

Sp^eL^AT_EX — Speech-enabled L^AT_EX

Walter Daems and Paul Levrie

Abstract

Text-to-speech software has existed for quite some time. However, it still struggles to properly read mathematical expressions and equations. The Sp^eL^AT_EX package, in conjunction with a Perl script, allows you to transform your text into a document with a hyperlink for every chunk of text, equation, table or figure. These hyperlinks, when clicked, cause your media player to read the content out loud, coping properly with mathematics and graphics.

The package targets an audience with reading disorders, as well as people looking for help in focusing on the text by having it read out loud while also reading it silently.

At present, the package does not rely on PDF tags for practical reasons, but that is probably where it is heading in the long term.

1 Introduction

The prevalence of significant reading disorders is estimated between 2% and 20% across studies. The value depends heavily on the definition used for the disorder, and the threshold that is used to judge whether they are significant [4].

At our institute, the University of Antwerp in Belgium, the number of students with reading and/or writing disorders (or at least are aware of these disorders) is increasing. Approximately 5% of the students are registered with such a disorder and the number has been rising steadily over the past 10 years. This number is probably an underestimation as some students choose not to register their disorder.

A large portion of the study materials we offer to students is still written material. We believe that this will continue to be the case, even given the multimedia and AI options that have become mainstream. Let's not go into this debate for now. However, written course texts are suboptimal for students with reading disorders.

For small texts, reading them out loud and recording them using a voice recorder to create an aid for our target group, is still feasible. We have done that in the past for exam assignments. However, for bigger texts (like course syllabi) this is beyond the time a professor can afford spending on students. Yes, economic laws also govern teaching! Sp^eL^AT_EX solves this issue by transforming a L^AT_EX manuscript into a PDF with hyperlinks to automatically generated speech (audio) files [9].

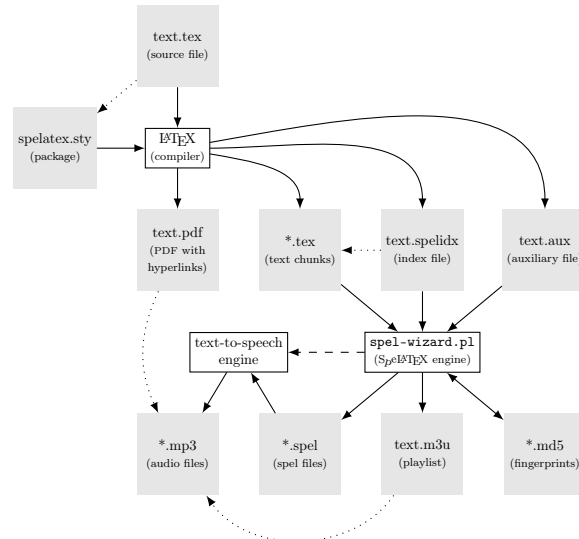


Figure 1: Basic tool setup of the Sp^eL^AT_EX package; filled boxes indicate files, outlined boxes indicate tools; solid lines indicate the use or creation of files, dotted lines indicate references, dashed lines indicate a control relationship.

2 The operational flow

How does Sp^eL^AT_EX work? Let's go through the flow as outlined in Fig. 1. The process consists of two steps: compilation and incantation.

Compilation

We start at the top with your manuscript (`text.tex`) by loading the `spelatex.sty` style file and embedding every chunk of your document that needs to be converted to speech in a `spelchunk` environment. When you compile your manuscript, a PDF file will be generated that contains clickable hyperlinks to audio files that need to be generated later in the process.

As shown, besides the PDF, as a side effect of the compilation, (1) every text chunk will be written to a separate `.tex` file in a dedicated subdirectory, and (2) a `.spelidx` index file will be generated that contains information on the chunks that need to be converted to speech, including extra information to help the text-to-speech conversion. As usual, (3) an auxiliary file (`.aux`) is generated that is also needed in the text-to-speech conversion.

