Accessible PDFs from LaTeX

Hunter Lehmann

January 9, 2026

Contents

1 Introduction

This document is intended to be a basic reference for creating accessible PDFs from LaTeX . The accessibility of PDFs, both those created by LaTeX and others, has been an issue for many years. Recent updates to the PDF standard have allowed for much better support for accessibility features, especially for mathematics. This is also especially important given the new rule on web content accessibility that comes into effect in April 2026 (see here). Most basic documents (e.g. course notes, worksheets, quizzes) can now be automatically tagged and made fully screen-reader accessible with minimal effort. More complex documents (e.g. exams, slides) need more work but are also doable. Examples of all kinds are included in this project.

2 Basics

To enable the tagging features, you need to add the following to your document preamble:

```
    \DocumentMetadata{
 lang        = en-US,
 pdfstandard = {ua-2, a-4f},
 tagging=on,
 tagging-setup={math/setup={mathml-SE,mathml-AF}}
}
```

This requires a 2025 version of the TeXLive installation or an up-to-date MikTeX installation. On September 4, 2025, Overleaf's LaTeX kernel was updated to support this, removing the need to work locally for most projects.

Basic tagging works with either `pdflatex` or `lualatex` compilation, but to handle mathematics properly, you need to use `lualatex` to compile and use the `unicode-math` package. The `mathml-SE` option creates MathML as Structure Elements, which works well with Adobe Acrobat and NVDA or JAWS with the MathCat add-on. The `mathml-AF` option creates MathML as Associated Files, which works well with Foxit Reader and NVDA or JAWS with the MathCat add-on.

Once this preamble is added, most parts of standard documents will work as is. The two major exceptions are images and tables. Images need to have alt text added or be marked as artifacts using the new `alt={...}` option to the `\includegraphics` command. Tables need to have their header rows marked using the `\tagpdfsetup{\table/header-rows={X}}` command. The `enumitem` package is not compatible, but most of its features are now implemented by default (e.g. the ability to start at a particular number, resume the count in a new list, use roman or alphabetical or arabic number - see the documentation links in Section 5.

Currently the `article` documentclass is well supported, but `exam` is currently incompatible and `beamer` will not be supported. To create quiz or exam documents, you will need to rebuild some of the structures from `exam` yourself. To create slides, you will need to transition to the newly under development `ltx-talk` documentclass. This shares some syntax with beamer, and has some basic documentation available. At present this class requires the November 2025 release of LaTeX to compile, so requires custom set up to be accessed in Overleaf. I strongly recommend working with a local TeX installation for this. More information about currently supported document classes and packages can be found in this table.

3 Implementation

In this section, I will outline how to use some common features of LaTeX to create accessible documents, focusing on places where either there are modifications from standard usage or it is important to make careful choices to ensure accessibility.

3.1 Graphics

All images included in the document need to either have alt text added or be marked as artifacts/decorative images. Each instance of an `\includegraphics` command or `\begin{tikzpicture}...\end{tikzpicture}` environment needs to be handled. To add alt text to one of these images, include the `alt={text}` optional argument to the command or environment. For example, if I include a graphic of a sine wave using the file `sinewave.png`, I might write:

```
\includegraphics[alt={A graph of the sine function from 0 to 2 pi}]{sinewave.png}
```

If the image is purely decorative and does not add any information to the document, you can mark it as an artifact by using the `artifact` optional argument. For example, if I have a decorative border image called `border.png`, I would write:

```
\includegraphics[artifact]{border.png}
```

When providing alt text, consider the purpose of the image and not just its content. Two resources for writing effective alt text are WebAIM's Alt Text Techniques and Nielsen Norman Group's Alt Text: What to Write.

3.2 Tables

For a table to be accessible, screen readers need to know which rows are header rows and which columns are header columns. You can set the header rows by invoking the command `\tagpdfsetup{table/header-rows={X,Y,Z}}`, where `X,Y,Z` is a comma-separated list of the row numbers that are header rows, starting from 1. This applies to all following tables until it is changed again. For example, if I have a table where the first two rows are header rows, I would write:

```
\tagpdfsetup{table/header-rows={1,2}}
\begin{tabular}{...}
...
\end{tabular}
```

To set a default for all tables in the document, place this command in the document preamble. To set header columns, use the command `\tagpdfsetup{table/header-columns={X,Y,Z}}` in the same way. Giving negative arguments will count rows/columns from the end of the table.

Sometimes tabular environments are used solely to control layout instead of presenting data. In that case, you should mark the table as a 'presentation table' to prevent screen readers from interpreting it as a data table. You can do this by adding the command `\tagpdfsetup{table/tagging=presentation}` immediately before the `tabular` environment.

The `tabularx` package that extends the functionality of table environments is also compatible with the tagging features and can be used in the same way as described above. Currently only `tabular, tabularx,tabular*,longtable` are supported.

3.3 Lists

The standard list environments of `itemize` and `enumerate` are fully supported. The extensions provided by the `enumitem` package are not compatible, but many of its features are now implemented by default using a key-value interface. A few of those keys are shown below, but for a full list, see the documentation links in Section 5.

