

Table of Contents

md2pdf Comprehensive User Manual	5
md2pdf Documentation Index	6
Available Guides	6
1. User Guide & Feature Reference	6
2. Styling & Themes	6
3. Plugin Authoring Guide	6
4. Visual Feature Showcase	7
md2pdf User Guide & Feature Reference	8
Installation & Setup	8
CLI Flag Reference	8
Configuration File (md2pdf.toml)	9
Configuration Schema Reference	10
YAML Front Matter & PDF Metadata	10
Example Markdown:	10
Cover Page Generation	11
Table of Contents (TOC)	11
Running Headers & Footers	11
Page Footers	11
Page Headers	11
Colour Emoji Support (Twemoji)	12
Download Timeout	12
Task Lists & Checkboxes	12
Footnotes	12
Syntax:	13
Layout Logic:	13
Inline Formatting	13
File Inclusion (!include)	13
Key Behaviors:	13
Admonitions & GitHub Alerts	14
1. Fenced Container Admonitions (Obsidian/MkDocs style)	14
2. Markdown Blockquote Alerts (GitHub-style)	14
Supported Styles and Severity Levels	14
Explicit Page Breaks	14
HTML Comment Syntax	14
Backslash Syntax	14
Progress Reporting	15

Stages reported:	15
Disabling Progress Reporting:	15
Tables & Layout Safeguards	15
Repeating Headers & Row Protection	15
Heading-to-Table Bonding (KeepTogetherParts)	15
Table Column Alignment	16
HTML Line Breaks ()	16
Images & Captions	16
Diagrams (Mermaid) & Math (LaTeX)	16
Optional Local LaTeX Rendering (Matplotlib)	17
Mermaid Syntax	17
LaTeX Math Syntax	17
Caching and Layout Protection:	18
Unicode & Fonts	18
Custom Font Validation	18
Themes & Styling System in md2pdf	19
Built-In Themes	19
The ThemeConfig Dataclass	19
Base Stylesheet Keys	20
Styling Override Plugins	21
Plugin Authoring Guide for md2pdf	22
Overview	22
Public API	22
Hook 1 — Pre-Processor	23
Hook 2 — Element Handler	23
Hook 3 — Post-Processor	23
Hook 4 — Stylesheet Override	24
Entry-Point Declaration	24
Config-File Alternative	24
Error Handling Contract	25
Rules for Plugin Authors	25
md2pdf Feature Showcase & Rendering Test	26
1. Quick Start Guide	26
Command Line Interface	26
Programmatic Python API	26
2. Heading Typography Hierarchy	26
Heading Level 1 (H1)	27
Heading Level 2 (H2)	27
Heading Level 3 (H3)	27

Heading Level 4 (H4)	27
Heading Level 5 (H5 - fallback to H4 style)	27
Heading Level 6 (H6 - fallback to H4 style)	27
3. Inline Elements and Styles	27
4. Lists	27
Unordered Bullet List	27
Ordered Numbered List	28
Task List Checkboxes	28
5. Blockquotes	28
6. Admonition & Callout Blocks	28
Fenced Admonitions	28
Blue / Slate Themes (note, info, todo)	28
Green Themes (tip, success, check)	29
Amber / Orange Themes (warning, attention)	29
Red Themes (danger, error, failure, bug, caution)	29
Teal / Cyan Theme (important)	30
Unknown Fallback Theme	30
GitHub-style Markdown Alerts	30
7. Code Blocks (Syntax Highlighting)	31
8. Tables	31
Table Alignment Showcase	31
9. Diagrams (Mermaid)	31
10. Math & Equations (LaTeX)	31
11. Images & Placeholders	32
Missing Image Placeholder	32
Working Image Example	32
12. Unicode & Special Characters	33
Latin Extended	33
Greek Alphabet	33
Cyrillic	33
Mathematical Operators & Symbols	33
Arrows	33
Box Drawing	33
Currency Symbols	33
Typographic Punctuation	34
Superscript & Subscript Lookalikes	34
Custom Font via md2pdf.toml	34
Colour Emoji (Twemoji)	34
13. Colour Emoji Showcase	34

