plantuml

Version 0.8.1

Embed PlantUML diagrams in LaTeX documents

Write PlantUML diagrams directly in your LaTeX document:

```
\begin{plantuml}
  Alice -> Bob: Hello
\end{plantuml}
```

The package renders the diagram with a local `plantuml.jar` or a PlantUML server and includes it as TikZ, PDF, PNG, SVG, or EPS. It works with lualatex and pdflatex. Rendered diagrams are cached, and can be committed for builds without shell escape, e.g., on Overleaf.

Preconditions

1. Environment variable `PLANTUML_JAR` set to the location of `plantuml.jar`. You get it from https://sourceforge.net/projects/plantuml/files/plantuml.jar/download. Not needed when rendering `eps`/`png`/`svg` through a PlantUML server (see Rendering via a PlantUML server). If neither a jar nor a server is available, the diagram is replaced by a visible placeholder rather than aborting the build.
2. Windows: Environment variable `GRAPHVIZ_DOT` set to the location of `dot.exe`. Example: `C:\Program Files (x86)\Graphviz2.38\bin\dot.exe`. You can install graphviz using `choco install graphviz`.
3. lualatex or pdflatex available, called with the command line parameter `-shell-escape`.
4. For `output=pdf`, use a PlantUML jar that can write PDF: the default `plantuml-<version>.jar` (GPL) or `plantuml-epl-<version>.jar` from the PlantUML releases. The permissively licensed jars (`plantuml-mit-*`, `plantuml-asl-*`, `plantuml-bsd-*`, `plantuml-lgpl-*`, `plantuml-gplv2-*`) lack the PDF library and fail with `NoClassDefFoundError: org/openpdf/...`. `output=pdf` needs neither inkscape nor a LaTeX engine for PlantUML, and the text in the diagram stays searchable. The PDF uses one of the standard PDF fonts Times-Roman, Helvetica, or Courier, not embedded, regardless of `skinparam defaultFontName`: names containing `Mono` or `Courier` give Courier, names containing `Serif` give Times-Roman (including PlantUML's default `SansSerif`), all others give Helvetica. For a sans-serif diagram, use `skinparam defaultFontName Helvetica`. If the font must be embedded (e.g., PDF/A or print), use `output=svg`, `output=eps`, or `output=latex`.
5. In case you want to have SVG images converted to PDF, ensure that `inkscape.exe` and `pdfcrop` are in your path. You can get inkscape using `choco install inkscape`. `pdfcrop` should be part of your latex distribution.

Examples

Minimal Example

LaTeX source:

```
\documentclass{scrartcl}
\usepackage{plantuml}
\begin{document}
\begin{plantuml}
!theme cerulean-outline
Alice -> Bob: test
\end{plantuml}
\end{document}
```

The `!theme` line is optional; it picks a PlantUML colour theme (the examples use `cerulean-outline` for a cleaner look than the colourful default). `@startuml` and `@enduml` are optional: when the diagram body omits them, PlantUML adds them automatically (issue #4), so the examples here leave them out. You may still write them explicitly if you prefer.

Compilation: `lualatex -shell-escape example-minimal` (or `pdflatex -shell-escape example-minimal`)

Result:

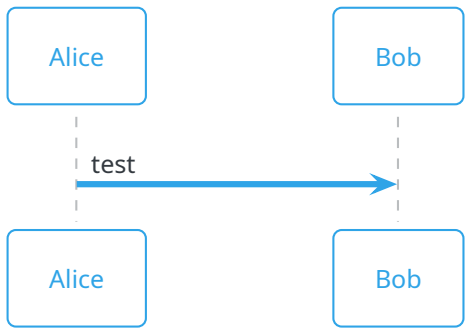


Figure 1: Minimal example

Rendering from a File

Use `\plantumlinput{<file>}` to render a diagram stored in an external PlantUML source file, analogous to minted's `\inputminted` and listings' `\lstinputlisting` (issue #3):

```
\documentclass{scrartcl}
\usepackage{plantuml}
```

```
\begin{document}
\plantumlinput{example-input-file.puml}
\end{document}
```

It honors the `output` mode and reuses the same caching, server, and `-output-directory` handling as the `plantuml` environment. The file is read relative to the current working directory, so paths in subdirectories (e.g. `\plantumlinput{diagrams/foo.puml}`) work when compiling from the project root. See `example-input-file.tex`.