Incantation

These three source files are used by the Perl script `spel-wizard.pl` to generate (a) plain English text (`.spel` files) for all chunks to be converted to speech, (b) mediaplayer playlists (`.m3u`) for sections that gather more than one chunk and (3) fingerprint files (`.md5`). These fingerprint files are preserved between L^AT_EX compilations to keep track of when a text

chunk changed. If the fingerprint of a chunk does not correspond to its old value, then the `spel-wizard.pl` script will launch the text-to-speech converter that reads the `.spel` file and transforms it into an audio file (in mp3 format¹ in Fig. 1). These are the audio files (and the `.m3u` files) that the PDF file references. Keeping track of the fingerprints avoids the costly action of regenerating audio files for chunks that did not change.

For a `text.tex` source file, this flow just takes two commands (with `$` being your shell prompt):

```
$ lualatex text
$ spel-wizard.pl -v text
```

The `-v` flag allows you to follow the text-to-speech conversion in a little bit more detail.

Unfortunately, the fact that multiple tools are required for this flow to work (your \LaTeX compiler, a Perl interpreter and the `SpEL:Wizzard` module, and a text-to-speech tool) makes the setup no *walk in the park*. We'll try to explain this as best we can in section 4.

3 Getting a taste of it

You can go to the web page of $\text{S}_p\text{e}\text{\LaTeX}$ on CTAN (<https://ctan.org/pkg/spelatex>) and right-click on the “Package documentation” to save it to disk. Then open the file with a PDF reader — but avoid your web browser opening the document; use `evince`, `okular` or Adobe Reader instead.

Many of the structure elements of this text, like the title, the author block and the (sub)section headers, can be clicked on. This will cause your browser to pop up and (if the file association of the `.mp3` file extension on our system correctly connects your browser to your media player) play the spoken version of the element you just clicked. If not, then skip this section and try the instructions below in section 4.

In addition, you will see small triangles after the (sub)section headers near the right margin of the text. If you click these, the entire (sub)section will be spoken out loud for you. Similarly, there are triangles at the left margin before each paragraph, which will speak that paragraph.

Finally, continue to section 6 of the document and click on some of the equations you see there. The entire equation area is clickable and will cause the formula to be spoken out loud for you.

That's all you need to know to be able to use a PDF that has been generated using $\text{S}_p\text{e}\text{\LaTeX}$. Where

¹ The mp3 format has been chosen for this article because it is best supported over all platforms and media-players. Note, however, that other formats exist that provide a better quality vs. size ratio, e.g., opus (in an ogg container).

do the audio files come from? They are available as part of the package on the CTAN servers (and their mirrors). Note that linking to files on servers is not the usual behavior. It's generally better to make a PDF that links to local files. This will save you quite a bit of internet traffic and also avoid the inevitable delay when you are not on a fast network.

Did you notice that this article — if you are reading a PDF version — is also $\text{S}_p\text{e}\text{\LaTeX}$ -enabled? We have hidden the markers (by specifying the package option `markervisibility=none`) to avoid confusion when reading this article on paper. Try clicking a section heading, or where the markers normally are. It works!

4 Setting things up

4.1 Preparing your PDF reader

Most PDF readers work out of the box. In some cases you need to allow your reader to open links to local files. It should give you an error message if it refuses to open such links. Also, make sure the file extensions `.mp3` (or whatever format you choose to use in the future) and `.m3u` link to your music player. Most media players are capable of playing these formats. We can recommend `SoX`, `totem` or `VLC` on GNU/Linux, `QuickTime` player on macOS, and `VLC` on Windows. Why not the standard Windows Media Player? The latter requires you to declare every folder containing audio files you want to play as a Media Player library. If you don't mind this extra hurdle: no problem, go ahead.