Inline Paragraph Emoji	35
Emoji in Headings and Lists	35
Today's Task List	35
Status Summary	35
ZWJ Sequences	35
Emoji in Blockquotes	35
14. File Inclusion (!include)	35
15. Page Breaks	35
HTML Comment Syntax	36
Backslash Syntax	36
16. Cover Page Generation	36

md2pdf Comprehensive User Manual

This manual is compiled directly from the `md2pdf` markdown documentation suite using the `md2pdf` typesetting engine itself. It provides the complete documentation for `md2pdf` in a single, print-ready PDF document.

md2pdf Documentation Index

Welcome to the md2pdf documentation! md2pdf is an automated programmatic Markdown-to-PDF typesetting engine written in pure Python. It compiles standard Markdown and advanced elements directly to print-ready PDFs without headless browser dependencies.

Use the guides below to learn about installing, configuring, styling, and extending the engine:

Available Guides

1. User Guide & Feature Reference

Learn about all supported Markdown structures, features, and configuration parameters including:

- **YAML Front Matter:** Metadata extraction to PDF properties.
- **Cover Page:** Auto-generating and prepending cover/title pages using YAML metadata.
- **Table of Contents (TOC):** Auto-generating clickable tables of contents.
- **Running Headers & Section Titles:** Document headers, templates, and page number footers.
- **Colour Emoji (Twemoji):** Automatically substituting emoji characters with high-res colour Twemoji images.
- **Task Lists:** GFM-style checklists (`- []` / `- [x]`) rendered using Twemoji images or Unicode symbols.
- **Footnotes:** Reference link maps and automatic page layout positioning.
- **Inline Formatting:** Support for `~~strikethrough~~`, `==highlight==`, superscript `x^2^`, and subscript `H~2~O` spans.
- **Tables:** Automatic column alignment parsing (`:---`, `:---:`, `---:`) and layout styling.
- **Admonitions & GitHub Alerts:** Fenced admonition blocks and inline markdown alerts with distinct color themes.
- **Page Breaks:** Manual pagination using comment directives and backslash syntax.
- **Mermaid & LaTeX:** Diagram and math rendering via the Kroki API, with concurrent pre-fetching, local caching, offline fallbacks, and optional fast local rendering via Matplotlib.

2. Styling & Themes

Understand how to customize fonts, colors, and layout metrics using `ThemeConfig` blocks inside your configuration files, and how to write stylesheet override plugins.

3. Plugin Authoring Guide

Explore the extension API of md2pdf to hook into the conversion lifecycle:

- **Stage 1 (Preprocessors):** Intercepting and transforming raw Markdown text.
- **Stage 3 (Element Handlers):** Mapping Markdown AST tokens to custom ReportLab flowables.

- **Stage 4 (Postprocessors):** Mutating and rearranging flowables lists before the final PDF compilation.
- **Stylesheet Overrides:** Merging custom theme overrides into the styling pipeline.

4. Visual Feature Showcase

View the primary rendering test document that demonstrates all supported elements, vertical spacing rules, and layout features. Compiling this document (`md2pdf docs/showcase.md -o docs/showcase.pdf`) serves as an end-to-end regression test for changes to the layout engine.

md2pdf User Guide & Feature Reference

This guide provides a comprehensive reference to all features, configuration settings, syntax extensions, and layout rules in the md2pdf Markdown-to-PDF typesetting engine.

Installation & Setup

Install the library using `uv` (recommended) or `pip`:

```
# Using uv
uv tool install pymd2pdf

# Using pip
pip install pymd2pdf
```

To run the CLI:

```
md2pdf input.md -o output.pdf
```

CLI Flag Reference

The CLI is built with `typer` and supports the following options:

Flag	Shortcut	Description
input	(Argument)	Path to the source <code>.md</code> file to be compiled.
--output	-o	Output PDF file path. Defaults to <code><input-filename>.pdf</code> .
--config	-c	Path to a custom <code>md2pdf.toml</code> config file.
--theme	-t	Theme name to apply (defaults to "default").
--offline		Skip external API requests (e.g. Kroki). Uses local placeholders instead.
--verbose	-v	Enable verbose debug-level logging to <code>stderr</code> .
--validate-only		Runs DX-first pre-render validation checks and exits.
--min-image-scale		Minimum image scale factor (e.g. 0.8) before moving images to a new page.
--toc		Prepend a dynamically generated Table of Contents page.
--cover		Prepend an auto-generated cover/title page.