Example Class Relations Rendered Using SVG

LaTeX source:

```
\documentclass{scrartcl}
\usepackage{graphics}
\usepackage{epstopdf}
\epstopdfDeclareGraphicsRule{.svg}{pdf}{.pdf}{
  inkscape "#1" --export-text-to-path --export-filename="\OutputFile"
}
\usepackage[output=svg]{plantuml}
\begin{document}
\begin{plantuml}
!theme cerulean-outline
class Car

Driver - Car : drives >
Car *- Wheel : have 4 >
Car -- Person : < owns
\end{plantuml}
\end{document}
```

For older Inkscape use this LaTeX source:

```
\documentclass{scrartcl}
\usepackage{graphics}
\usepackage{epstopdf}
\epstopdfDeclareGraphicsRule{.svg}{pdf}{.pdf}{
  inkscape -z --file=#1 --export-pdf=\OutputFile
}
\usepackage[output=svg]{plantuml}
\begin{document}
\begin{plantuml}
!theme cerulean-outline
class Car

Driver - Car : drives >
Car *- Wheel : have 4 >
```

```
Car -- Person : < owns
\end{plantuml}
\end{document}
```

Compilation: `lualatex -shell-escape example-class-relations` (or `pdflatex -shell-escape example-class-relations`)

Result:

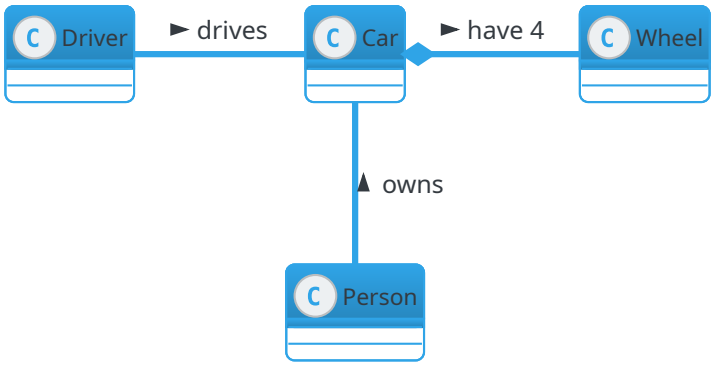


Figure 2: Class relations rendered using SVG

Rendering via a PlantUML server

Instead of a local `plantuml.jar`, `eps`, `png`, and `svg` diagrams can be rendered by a PlantUML server over HTTP. This needs only `curl` — no Java and no local PlantUML installation — which is handy on CI and shared build machines (see issue #6).

Point the package at a server with the `server` option:

```
\usepackage[output=svg, server=https://www.plantuml.com/plantuml]{plantuml}
```

Alternatively set the `PLANTUML_SERVER` environment variable (the package option takes precedence over it):

```
export PLANTUML_SERVER=https://www.plantuml.com/plantuml
```

You can run your own server, for example with Docker:

```
docker run -d -p 8080:8080 plantuml/plantuml-server:jetty
# then use server=http://localhost:8080
```

Notes:

- For vector output without extra tools, use `output=eps`: LaTeX converts the EPS to PDF itself (via `epstopdf`/Ghostscript, shipped with TeX Live),

so neither inkscape nor a local jar is needed. Its text is drawn as outlines, so it is sharp but not searchable in the PDF; use `svg` (needs inkscape) for real text. The server offers no `pdf` or `latex`/TikZ output.
- When a server is configured, `output=eps`, `output=png`, and `output=svg` are rendered by it. `output=pdf` and `output=latex` (TikZ, the default) always use the local `plantuml.jar`, as the server cannot produce them, so keep `PLANTUML_JAR` set for those.
- See `example-server--eps.tex`, `example-server--png.tex`, and `example-server--svg.tex` for complete examples.
- The diagram source is sent hex-encoded in the request URL, so a very large diagram may hit the server's URL-length limit.

Caching

Generated diagrams are cached by a hash of their source, similar to minted and memoize. Each diagram is written to `plantuml-<hash>-converted-to.<ext>`, and PlantUML (or the server) is invoked only when no file for that hash exists yet. Unchanged diagrams — as well as repeated or reordered ones — reuse the cached output, so recompiles are fast and only edited diagrams are regenerated. The `-converted-to` suffix means the standard TeX `.gitignore` already ignores these files (via `*-converted-to.*`), so no extra entry is needed. Delete the `plantuml-*-converted-to.*` files to clear the cache.

Committing rendered diagrams (e.g. for Overleaf)

The cached files can be committed, so the document builds where PlantUML is not available, for instance on Overleaf or without `-shell-escape`. A diagram whose `plantuml-<hash>-converted-to.*` file exists is used as is; neither Java nor shell escape is needed for it. Only new or changed diagrams need PlantUML: build locally with `PLANTUML_JAR` set and `-shell-escape`, then commit the new files. Until then, such a diagram shows the "could not be generated" placeholder. This also works with `-output-directory`: a diagram file is looked up in the output directory first, then next to the document. TeX Live before 2023 does not pass the output directory to packages: `lualatex` reads it from its command line, `pdflatex` via `ps` (Linux and macOS only).