4.2 Installing the package

Three parts are needed to get the $\text{S}_p\text{e}\text{\LaTeX}$ system operational:

1. The \LaTeX part
2. The Perl part
3. The Text-to-speech part

The \LaTeX part

If you are using a major $\text{T}_\text{E}\text{X}$ distribution like $\text{T}_\text{E}\text{X}$ Live or $\text{M}\text{i}\text{K}\text{T}_\text{E}\text{X}$, the latest version of the package will be installed on your system. If it is not, use the package manager to install it manually. On $\text{T}_\text{E}\text{X}$ Live this is as simple as running the following command in a shell:

```
$ tlmgr install spelatex
```

For $\text{M}\text{i}\text{K}\text{T}_\text{E}\text{X}$, start the $\text{M}\text{i}\text{K}\text{T}_\text{E}\text{X}$ console and search for the $\text{S}_p\text{e}\text{\LaTeX}$ package in the Packages section. Right click on it and press ‘install’.

If you don't like package managers, you also can use the style file `spelatex.sty` directly. Just make sure it is somewhere on your \LaTeX search path.

The Perl part

On GNU/Linux and macOS, a Perl interpreter is normally part of the system installation. If not, install a Perl interpreter and its package manager (`cpanminus`) using your system's software package manager. Then, use the perl package manager to install the module `SpEL::Wizard` as follows:

```
$ cpanm SpEL::Wizard
```

On Windows, we recommend that you install Strawberry Perl [10]. Then open a *Command Prompt* and type

```
$ cpan SpEL::Wizard
```

This will install the `spel-wizard.pl` script on your system. Make sure it is in your search path.

The text-to-speech part

This is probably the hardest part. The starting point is a configuration file for `spel-wizard.pl`. You can check the `SpELETeX` package documentation to see what the various options are for its location. To keep things simple in this article, we'll assume you put it in your home directory on GNU/Linux or macOS or in your `%userprofile%` folder on Windows, and that you name it `tts.conf`.

This configuration file determines the text-to-speech tool that will be used during incantation, including the voice you prefer.

We can recommend two free options for a text-to-speech tool:

- For GNU/Linux and macOS: Festival [5].²
- For Windows: Balabolka [3].

Read the documentation of these packages to learn how to install them. Make sure your `PATH` environment variable contains the path to the correct executable.

Here are working configuration files for each of these systems:

(for Festival)	(for Balabolka)
[engine]	[engine]
tts=festival.pl	tts=balabolka.pl
[languagetags]	[languagetags]
english=en-gb	english=en-us
[voices]	[voices]
english=en1_mbro1a	english=Zira

The `tts=festival` line specifies which text-to-speech script (on the left, `festival.pl`) to call for the conversion. The language tag `english=en-gb` specifies which internationalization (`i18n`) module

² Many package managers can install them; look for the keywords `festival` and `festvox`.

(`en-gb`) the `spel-wizard` script should use for the selected babel language (`english`). Finally, the voices section specifies which voice (`en1_mbro1a`) to use for the selected babel language (`english`). For multilingual documents you will need multiple lines in both the `languagetags` and `voices` sections.

A conversion script (e.g. `festival.pl`) can be written in any available scripting language. It takes three arguments. In order:

1. the name of the `.tex` file to convert to an audio file,
2. the name of the audio file; the file extension will determine which format will be generated, and
3. the voice to be used.

If you want to integrate your own text-to-speech tool, take a peek into the existing scripts. If you don't manage yourself to create a working setup, you're welcome to ask help from us. If it is a text-to-speech tool that might be of use to others, we will be glad to help you out (and incorporate the setup in the `SpELETeX` package).

Personally, we often use AWS Polly [2] and Azure TTS [7] for high-quality voices. Conversion scripts are provided with the package; however, you need to apply for your own Amazon AWS or Microsoft Azure account and perform some setup involving configuration and credential files. It is beyond the scope of this article to explain this in detail.

5 Using the package

Once the setup is working, using the package is fairly straightforward. The following plan should get you going. We assume that `manuscript.tex` is the name of your document.

