minim-pdf

version 2023/1.3

author Esger Renkema
contact minim@elrenkema.nl

This package adds low-level support to plain luatex for marking up the structure of a pdf document. The implementation is rather basic, but should allow you to make your pdfs fully pdf/a-compliant. Load the package by saying `\input minim-pdf`.

The creation of tagged pdf will be described in the second half of this manual; all other features will be covered first.

Hyperlinks

For most simple cases, you can use `\hyperlink [name {...} | url {...}] ... \endlink` for linking to a named destination in your own document or to an external hyperlink respectively. There is no support for nonsimple cases.

A named destination can be created with `\nameddestination {...}` (also in horizontal mode, unlike the backend primitive) and if you cannot think of a name, `\newdestinationname` should generate a unique one. If you need the latter twice, `\lastdestinationname` gives the last generated name.

Bookmarks

Bookmarks can be added with `\outline [open|closed] [dest {name}] {title}`. Add `open` or `closed` to have the bookmark appear initially open or closed (default), and say `dest {name}` for having it refer to a specific named destination (otherwise, a new one will be created where the `\outline` command appears).

A bookmark is automatically associated with the current structure element and the hierarchy of structure elements determines the nesting of bookmarks. Therefore, if you want nested bookmarks, you *must* precede the `\outline` command with a declaration of the current structure element, even if you have otherwise disabled tagging. See the next chapter on how to do this.

Page labels

If the page numbers of your document are not a simple sequence starting with 1, you can use `\setpagelabels [pre {prefix}] style nr` for communicating this to the pdf viewer. This command affects the page labels from the next page on: `nr` should be the numerical page number of that page. The `prefix` is prepended to each number and the `style` must be one of `decimal`, `roman`, `Roman`, `alphabetic`, `Alphabetic` or `none`. In the last case, only the prefix is used.

PDF/A

You can declare pdf/a conformance with `\pdfalevel xy`, with version `x` $\in \{1, 2, 3\}$ and conformance level `y` $\in \{a, b, u\}$. This will set the correct pdf version and `pdfaid` metadata. If the conformance level is '`a`', tagging will be enabled (see the next chapter). Finally, a default RGB colour profile will be included. The conformance level can be queried from the `\pdfaconformancelevel` register.

Note that merely declaring conformance will not make your document pdf/a compliant, and that minim will not warn you if it is not. However, the features described in this chapter and the next should be enough to make pdf/a compliance possible.

Also note that there currently is no documented way of choosing a different colour profile from the default (i.e. the default rgb profile provided by the colorprofiles package). Should you need do that, you will have to do so manually, after disabling the automatic inclusion by saying `\expandafter\let \csname minim:default:rgb:profile\endcsname = \relax`.

Finally, note that pdf/a requires that spaces are represented by actual space characters and that discretionary hyphens are marked as soft hyphens (`U+00AD`). Since both features benefit accessibility and text extraction in general, they are enabled by default. You can disable them by setting `\writehyphensandspaces` to a nonpositive value.

Embedded files

You can attach (associate) files with `\embedfile <options>`. The file will be attached to the current structure element (see the next chapter) unless the `global` option is given: then it will be added to the document catalog. Arguments consisting of a single word can be given without braces and exactly one of the options `file` or `string` must be present.

`file`	`{...}`	The file to embed.
`string`	`{...}`	The string to embed.
`global`		Attach to the document catalog.
`uncompressed`		Do not compress the file stream.
`mimetype`	`{...}`	The file's mime type.
`moddate`	`{...}`*	The modification date (see * below).
`desc`	`{...}`	A description (the `/Desc` key).
`relation`	`{...}`	The `/AFRelationship` value as defined in pdf/a-3.
`name`	`{...}`	The file name (only required when writing a `string`).

* The modification date must be of the form `yyyy[-m[m][-d[d]]]`. A default moddate can be set with `\setembeddedfilesmoddate {default}`. The `default` date will be expanded fully at the time of embedding. With the minim-xmp package, a useful setting is `\setembeddedfilesmoddate {\getmetadata date}`.

Lua module

The interface of the lua module (available via `local M = require('minim-pdf')`) should be stable by now. Though it contains lua equivalents for most tex commands described here, using them directly is not very ergonomical and not recommended. Please consult the source if you do want to use them anyway.

