to provide structure, some features are dropped. Also, support in browsers comes and goes, either native or depending on JavaScript.

Because there is much freedom in how mathematical symbols and constructs are used, you might need to help math tagging bit. The process is driven by group sets that refer to domains. An example of a domain is chemistry. For now we just mention that this features is there and as time flies by we can expect more granular usage.

```
\definemathgroupset
  [mydomain]
  [every] % a list of dictionaries

\setmathgroupset
  [mydomain]
```

For now you can ignore these commands because we default to `every`.

Todo: list all possible dictionaries.

You can control the tagger by specifying what symbols and characters actually mean, for instance:

```
\registermathfunction[]
\registermathfunction[]

% \registermathsymbol[default][en][][the vector]
% \registermathsymbol[default][en][][the vector]
% \registermathsymbol[default][en][][the matrix]

\registermathsymbol[default][en][lowercasebold]           [the
  vector] % [of]
\registermathsymbol[default][en][uppercasesansserifnormal][the
  matrix]
```

From the language tag being used here you can deduce that this can be done per language.

You can trace math translations with:

```
\setupnote[mathnote][location=page]
\enabletrackers[math.textblobs]
```

which is what we used when developing these features. In a few hundred page math book one easily gets thousands of notes.

In `examples-mathmeanings` you can find a lot of examples. In due time we expect to offer more translations. The English and Swedish are for now the benchmark.[1] Likely other languages will be served by Tomáš Hala as result of courses on typesetting. Feel free to contact all those involved in this.

[1] As a proof of concept, at BachoTEX 2024, the Ukrain translations were provided by Team Odessa, but they need some tuning.

5 Structure

Although today all goes to pdf, that is not what TEX macro packages started with. Basically they just use the TEX engine to render something in the tradition of printing but using a target format that can be converted to something that a printer understands. Nowadays that just happens to be pdf.

So, although the target is pdf, that doesn't mean that pdf drives (or should drive) the process. If we want what is called tagged pdf where tagging represents structure, one could argue that this is then a follow up on whatever structure users used in the process. In ConTEXt we start from the TEX input end, not from some tagging related pdf wish list which could handicap us. So called tagged pdf is not the objective, it is just a possible byproduct.

Keep in mind that tagging related to structure serves a few purposes: reflow, conversion, and accessibility. We're not at all interested in reflow of pdf, because is that a which one should just produce html. We're also not interested in conversion because, again, one could just use a different workflow, maybe one that starts from xml and can target different media. When it comes to accessibility this mixed bag contains options like generating different versions, each tuned to a specific target audience. Typesetting is about generating some visual representation and just like people have different food preferences, one can imagine different representations: there is no reason to only produce pdf. And even if there are ways to help something rendered for printing, or reading on screen, for instance by providing audio, there is no need to do that for very complex documents. Given the often poor quality of simple TEX documents one can even wonder if that tool should be used at al then. It's not like TEX is the only system that can do math nowadays.

When we look at structure, this is how ConTEXt sees a section:

```
\startchapter[title={This is a chapter.}
  \startsection[title={This is a section.}
    Some text here.
  \stopsection
\stopchapter
```

In xml that could be something like this with the number being optional as it can be generated:

```
<section detail="chapter">
  <sectioncaption>
    <sectionnumber>1</sectionnumber>
```

```
    <sectiontitle>This is a chapter.</sectiontitle>
  </sectioncaption>
  <sectioncontent>
    <section detail="section">
      <sectioncaption>
        <sectionnumber>1.2</sectionnumber>
        <sectiontitle>This is a section.</sectiontitle>
      </sectioncaption>
      <sectioncontent>
        Some text here.
      </sectioncontent>
    </section>
  </sectioncontent>
</section>
```

In a pdf there can also be additional rendered material, like headers and footers and maybe the section title is rendered in a special way but we ignore that for now.

When I comes to the content blob, we have to look at the TEX end. User input normally will give this:

```
<sectioncontent>
  A first paragraph.

  A second paragraph.
</sectioncontent>
```

An empty line starts a paragraph but it can also be explicitly forced (think \par).

```
<sectioncontent>
  A first paragraph.
  <break/>
  A second paragraph.
</sectioncontent>
```

But one can also explicit encode paragraphs and then get:

```
<sectioncontent>
  <paragraph>A first paragraph.</paragraph>
  <paragraph>A second paragraph.</paragraph>
</sectioncontent>
```

which in TEX speak is:

```
\startchapter[title={This is a chapter.}
  \startsection[title={This is a section.}
    \startparagraph A first paragraph.  \stopparagraph
    \startparagraph A second paragraph. \stopparagraph
  \stopsection
\stopchapter
```

But it will be clear that not all users want to do that, which means that we end up with the <break/> variant. Now you can ask, why not infer this extra level of structure and the answer is: it's not how TEX works. The content can be anything and the fact that there is no real clear solution is actually reflected in how pdf tagging maps onto pseudo html elements: not all can nest, so for instance a paragraph cannot contain a paragraph. That means that we cannot reliable add that level of structure automatically as it limits the degrees of freedom that users have. As mentioned: tagging in pdf is not the starting point, just a possible byproduct.