1. Make a subdirectory `manuscript-spel` in the directory that contains `manuscript.tex`.
2. Add the following lines to your preamble:


```
\usepackage[english]{babel}
\usepackage{spelatex}
```

 This will select the language you need and load the `SpELETeX` package. Note that the `SpELETeX` package only supports English and Dutch at the time of writing this article. Do you want to contribute your own language? Contact us.
3. Replace every `\item` by `\spelitem` and wrap the text after it in curly braces.
4. Wrap every paragraph in a `spelchunk` environment or put it as an argument to the `\<<<` macro. You will find that the `spelchunk` environment is a bit more robust, while the triple-lesser-than-macro alternative is a bit less visually intrusive in your text editor.

5. Provide the non-textual elements of your document (e.g. graphics) with an appropriate annotation, by embedding them in a `spelchunk` environment followed by a `spelchunkad` environment that contains a textual description of the non-textual element (the suffix `ad` stands for ‘audio description’).

Then, execute (assuming we use Lua \LaTeX):³

```
$ lualatex manuscript
$ spel-wizard.pl -v manuscript
$ evince manuscript.pdf # okular, acroread, ...
and enjoy!
```

6 Example

Below is a minimal working example, that you can start from, to explore the package. Don’t forget step one of the plan that we described in the previous section, to make a `...-spel` directory.

The example uses a macro that we did not describe above: `\spelmacad`. Check the documentation for that.

```
\documentclass[a4paper]{article}

\usepackage[margin=1in]{geometry}
\usepackage[english]{babel}
\usepackage{spelatex}
\usepackage{tikz,pgfplots}

\title{Simple \spelatex-example}
\date{11/09/2024}

\begin{document}
\maketitle
\section{Basic use}
\begin{spelchunk}
  You just have to embed every paragraph in a
  spelchunk environment. The alternative is to
  use the triple-lesser-than-macro (as is
  done in the next paragraph), which does
  exactly the same, though it is less robust.
\end{spelchunk}

\<<<{We illustrate how to treat lists with a
  description of the taste of some fruits:}
\begin{description}
\spelitem[lemon]{tastes sour}
\spelitem[banana]{tastes sweet}
\end{description}

\<<<{
  Figure~\ref{fig:example} is also audio
  enabled, both in the caption and with a
  separate comment.
}
```

³ On Windows, omit the `.pl` extension in the second command.

```
\begin{figure}
\centering
\begin{spelchunk}[arealink]
\begin{tikzpicture}
\begin{axis}[xlabel=$x$,
ylabel=$y$,zlabel=$z$,grid=both]
\addplot3[patch,shader=faceted interp,
samples=10] {x^2-y^2};
\fill (axis cs:0,0,0)
circle[radius=1pt] node[above] {$p$};
\end{axis}
\end{tikzpicture}
\end{spelchunk}
\begin{spelchunkad}
  This is the graph of a biquadratic surface.
  Note the point  $p$  which we call a saddle
  point. This is a point where the first
  partial derivatives are zero and the second
  partial derivatives in  $x$  and  $y$  have
  opposite signs.
\end{spelchunkad}
\caption{A simple biquadratic surface}
\label{fig:example}
\end{figure}
```

```
\section{Some equations}
\<<<{
  The \texttt{arealink} option causes the link
  to cover the entire area of the equation.
  This section shows also some advanced
  features that are explained in the
  documentation of the package.
}
\newcommand\dd[1]{\,\mathrm{d}\#1}
\spelmacad{eu}{e}
\spelmacad{dd}[1]{d #1}

\begin{spelchunk}[arealink]
\begin{equation}
\int_{0}^{+\infty} \mathrm{e}^{-cx} \dd{x}
= \frac{1}{c}
\end{equation}
\end{spelchunk}
\<<<[arealink]{
  The arealink option also works with the
  triple-lesser-than-macro alternative.
}
\end{document}
```

7 Discussion

In this section, we will first discuss the difficulty of joining a free-form typesetting language like \LaTeX with the requirement of parsing the constructs to convert them to meaningful language again. Then we’ll discuss the things that went well in developing the package, and also the things that are suboptimal and even fragile.