Flag	Shortcut	Description
<code>--header</code>		Header text/template. Supports {title} and {section} placeholders.
<code>--header-on-first-page</code>		Render the running header on the first page.
<code>--emoji / --no-emoji</code>		Enable or disable colour Twemoji image substitution.
<code>--progress / --no-progress</code>		Show or hide stage-level compilation progress on stderr (default: enabled).

Configuration File (md2pdf.toml)

When running the conversion, the CLI automatically looks for configuration files in the following order of precedence:

- 1 File explicitly passed via `--config / -c`.
- 2 `md2pdf.toml` in the current working directory.
- 3 `~/.config/md2pdf/md2pdf.toml` in your home directory.
- 4 `~/.md2pdf.toml` in your home directory.

All settings in the file are optional. If omitted, built-in defaults are used.

Configuration Schema Reference

```
[md2pdf]
output_file      = "output.pdf"
theme            = "default"           # Theme name: default, academic, minimal, dark
offline          = false               # true = skip Kroki API calls, render placeholders instead
cache_dir        = "~/.cache/pymd2pdf"
min_image_scale  = 0.8                 # minimum image scale factor before deferring to a new page
toc              = false               # true = generate a Table of Contents page
cover            = false               # true = prepend an auto-generated cover/title page
header           = "{title} | {section}" # Running header template (supports {title} and {section} place
header_on_first_page = false           # true = render running header on the first page
emoji            = true                # true = translate emoji codepoints into Twemoji images

[theme]
# Typography
font_body        = "DejaVuSans"
font_heading     = "DejaVuSans-Bold"
font_mono        = "DejaVuSansMono"

# Custom TTF fonts
# To use your own font, set BOTH the logical name AND the path to the .ttf file.
# font_file_body   = "/path/to/font.ttf"
# font_file_heading = "/path/to/font-bold.ttf"
# font_file_mono   = "/path/to/font-mono.ttf"

# Colors (Hexadecimal)
color_body_text   = "#000000"
color_blockquote_text = "#555555"
color_link        = "#0366d6"
color_hr          = "#cccccc"
color_page_bg     = "#ffffff"           # Background color for document pages

# Tables
color_table_header_bg   = "#2c3e50"
color_table_header_text = "#ffffff"
color_table_grid        = "#cccccc"
color_table_row_odd     = "#ffffff"
color_table_row_even    = "#f5f5f5"

# Blockquotes & Code Blocks
color_blockquote_bar = "#cccccc"
color_code_bg        = "#f5f5f5"
syntax_style         = "default"       # Pygments style name for code blocks
```

YAML Front Matter & PDF Metadata

If your Markdown document starts with a YAML front-matter block, md2pdf automatically parses it and applies the values to the final PDF file metadata.

Example Markdown:

```
---
title: "Quarterly Financial Analysis"
author: "Acme Corp Analytics"
subject: "Q2 Progress and Metrics Report"
keywords: "finance, analytics, report, quarterly"
---

# Q2 Progress Report
...
```

If these fields are missing, the engine falls back to:

- **Title:** The base name of the input file (e.g. README).

- **Author:** "pymd2pdf".
-

Cover Page Generation

When cover page generation is enabled (using CLI `--cover` or setting `cover = true` in `md2pdf.toml`), `md2pdf` automatically generates and prepends a clean, professionally formatted cover/title page.