Tagged PDF

This chapter is a continuation of the previous and describes the parts of minimpdf that concern the creation of tagged pdf. All features in this chapter must be explicitly enabled by setting `\writedocumentstructure` to a positive value. This will be done automatically if you declare pdf/a conformance (see above).

This part of the package is rather low-level and this chapter rather technical. For a more general introduction to and discussion of tagged pdf, please read the (excellent) manual of latex's tagpdf package.

Purpose, limitations and pitfalls

The main purpose of this package is semi-automatically marking up the (hierarchical) structure of your document, thereby creating so-called tagged pdf. The mechanism presented here is not quite as versatile as the pdf format allows. The most important restriction is that all content of the document must be seen by tex's stomach in the *logical* order.

Furthermore, while the macros in this package are sophisticated enough that tagging can be done without any manual intervantion, it is quite possible and rather easy to generate the wrong document structure, or even cause syntax errors in the resulting pdf code. You should always check the result in an external application.

This is the full list of limitations, pitfalls and shortcomings:

1. Document content must be seen by tex in its logical order (although you can mark out-of-order content explicitly if you know what you are doing; see below).
2. The contents of `\localleftbox` and `\localrightbox` must be marked manually, probably as artifact.
3. You must mark page header, page footer and footnote rule yourself; no default is set.
4. There currently is no way of marking xforms or other pdf objects as content items of themselves.
5. The content of xforms (i.e. pdf objects created by `\useboxresource`) should not contain tagging commands.
6. Likewise, you should be careful with box reuse: it might work, but you should check.
7. This package currently only supports pdf 1.7 tagging and is not yet ready for use with pdf 2.0.

In order to help you debugging, some errors will refer you to the resulting pdf file. If you get such errors, decompress the pdf and search for the string 'Warning:'. It will appear in the pdf stream at the exact spot the problem occurs.

General overview

When speaking about tagging, we have to do with two (or perhaps three) separate and orthogonal tagging processes. The first is the creation of a hierarchical *document structure*, made up of *structure elements* (SEs). The document structure describes the logical structure of a document, made up of chapters, paragraphs, references etc. The second tagging process is the tagging of *marked content items* (MCIs): this is the partition of the actual page contents into

(disjoint) blocks that can be assigned to the proper structure element. Finally, as a separate process, some parts of the page can be marked as *artifacts*, excluding their content from both content and structure tagging.

When using this package, artifacts and structure elements (excluding paragraphs; see below) must be marked explicitly, while marked content items will be created, marked and assigned automatically. There is some (partial and optional) logic for automatically arranging structure elements in their correct hierarchical relation.

The mechanism through which this is achieved uses attributes and whatsits for marking the contents and borders of SEs, MCIs and artifacts. At the end of the output routine, just before the pdf page is assembled, this information will be converted into markers inserted in the pdf stream.

Marked content items

Content items are automatically delineated at page, artifact and structure element boundaries and terminated at paragraph or display skips. This should relieve you from any manual intervention. However, if you run into problems, the commands below might be helpful.

Use of `ActualText`, `Alt` or `Lang` attribute on MCIs, while allowed by the pdf standard, is not supported by this package. You should set these on the structure element instead.

The begining and ending of a content item can be forced with `\startcontentitem` and `\stopcontentitem`, while `\ensurecontentitem` will only open a new content item if you are currently outside any. If you need some part to be a single content item, you can use `\startsinglecontentitem ... \stopsinglecontentitem`. This will disable all SE and MCI tagging inside.

Tagging (both of MCIs and SEs) can be disabled and re-enabled locally with `\stoptagging` and `\starttagging`.

Artifacts

Artifacts can be marked in two ways: with `\markartifact {type} {...}` or with `\startartifact {type} ... \stopartifact`. The `type` is written to the pdf attribute dictionary directly, so that if you need a subtype, you can write e.g. `\startartifact {Pagination /Subtype/Header} etc`.

Inside artifacts, other structure content markers will be ignored. Furthermore, this package makes sure artifacts are never part of marked content items, automatically closing and re-opening content items before and after the artifact. While the pdf standard does not require the latter, not enforcing this seems to confuse some pdf software.