7.1 L^AT_EX: Praise and critique

L^AT_EX is built around the principle of separating content and formatting. While writing the document, the author can focus on the content, indicating the structure using sectioning and enumeration commands. It supports non-textual material such as graphics and equations in a structured way. The L^AT_EX compiler does all the formatting and book-keeping for textual aids, like references, tables and so forth. The end result looks magnificent. We all love it. So far the praise.

However, when it comes to mathematics, L^AT_EX is a lot more liberal. That is for good reason: mathematical notation is also quite liberal. And we must admit: we all like to create our own special notation, and more than once we have used illogical constructions, or worse, reused existing constructions by ‘overloading’ them. In addition, how to read mathematical equations out loud is not fixed in rules that are generally agreed on. This creates a problematic starting position for a tool that reads the equations based on parsing their L^AT_EX code.

Below are a few examples of our critique.

Simple vs. rigorous reading

Consider the following:

$$y = \sqrt{\sin \frac{x}{x+1} - 1} \quad (1)$$

A simple way to read this could be: “y equals the square root of the sine of x over x plus one minus one.” Note that this incantation is full of ambiguity. A more rigorous way of reading this expression would be: “y equals the square root of the sine of x over x plus one, this ends the fraction and ends the sine minus one, this ends the square root.”

We opted for the first version, based on the fact that we assume the text is in front of the reader when the text is being spoken. The latter version would allow for listening to the text in a podcast-like style, without the document in front of you. However, we think that only a few people possess the mental ability to keep an overview of an equation with all the nested ‘end’ clauses.

In short, S_peL^AT_EX opts for *visually supported reading*. Maybe in the future an option will be implemented to allow you to make the choice yourself at incantation time.

Function argument vs. grouping brackets

Consider the following:

$$y = c(a + b) \quad (2)$$

$$y = f(x + 1) \quad (3)$$

If the two lines were shown to you separately, you’d probably read the former as “y equals c times a plus b” and the latter as “y equals f of x plus one”. This is based on our capability to use contextual information to see which notation has been used. However, S_peL^AT_EX is not an AI tool and does not have this capability.

In the current implementation, a single letter followed by brackets is recognized as a function. One can force reading it as group brackets by using `\left` and `\right` before the brackets. This has been done in the equation pair above.

Switching between math and text mode

When reviewing articles of researchers in our group, we often correct a typesetting error made when specifying a variable with a subscript that is an abbreviation, such as τ_{ff} . In such a case you need to make clear to L^AT_EX that the subscript is not itself a mathematical expression, therefore you need to enclose the double f in a `\mathit` (as above) or `\mathrm` command, such that it is not typeset as τ_{ff} . If your subscript contains spaces (which we don’t recommend), you might even need to use `\textit` or `\textrm`. S_peL^AT_EX also needs this clarity, in order not to read the construct as “tau f times f”.

Meta-annotations

L^AT_EX allows you to put nice extras in relation to sub-expressions (such as braces) to clarify their meaning, such as:

$$H(s) = \frac{\overbrace{\alpha}^{\equiv B_1}}{\alpha + 1} s + 1 \quad (4)$$

$$\underbrace{(\beta - 1)s^2}_{\equiv A_2} + \underbrace{2\zeta}_{\equiv A_1} s + \underbrace{\delta + 3}_{\equiv A_0}$$

We have chosen not to read that extra information, as it complicates reading the original equation itself. Your feedback or suggestions on these issues are welcome.

Units

We are big fans of the `siunitx` package [8] as it forces you to use correct units and typesets them flawlessly. In addition, if you refrain from using the abbreviations the package provides, e.g. writing `\nano\meter` instead of `\nm`, S_peL^AT_EX works out of the box, by just removing the backslashes. As always, *Joseph is right* in the implementation of his packages.