- **Front-Matter Metadata:** The cover page uses keys declared in the document's YAML front-matter metadata block (`title`, `author`, `date`).
 - **Defaults and Suppression:**
 - The title automatically falls back to the input file name if it is not explicitly provided.
 - To avoid generic library branding, the author name and date are **only** rendered on the cover page if they are explicitly declared in the front-matter.
 - **Layout Interaction:**
 - If both Cover Page and Table of Contents (TOC) are enabled, the cover page will be page 1, followed by the TOC on page 2, with the main content starting on page 3.
 - Running headers, header divider lines, and page numbers are automatically suppressed on the cover page (page 1).
-

Table of Contents (TOC)

When Table of Contents generation is enabled (using CLI `--toc` or setting `toc = true` in `md2pdf.toml`), `md2pdf` automatically inserts a Table of Contents page immediately after the first page of your document.

- The TOC generator scans the document for heading bookmarks (H1 to H6).
 - Standard layout formatting ensures headings are indented by depth level.
 - Clickable, colored hyperlink lines map the heading titles directly to their pages.
 - Page numbers are computed dynamically using a two-pass rendering cycle.
-

Running Headers & Footers

Page Footers

Every page of the compiled PDF displays a centered footer showing the page number in the format `– Page X –`.

Page Headers

Running headers are configurable and disabled by default. When enabled (via CLI `--header` or config `header`), they print at the top left of the page margins, separated by a thin rule.

- **Placeholder Substitution:**
 - `{title}` is substituted with the document title parsed from the YAML front matter or fallback file name.

- `{section}` is substituted with the most recent H1 or H2 heading currently appearing on or before that page.
- **Control Page 1:** The CLI flag `--header-on-first-page` or config `header_on_first_page = true` toggles whether headers render on the first page.

Colour Emoji Support (Twemoji)

md2pdf supports colour emojis inline in prose, list items, headings, and blockquotes. Because standard PDF reader engines and the ReportLab layout engine do not support colour tables in TTF fonts, the engine runs an emoji processor:





- 1 A preprocessor scans the Markdown source for unicode emoji characters.
- 2 Emojis are replaced with an inline HTML `` tag pointing to high-quality Twemoji PNG graphics.
- 3 On the first run, the images are downloaded via HTTPS and stored locally in the cache directory (`~/.cache/pymd2pdf/emoji/`).
- 4 Subsequent runs use the cache directly with zero network requests.
- 5 If the system is run with `--offline`, placeholders are rendered instead of downloading missing graphics.
- 6 To disable emoji rendering completely, pass `--no-emoji` or set `emoji = false` in `md2pdf.toml`.

Download Timeout

Each emoji PNG download enforces a **10-second timeout** by default. If the network hangs or a request exceeds the limit, the original emoji character is kept as a fallback — the pipeline never blocks indefinitely on a slow connection.

Task Lists & Checkboxes

md2pdf supports GFM-style task list checklists:

- **Syntax:**
 - `[]` Uncompleted task list item - `[x]` Completed task list item - `[X]` Completed (case-insensitive)
- **Styling:**
 - If color emoji support is enabled (using the `--emoji` flag or `emoji = true` in config), checkboxes are rendered using Twemoji's checked () and unchecked () box images.
 - If color emoji support is disabled or the images cannot be loaded, checkboxes fall back to using Unicode ballot box symbols ( and ) from the default DejaVu Sans font.

Footnotes

Footnotes allow citing explanations or details at the bottom of the page.

Syntax:

```
Here is a paragraph with a footnote reference[^1] and a second one[^detail].

[^1]: This is the text of the first footnote.
[^detail]: Detailed notes can include formatting like bold text or code.
```

Layout Logic:

- Footnote reference citations (e.g., [1]) are clickable anchors.
- The matching footnote definitions are aligned at the bottom of the page where the reference first appears.
- A two-pass compile cycle is utilized to calculate dynamic content height and prevent text overlap at the bottom page margin.