Document structure

Like artifacts, structure elements can be given as `\markelement {Tag} {...}` or `\startelement {Tag} ... \stopelement {Tag}`. Here, in many cases the `\stopelement` is optional: whenever opening an element would cause a nesting of incompatible `Tag`s, the current element will be closed until such a nesting is possible. Thus, opening a `TR` will close the previous `TR`, opening an `H1` will automatically close any open inline or block structure elements, opening a `TOCI` will close all elements up until the current `TOC` etc. etc.

As a special case, the tags `Document`, `Part`, `Art`, `Sect` and `Div` (and their aliases) will try and close all open structure elements up to and including the last structure element with the same tag. (An alias will of course only match the same alias.)

While the above can greatly reduce the effort of tagging, the logic is neither perfect nor complete. You should always check the results in an external application. Particular care should be taken when 'skipping' structure levels: the sequence chapter – subsection – section will result in the section beneath the subsection. If you are in doubt whether an element has been closed already, you can use `\ensurestopelement {Tag}` instead of `\stopelement` to prevent an error being raised.

All these helpful features can be disabled by setting `\strictstructuretagging` to a positive value. Then, every structure element will have to be closed by an explicit closing tag, as in xml. In this case, `\stopelement` and `\ensurestopelement` will be equivalent.

By default, `P` structure elements are inserted automatically at the start of every paragraph. The tag can be changed with `\nextpartag {Tag}`; leaving the argument empty will prevent marking the next paragraph. Keep in mind that the reassignment is local: if a paragraph marked with `\nextpartag` starts inside a group, it will not reset. Auto-marking paragraphs can be (locally) disabled or enabled by saying `\markparagraphsfalse` or `\markparagraphstrue`.

You can query the place in the document structure of any point with `\showdocumentstructure`.

Structure element aliases

New structure element tags can be created with `\addstructuretype [options] Existing Alias`. This will create a new structure tag named `Alias` with the same properties as `Existing`. The properties can be modified by specifying `options`: these will set values of the corresponding entry in the `structure_types` table (see the lua source file for this package). Any aliases you declare will be written to the pdf's `RoleMap` only if they have actually been used.

Manipulating the logical order

With the process outlined above, the logical order of structure elements has to coincide with the order in which the SEs are 'digested' by tex. This, together with the marked content items being assigned to structure elements in their order of appearance, lies behind the restriction that logical and processing orders should match.

With manual intervention, this restriction can be relaxed somewhat. Issuing the pair `\savecurrentelement ... \continueelement` will append the MCIs following `\continueelement` to the SE containing `\savecurrentelement`. Since the assignments made here are global, this process cannot be nested; in more complicated situations you should therefore use `\savecurrentelementto\name ... \continueelementfrom\name` which restores the current SE from a named identifier `\name`.

Structure element options

The `\startelement` command allows a few options that are not mentioned above: its full syntax is `\startelement <options> {Tag}`. The three most useful options are `alt` for setting an alt-text (the `/Alt` entry in the structure element dicionary), `actual` for a text replacement (`/ActualText`) and `lang` for the language (`/Lang`; see the next section). The alternative and actual texts can also be given after the fact with `\setalttext {...}` and `\setactualtext {...}`; these apply to the current structure element.

Structure element attributes can be given with `attr <owner> <key> <value>`, e.g. `attr Layout Placement /Inline` or added later with `\tagattribute`. Note that for the `owner` and `key` the initial slash must be omitted; the `value` on the other hand will be written to the pdf verbatim. Any number of attributes can be added.

An identifier can be set with the `id` option, or after the fact with `\settagid {...}`. This identifier will be added to the `IDTree` and is entirely optional; you will probably already know when you need it.

Finally, structure element classes can be given with the `class <classname>` keyword, which can be repeated. Classes can be defined with `\newattribute-class classname <attributes>` where `<attributes>` can be any number of `attr` statements as above.

Languages

If you do not specify a language code for a structure element, its language will be determined automatically. In order for this to work, you must associate a language code to every used language; you can do so with `\setlanguagecode name code`, where `name` must be an identifier used with `\uselanguage {name}` and `code` must be a two or three-letter language code, optionally followed by a dialect specification, a country code, and/or some other tag. Note that the language code is associated to a language *name*, not to the numerical value of the `\language` parameter. This allows you to assign separate codes to dialects.