High-level constructs

It is difficult to give meaning to high-level constructs if one typesets them on a low level. Consider as an

example:

$$(x + y)^n = \sum_{i=0}^n \binom{n}{i} x^{n-i} y^i \quad (5)$$

If one typesets this using the `\matrix` environment, it will be read out loud as a matrix, instead of a binomial coefficient.

However, if the `\binom` macro is used, and provided an appropriate audio description, such as

```
\spelmacad{binom}[2]{%
  the binomial coefficient of #2 out of #1,}
```

the construct reads out loud like a charm.

Conclusion

Many more examples could be shown, but the general advice should be clear by now: *be explicit and keep the description at a high level!*

7.2 `SpeLATeX`: The good, the bad and the ugly

Usability of the resulting PDF

Overall, we think that the learning curve of using a PDF that is equipped with hyperlinks to the audio files is smooth and short. After a while, even working with a PDF that only contains the right-hand-side markers, requiring you to click on the hidden link-area before the first line of a paragraph, feels very natural.

Direct access to the required material is possible by directly clicking before a paragraph or on an equation, while more linearly sequential access is facilitated using the right-hand-side markers.

Disadvantages that we see:

- you need to distribute your document as a file plus a folder full of audio files; embedding the document in an archive file like zip or tar alleviates this problem. Common document formats (like `.docx`) do the same. We even might claim our own file extension.
- a document will take a lot more disk space, which is not very good for sustainability.

Advantages we see:

- bundling the audio files with a document allows for providing high-quality audio.
- the costs are kept low for the reader, as the author pays for the high-quality audio generation.

In the medium term, embedding the audio files in the PDF itself is a logical choice. It avoids the zoo of audio files. We opted not to do that now, as the current PDF viewers struggle with playing the audio files without requiring extra mouse clicks to confirm various things or to save them to disk first.

In the long term, embedding a plain-text version of the graphical material (like equations and descriptions of graphical material or tables) as tags inside the PDF, such that the reader of your PDF viewer can take care of the text-to-speech conversion is also a logical choice. We opted not to do that either, as most PDF readers do not support this feature. In addition, the quality of the built-in readers is not as high as the currently available AI voices from some major providers (like Google and Amazon).

In any case, having a PDF that is read out loud properly is a big advantage for students suffering dyslexia, both for course material as well as for (exam) assignments.

Overhead when preparing a manuscript

Only part of the document is equipped automatically with links (title, date, author, footnotes, (sub)section headers, ...). The overhead of having to encase all the paragraphs manually in `spelchunk` environments is considerable. It also puts a big visual burden on managing your manuscript in an editor. For this reason, the triple-lesser-than-macro alternative has been implemented, although it is not as robust.

The culprit for the encasing requirement is the fact that paragraphs are rather inaccessible in `LATEX`. We considered auto-wrapping them using `LATEX3` constructs. It is possible, but not elegant either, as (sub)section headers and include files keep getting in the way of the wrapping process. Any fresh ideas on this front would be very welcome!

In an earlier version of the `SpeLATeX` package we had planned to make an `orgmode` exporter available for Emacs to take care of this. However, the user base for that is too small. Creating it would deflect the focus from solving the real problem within the `LATEX` engine.

Flexibility of the package

The flexibility of the package has not been highlighted much in this article. However, it is made multilingual with proper Babel support and `i18n` support in the Perl `SpeL:Wizard` module.

In addition, both the `\spelmacad` macro and the `spelchunkad` environment allow you to override any result of `spel-wizard.pl` that you are not fully pleased with. So, you can work around our bugs or other problems when needed.

If you are interested in learning more about this in a subsequent article, we will be happy to devote some time to that.

Fragility of the package

The `SpeLATeX` package redefines most sectioning commands, as well as titlepage information, footnotes, etc. This makes the package fragile with respect to

compatibility with other packages that do similar things. Maybe some $\text{\LaTeX}3$ hooks can be used for our implementation, without having to redefine these commands.