The standard TeX `.gitignore` ignores these files, so add an exception:

```
!plantuml-*-converted-to.*
```

Files of outdated diagrams are not removed automatically. To clean up, delete all `plantuml-*-converted-to.*` files, build once, and commit the result.

Global Preamble

The default PlantUML style is colorful, which is not ideal for printouts. Instead of repeating `skinparam` settings in every diagram, set a **global preamble** once that is applied to all diagrams (issue #5).

Write it inline with the `plantumlpreamble` environment:

```
\begin{plantumlpreamble}
skinparam monochrome true
skinparam backgroundColor white
skinparam defaultFontName sans-serif
\end{plantumlpreamble}
```

…or keep it in a file and point at it with `\plantumlpreamblefile{<file>}`. Either form affects every diagram that follows (set it once near the top of the document). The preamble is folded into the diagram cache key, so changing it regenerates the diagrams. See `example-preamble.tex`.

The local jar applies the preamble to every diagram via PlantUML's `-config`. The PlantUML server has no such option, so there the preamble is prepended to the source instead; this styles the usual marker-less diagrams, but a server diagram that spells out its own `@startuml`/`@enduml` keeps the default style (render it through the local jar if you need the preamble applied).

Captions and Labels

Diagrams can carry a numbered, `\ref`-able caption (issue #8). The options are:

- `caption={...}` — the caption text,
- `label={...}` — a `\label` for cross-references,
- `float` — place the diagram in a floating `figure` (optional placement, e.g. `float=htbp`; `float=H` needs the `float` package). Without `float` the diagram and its caption are typeset **in place** (no floating), like listings' `caption=` — which avoids the erratic placement of a bare `figure`.

`\plantumlinput` takes these as an optional argument:

```
\plantumlinput[caption={A diagram from a file.}, label={fig:demo}]{diagram.puml}
```

The `plantuml` environment can't take an optional argument (a verbatim environment without a mandatory argument can't carry one), so set the options for the next environment with `\plantumlset`, in the spirit of `\lstset`:

```
\plantumlset{caption={A sequence diagram.}, label={fig:seq}}
\begin{plantuml}
Alice -> Bob: hi
\end{plantuml}
```

See `example-caption.tex`.

Diagram Font (latex/TikZ output)

With the default `output=latex` (TikZ), PlantUML computes each box's size using a **sans-serif** font, then emits TikZ that your document typesets. If that text is set in the document's default (serif) font, it doesn't fit the boxes and the margins look uneven (issue #14). The package therefore renders the diagram

text with `\PlantUmlTikzFont`, which defaults to `\sffamily` so the text matches the boxes. Change it if needed:

```
\renewcommand\PlantUmlTikzFont{\rmfamily}  % serif, e.g. to match a serif skinparam
\renewcommand\PlantUmlTikzFont{}           % use the surrounding document font
```

`png`/`svg` output is unaffected (PlantUML renders the text itself), so it is the simplest choice when exact text fit matters. See `example-mindmap.tex`.

Beamer

The package works in `beamer` (issue #11). The only requirement is that any frame containing a `plantuml` environment is declared `[fragile]`, because the environment captures its body verbatim:

```
\documentclass{beamer}
\usetheme{moloch}
\usepackage{plantuml}
\begin{document}
\begin{frame}[fragile]{PlantUML in Beamer}
\begin{plantuml}
Alice -> Bob: Authentication Request
Bob --> Alice: Authentication Response
\end{plantuml}
\end{frame}
\end{document}
```

Without `[fragile]` you get errors such as `Paragraph ended before \beamer@doframe was complete`. See `example-beamer.tex`.

Installation

Your latex distribution should take care.

For manual installation, copy `plantuml.*` to your local `texmf` folder in the sub directory `tex/latex/plantuml`. See the discussion at tex.sx for the concrete location of the folder on your system.

Development

ADR-0001 explains why pdfLaTeX calls PlantUML via shell escape (#1).

The release is built using GitHub Actions (workflow file) using `release.sh`.

Release preparation:

1. Adapt copyright year (line 1)
2. Adapt as date and version number (line 6) in `plantuml.sty`.
3. Adapt `CHANGELOG.md`.
4. Set a git tag and push.

5. Upload `plantuml.tar.gz` (artifact `ctan` of the Release workflow run for the tag) to CTAN.

Alternative Solutions

TikZ-UML is a very powerful package based on TikZ. More alternative solutions are collected at the CTAN topic UML.

License

`SPDX-License-Identifier: LPPL-1.3c+`