Inline Formatting

md2pdf supports standard and extended inline formatting options:

- **Strikethrough:** Wrap text in double tildes (`~~strikethrough~~`) to draw a line over it. This compiles to ReportLab `<strike>` tags.
- **Highlight:** Wrap text in double equals (`==highlight==`) to draw a filled background rectangle behind the text. This compiles to ReportLab `` tags.
- **Highlight Color Configuration:** The background color of highlights defaults to yellow (`#ffff00`). This is fully customizable in the `[theme]` section of your `md2pdf.toml` configuration:
`[theme] color_highlight = "#ffff00" # Hex color code for text highlights`
- **Superscript:** Wrap text in single carets (`x^2^`) to raise it above the baseline. This compiles to ReportLab `<sup>` tags natively.
- **Subscript:** Wrap text in single tildes (`H~2~O`) to lower it below the baseline. This compiles to ReportLab `<sub>` tags natively.

File Inclusion (!include)

You can split documents into multiple reusable files and combine them at compile-time using the `!include` directive:

```
# Main Document

Here is some content.

<!-- Included file not found: subdirectory/another_file.md -->
```

Key Behaviors:

- File paths can be absolute or relative to the source document's directory.
- Inclusion is resolved **recursively** (included files can include other files).
- All text styles, formatting, code blocks, math, and diagrams inside the included documents work exactly like they were written in the main document.

Admonitions & GitHub Alerts

md2pdf supports two distinct markdown patterns for highlighting informational callouts and alerts:

1. Fenced Container Admonitions (Obsidian/MkDocs style)

Surround blocks with triple colons (:::) followed by the admonition type. You can supply an optional custom title in quotes:

```
:::warning "Caution Required"
This action might override existing files. Make sure you back up your work!
:::
```

2. Markdown Blockquote Alerts (GitHub-style)

Format alerts using blockquote symbols containing bracketed headers:

```
> [!NOTE]
> This is a standard GitHub-style note alert.
```

Supported Styles and Severity Levels

Each type is parsed into a specific severity styling theme containing unique left borders and background tints:

- **Blue/Slate Theme** (note, info, todo): For general, neutral informational blocks.
- **Green Theme** (tip, success, check): For helpful suggestions, tips, or positive outcomes.
- **Amber/Orange Theme** (warning, attention): For cautionary info or things requiring care.
- **Red Theme** (danger, error, failure, bug, caution): For critical warnings, bugs, errors, or risks of data loss.
- **Teal/Cyan Theme** (important): Highlight critical requirements.
- **Fallback Theme**: Any custom type is rendered using a default theme palette.

Explicit Page Breaks

You can manually control document pagination using either of the following directives:

HTML Comment Syntax

```
<!-- pagebreak -->
```

Backslash Syntax

```
\pagebreak
```

Constraints: Both directives must reside on their own line. Leading/trailing whitespace is allowed, and they are case-insensitive. When parsed, they compile directly to a ReportLab PageBreak flowable, pushing subsequent elements to the top of a new page.

Progress Reporting

For large documents or compilation pipelines containing external resources (e.g., Mermaid diagrams, LaTeX blocks, colour emojis), compilation can take some time. By default, `md2pdf` outputs compilation progress stage-by-stage to `sys.stderr`.

Stages reported:

- 1 **Pre-processing document:** Includes YAML front-matter stripping, recursive include resolution, and pre-scanning/downloading colour emojis.
- 2 **Parsing Markdown:** Internal mistletoe parsing of the document content.
- 3 **Mapping tokens to flowables:** Running element handlers, including rendering Mermaid diagrams and LaTeX math blocks via Kroki.
- 4 **Generating PDF layout:** Dynamic two-pass pagination and layout construction.

Disabling Progress Reporting:

To suppress progress messages, use the `--no-progress` CLI flag:

```
md2pdf input.md --no-progress
```

Programmatic users can pass a `progress_callback` function to the `Pipeline` constructor or the high-level `convert()` function:

```
from md2pdf import convert

def my_callback(event: str, data: dict):
    print(f"Compilation event: {event} -> {data}")

convert("input.md", "output.pdf", progress_callback=my_callback)
```

Tables & Layout Safeguards

`md2pdf` enforces strict rules to prevent bad pagination and document overflow:

Repeating Headers & Row Protection

Tables are compiled such that rows never split mid-line across page boundaries. Headers repeat at the top of every page dynamically.

Heading-to-Table Bonding (`KeepTogetherParts`)

If a section heading is followed immediately by a table, standard PDF layout engines can sometimes push the entire table to the next page if it is too long, leaving a massive gap below the heading.