The Perl `SpeL::Wizard` uses a regular expression grammar parser, which is very capable, but also slow and not easy to debug. *Not a concern for me as a user of the package*, you might think. However, sometimes it even fails to finish in an acceptable time (it continues looping, without matching the correct grammatical construct). This might cause the script to stall. Very annoying. The infinite flexibility of \LaTeX , allowing the user to write equations in exotic ways, does not help either. We did not yet implement a timeout mechanism to break free from these loops, but it is on the todo list.

Immaturity of the package

The first version of $\text{Sp}\text{\LaTeX}$ was implemented back in 2008. Then, due to professional constraints, we were not able to continue working on it. We started working on it again in 2023, with the single goal to finish the package as soon as possible to be able to involve the community in the further development. If not for actual help in coding, then at least in providing feedback.

This explains why most of the code still heavily relies on rather primitive \TeX constructs, rather than on the more powerful $\text{\LaTeX}3$ [6] machine gun that package writers can use these days. This makes the package rather ugly and inconsistent in the current version. We have no doubt that it will mature over time.

8 Missing in action

Many issues have not been discussed in this article, such as:

- the stubbornness of some text-to-speech converters which makes $\text{Sp}\text{\LaTeX}$'s life difficult,
- its multilingual capabilities and how they have been implemented,
- how integration with other packages has been conceived (e.g., `amsmath` [1]), and
- how other text-to-speech services may be integrated in the tool chain.

These might be subjects for a subsequent article.

9 Help!

We can help you ...

The current $\text{Sp}\text{\LaTeX}$ package is in active development. That means that there will be errors. If you find any, please signal them to us. We will be happy to follow up on them and try to help you to find a temporary workaround.

... and you can help us

By asking our assistance with chunks that are not converted well, you help us to improve the package and the `spel-wizard.pl` script as well. However, expect us to judge whether the constructs that you try to push through $\text{Sp}\text{\LaTeX}$ make sense. This especially holds for equations. You must be prepared to bring structure to them. Only in this way can we make them parsable by the `spel-wizard.pl` script. No doubt, discussions will emerge on how to properly read specific mathematical constructs.

And you can contribute even more. If you speak a language that we don't master, and would like to use it with $\text{Sp}\text{\LaTeX}$, you can volunteer in translating the internationalization (`i18n`) module in the `SpeL::Wizard` module. You'll earn a spot in the thank-you section of the package.

10 Conclusion

We think $\text{Sp}\text{\LaTeX}$ deserves a spot in the current range of tools supporting people with reading disabilities. It also supports you in creating richer texts in general. Though it is still in development, it is usable in its current state. We look forward to your feedback!

References

- [1] `amsmath` — AMS mathematical facilities for \LaTeX . ctan.org/pkg/amsmath.
- [2] Amazon Polly. aws.amazon.com/polly.
- [3] Balabolka text-to-speech program for MS Windows. cross-plus-a.com/balabolka.htm.
- [4] C. Di Folco, A. Guez, et al. Epidemiology of reading disability: A comparison of DSM-5 and ICD-11 criteria. *Scientific Studies of Reading*, 26(4):337–355, 2022. doi.org/10.1080/10888438.2021.1998067
- [5] The Festival speech synthesis system. www.cstr.ed.ac.uk/projects/festival.
- [6] The \LaTeX project. latex-project.org/latex3/.
- [7] Microsoft Azure TTS. learn.microsoft.com/en-us/azure/ai-services/speech-service/text-to-speech.
- [8] `siunitx` — A comprehensive (SI) units package. ctan.org/pkg/siunitx.
- [9] $\text{Sp}\text{\LaTeX}$ — Create PDF documents with hyperlinks to audio fragments. ctan.org/pkg/spelatex.
- [10] Strawberry Perl for MS Windows. strawberryperl.com/.

◇ Walter Daems and Paul Levrie
University of Antwerp, Belgium
`walter dot daems (at)
uantwerpen dot be`