There is a small set of default language code associations, which can be found in the file `minim-languagecodes.lua`. It covers most languages defined by the hyph-utf8 package, as well as (due to their ubiquitous use) some ancient languages.

An actual language change introduced by `\uselanguage` will not otherwise be acted upon by this package. Therefore, you will probably want to add `\startelement{Span}` after every in-line invocation of `\uselanguage`.

You can set the document language with `\setdocumentlanguage language-code`. If unset, the language code associated with the first `\uselanguage` statement will be used, or else `und` (undetermined). The only function of the document language is that it is mentioned in the pdf catalog: it has no other influence.

New languages can be declared with `\newnamedlanguage {name} {lhm} {rhm}` and new dialects with `\newnameddialect {language name} {dialect name}`. Dialects will use the same hyphenation patterns (and will indeed have the same `\language` value) as their parent languages; newly declared languages will start with no hyphenation patterns. Do note that you will probably also have to specify language codes for new languages or dialects.

This package ensures the existence of the `nohyph`, `nolang`, `uncoded` and `undetermined` dummy languages, all without hyphenation.

Mathematics

You can auto-tag equations as formulas by specifing `\autotagformulas`. After this command, auto-tagging can be switched off and on with `\stopformulatagging` and `\startformulatagging`. Auto-tagging formulas is dangerous, because sometimes equations are used for lay-out and should not be marked as such. It is also somewhat fragile, as it requires equations to end with dollar signs (and not with `\Ustopmath` or `\Ustopdisplaymath`).

The tex source of an equation can be associated with the `Formula` structure element in various ways, which can be configured with `\includeformulasources {options}`, where the `options` must be a comma-separated list of `alttext`, `actualtext` or `attachment`. The `alttext` and `actualtext` option will set the `/Alt` or `/ActualText` attributes to the unexpanded source code of the equation, surrounded by the appropriate number of dollar signs. The `attachment` option attaches the source of the formula as an embedded file with its `/AFRelation` set to `Source`; this will only work if `\pdfaconformancelevel` equals three. The name of this file can be changed by redefining `\formulafilename` inside the equation. The default value is `{actualtext,attachment}`.

Note that the contents of the equation will be expanded fully (as in `\xdef`) before their inclusion as the equation source. This may place restrictions on the macros you want to use (those in minim-math should be safe). Any occurrence of `\alttext` or `\actualtext` overrides the automatically-assigned value and will be stripped from the equation source.

Tables

For marking up tables, a whole array of helper macros is available. First, `\marktable` should be given *before* the `\halign`. Then, in the template, the first cell should start with `\marktablerow \marktablecell` and each subsequent cell with `\marktablecell`. If your table starts with a header, insert `\marktableheader` before it and `\marktablebody` after. Before a table footer, insert `\marktablefooter`.

For greater convenience, insert just `\automarktable` before the `\halign`. Then you can leave out all the above commands (unless you `\omit` a template of course). This assumes the table has a single header row and more than one column. If you use a table for typesetting a list, you can use `\marktableaslist` instead, which marks the first column as list label and the second column as list item. Of course, this only works with two-column tables.

Cells spanning multiple cells or rows can be marked with `\markcolumnspan {width}` and `\markrowspan {height}`; these statements may not occur before `\marktablecell`. Note that while `\markcolumnspan` properly increases the (internal) column number, `\markcolumnrow` does nothing of the sort (and indeed, no general logic can be given in the latter case). Always proceed with caution when using cells spanning multiple rows, and inspect the resulting structure carefully.

Marking a table header (either manually or with `\automarktable`) will not connect normal cells with their headers; you will have to connect these manually by including `\markcolumnhead` or `\markrowhead` in the appropriate header cells. This must be done *after* `\markcolumnspan` if the latter appplies. If properly

setup like this, other cells of the table (including header cells) will be assigned to matching row or column headers automatically.

Other helper macros

For marking up an entry in a table of contents, you can use the macro `\markertocentry {dest} {lbl} {title} {filler} {pageno}`, which should insert all tags in the correct way. (The `dest` is a link destination and can be empty; the `lbl` is a section number and can also be empty.)

Licence