To start an `enumerate` list at a specific number, use the `start=X` key, where `X` is the desired starting number. For example, to start a list at 5, you would write:

```
\begin{enumerate}[start=5]
\item First item, but numbered 5.
\end{enumerate}
```

To resume numbering from a previous list, use the `resume` key. For example, the code below produces a list that continues numbering from a previous list:

```
\begin{enumerate}
\item First item
\end{enumerate}
More stuff here.
\begin{enumerate}[resume]
\item Second item
\end{enumerate}
```

You can change the numbering style using the `item-label=...` key. Possible counter types include `\arabic` (the default), `\roman`, `\Roman`, `\alph`, and `\Alph`. For example, to create a list with lowercase roman numerals in parentheses as the labels, you would write:

```
\begin{enumerate}[item-label=(\roman*)]
\item First item
\item Second item
\end{enumerate}
```

3.4 Headings & Document Structure

To produce accessible documents, it is important to use the proper heading commands to create a logical document structure. Every document should include `\title{...}` in its preamble, even if you don't intend to call `\maketitle` in the document body. This ensures that the document has a title tag in the PDF structure.

If you do intend to use `\maketitle`, do not patch the command to adjust its appearance; this will break the tagging structure. Customization options for the appearance of titles and headers are being developed and are planned to be released later in 2026.

Similarly, use the standard sectioning commands (`\section{...}`, `\subsection{...}`, `\subsubsection{...}`, etc.) to create headings in your document. Avoid using custom formatting (e.g., changing font size or style) to create headings, as this will not be recognized by screen readers. If you choose to use unnumbered sections with `\section*{...}`, be aware that these will not appear in the table of contents and the bookmarks in the output pdf

by default. You can manually add these to the structure by using the `\addcontentsline` command, as shown in the example below.

```
\section*{Course Introduction} \label{day1}
\addcontentsline{toc}{section}{Course Introduction}
```

3.5 Mathematics

To ensure that mathematical content is accessible, you need to compile your document using `lualatex` and include the `unicode-math` package in your preamble. There are two different methods of attaching the MathML representation of your mathematics to the PDF structure, each of which works better with different pdf viewer/screen reader combinations. The `mathml-SE` option creates MathML as Structure Elements, which works well with Adobe Acrobat and NVDA or JAWS with the MathCat add-on. The `mathml-AF` option creates MathML as Associated Files, which works well with Foxit Reader and NVDA or JAWS with the MathCat add-on. You can include both options to cover both combinations, as shown in the preamble example in Section 2.

In general, you should avoid the practice of nesting math in text in math mode. So instead of writing

```
\begin{cases}
 x +y = 10 & \text{if $x$ > 0} \\
 x - y = 5 & \text{otherwise}
\end{cases}
```

write

```
\begin{cases}
 x +y = 10 & \text{if } x > 0 \\
 x - y = 5 & \text{otherwise}
\end{cases}
```

3.5.1 MathML Intents

To provide additional context for screen readers, you can use MathML intents to specify the reading of certain mathematical expressions. This is especially useful for symbols that have different meanings in different contexts.

TODO: Add examples and explanation.

3.5.2 Screen Readers and MathML in PDF

At present, there is a chicken-and-egg problem with MathML in tagged PDFs. For many years, no one has produced properly tagged PDFs with mathematics content, so most pdf viewers and many screen readers have not implemented support for properly set up documents. This is slowly changing, but at present, the best combination for reading mathematics in PDFs produced by LaTeX is to use Adobe Acrobat Reader or Foxit Reader to view the PDF and then use NVDA or JAWS with the MathCat library add-on to read the document. Any of these combinations does a good job of reading mathematics content when the document is properly tagged.

Unfortunately, many automated accessibility checkers do not yet handle the PDF 2.0 schema with MathML properly, so they will report errors even when the document is correctly tagged. For example, Ally reports that all mathematics is inaccessible even when it is properly tagged with MathML, because it only checks against the older PDF 1.7 standard which would require alt text to be given on each formula object. You can enable generation of alt-text in the tagging-setup options, which will automatically include your LaTeX source code as alt text for each formula and pass these checkers, but this is not recommended because some pdf viewer/screen reader combinations will read the alt text instead of the MathML! This happens for Foxit Reader + NVDA with MathCat, for example.

4 Examples

In this shared folder I have included several examples of accessible documents I have created with LaTeX recently. These include:

- a worksheet
- an exam with both multiple choice and free response questions and solutions
- a partial set of course notes
- a set of presentation slides (converted from `beamer` to the `ltx-talk` documentclass)

This reference document itself is also an example.

Many more examples are available in the Tagging Project Examples page.

TODO: Add better comments explaining the important parts of each example.

5 Reference Links

5.1 References on Creating Tagged PDFs

Much of the ongoing work and current documentation on the creation of tagged and accessible PDFs from LaTeX can be found at via the LaTeX3 Tagging Project repository on GitHub: `https://latex3.github.io/tagging-project/` More detailed documentation for many of the features described in this reference document can be found there.

In particular, basic guidelines on implementing the tagging code are located here. More advanced documentation is here.

Two of the project leaders gave talks at the TUG 2025 conference on the current state of the project, with examples. You can find the talk from Frank Mittelbach here and the talk from Ulrike Fischer here.

Bugs and issues can be reported on the project's page: `https://github.com/latex3/tagging-project/issues`. This is a project that is constantly under development, and the best way to bring attention to important failures in accessibility is to report them there.

5.2 Ways to Check Accessibility

You can use either Adobe Acrobat or the Foxit PDF reader along with the NVDA screen reader using the MathCat add-on (instructions to install) to check whether your documents can be read properly. Both also have accessibility checkers in their pro versions, but free accessibility checkers are available online, e.g.

- VeraPDF
- ngPDF - derives HTML from tagged PDFs
- showtags - Evaluates the PDF-structure (produced by LaTeX project)