Let's look at another structure element:

```
\startitemize
  \startitem A first item. \stopitem
  \startitem
    A second item.
    \startitemize
      \startitem Again first item. \stopitem
      \startitem And a last one. \stopitem
    \stopitemize
  \stopitem
  \startitem A third item. \stopitem
\stopitemize
```

If we start from input, we can use this kind of xml:

```
<itemize>
  <item>A first item.</item>
  <item>
    A second item.
    <itemize>
      <item>Again a first item.</item>
      <item>And a last one.</item>
    </itemize>
  </item>
  <item>A third item.</item>
</itemize>
```

But once we're done, we actually have something typeset, so we end up with more detail:

```
<itemgroup>
  <item>
    <itemtag>1.</itemtag>
    <itemcontent>
      <itemhead/>
      <itembody>A first item.</itembody>
    </itemcontent>
  </item>
  <item>
    <itemtag>2.</itemtag>
    <itemcontent>
      <itemhead/>
      <itembody>
        A second item.
        <itemgroup>
          <item>
            <itemtag>a.</itemtag>
            <itemcontent>
               <itemhead/>
               <itembody>Again a first item.</itembody>
            </itemcontent>
          </item>
          <item>
            <itemtag>b.</itemtag>
            <itemcontent>
              <itemhead/>
              <itembody>And a last one.</itembody>
            </itemcontent>
          </item>
        </itemgroup>
      </itembody>
    </itemcontent>
  </item>
  <itemtag>3.</itemtag>
  <itemcontent>
    <itemhead/>
    <itembody>
      A third item.
    </itembody>
```

```
  </itemcontent>
</itemgroup>
```

This represents what gets rendered but one can leave out the tag and let whatever interprets this deal with that.

So, we have input that can be either explicit (given numbers and tags) or implicit (the system generates them) but output can also be explicit or implicit. And when output carries structure the question is: do we want to preserve abstraction or do we want the rendered results. It only makes sense to invest in this when it pays off, also because the resulting pdf file get bloated a lot.

6 PDF

The specification (say the second edition of 2020) has a section about tagged pdf in the perspective of reflow, conversion and accessibility. As we mentioned already the lack of tools using any of this didn't help much in clarifying all this. At some point it became possible to verify a pdf file and some rudimentary converters popped up but it looks like everyone had to interpret the rules laid out in the specification. Mid 2024 we also had numerous errata and/or clarifications of the specification (not only tagging) but (at least for us) it is not clear what criteria for changes (more restrictions) were.

For instance, the ISO 32000-2 specification explicitly mentions the usefulness of the `H` mapping for classes of documents. But in ISO 14289-2:2024 we can read "The `H` structure type requires processors to track section depth, which adds an unnecessary burden on processors and can cause ambiguity." This is a quite baffling remark given the complexity of pdf and web technologies in general. So in the end it was dropped in order to make some developers lives easier, while complicating that of others.

It is this, and other vague (and changing) descriptions, take for instance `Part` and `Sect`, that made us decide to draw a line. We tried some in our opinion reasonable variants but could never satisfy the validators completely. Keep in mind that we started from tagging everything, not just the easy bits and pieces and then wrapping whatever left in artifacts or a wildcard paragraph. When we looked at tagging with 2024 glasses on we were willing to adapt but in the end it makes little sense. We can add a few mappings but in general it is too conflicting with our approach to structure, one that goes back decades. So, as long as a (audio) reader can do a reasonable job, we're okay. There are more interesting challenges on our plates anyway.

7 Tracing

There are a few trackers that relate to tagging but these are more for ourselves so we just mention them:

```
\enabletrackers[structures.tags]
\enabletrackers[structures.tags.info]
\enabletrackers[structures.tags.math]
\enabletrackers[structures.tags.blobs]
\enabletrackers[structures.tags.internals]
\enabletrackers[structures.tags.suspects]
\enabletrackers[structures.tags.attribute]
\enabletrackers[structures.tags.paragraphs]
\enabletrackers[structures.tags.objects]
```

The first few are probably the most useful as it shows how ConTEXt sees the structure of your document. Others are there because they help figuring out issues.

We also have a script that can help checking:

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

This will use (currently only) verapdf to check the file and show a summary. Another feature is:

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

and both can be done in one step with:

```
mtxrun --script pdf --check yourfile
```

When you tag a document, best do that as late as possible because it adds runtime and checking doesn't really help much when writing something. You might want to put the setup in a mode so that you can enable it from the command line. The documents also become way bigger. And, if you really care about accessibility, consider making two versions: one that suits your typographic needs, and one that targets the audience that needs accessibility. And, for printed documents, you can just ignore it.