`md2pdf` solves this by automatically wrapping them in a smart `KeepTogetherParts` container:

- It calculates the height of the heading plus the height of the table's header and first data row.
- If that minimum subset fits at the bottom of the current page, it lets the table start printing there and flow/break naturally to the next page.

- If it does not fit, it moves the heading along with the start of the table to the next page, preventing orphaned headings.

Table Column Alignment

You can control the alignment of table columns using standard Markdown syntax in the separator row:

- **Left Aligned:** :--- (or default ---)
- **Center Aligned:** :---:
- **Right Aligned:** ---:

The alignment settings are automatically parsed and passed through to both the ReportLab `TableStyle` cell properties (via `ALIGN` commands) and individual cell `Paragraph` styles, ensuring text wraps and aligns correctly.

HTML Line Breaks ()

In addition to standard Markdown line breaks (two trailing spaces or a backslash), md2pdf natively processes raw HTML line breaks inside inline text (including paragraph content, headings, and table cells).

Tags like

,

, or

are automatically normalized into PDF-compatible line breaks (), allowing explicit vertical spacing controls inside table cell text.

Images & Captions

md2pdf supports standard Markdown images (`![alt](path)`) and HTML `` tags, implementing layout safeguards and styling enhancements:

- **Image Auto-scaling:** Images are dynamically scaled down to fit within the printable page margins.
- **Image Captions:** If an image includes an alt-text caption (e.g. `![Figure 1: Pipeline Flow](flow.png)`), the engine automatically extracts the alt-text and renders it as a small, centered, and styled caption paragraph immediately below the image.
- **Layout Safeguards:** Tall images are scaled down automatically. If a block image cannot fit at the bottom of the current page, the layout engine shifts it to the top of the next page to prevent layout overlap or clipping.
- **Missing Image Placeholders:** If an image path is corrupt or missing, a clean placeholder box is rendered instead of throwing a compiler error.

Diagrams (Mermaid) & Math (LaTeX)

Diagrams and complex equations can be compiled locally or remotely, and are embedded as cropped high-resolution graphics in the PDF.

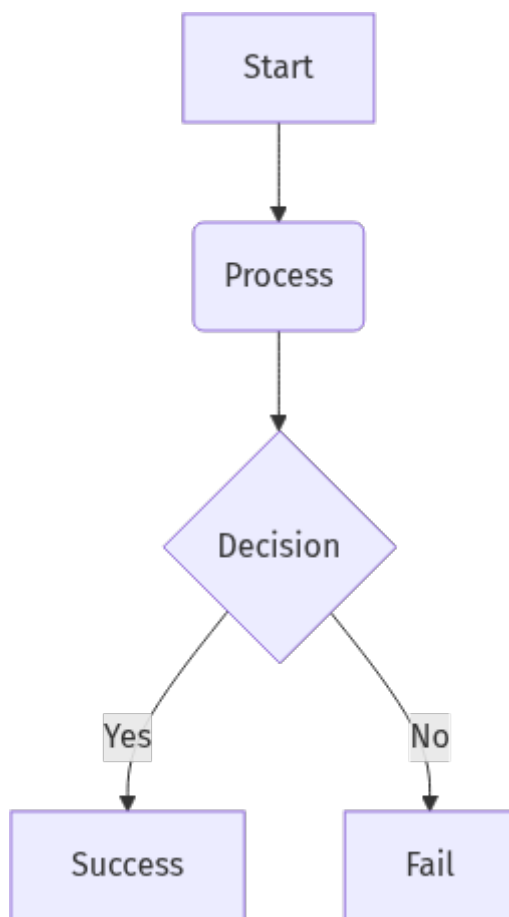
Optional Local LaTeX Rendering (Matplotlib)

By default, the engine compiles LaTeX equations remotely via the Kroki API. However, you can opt-in to fast, 100% offline local rendering using Matplotlib's MathText engine:

- 1 Install the optional dependency extra:

```
pip install "pymd2pdf[matplotlib]"
```
- 2 When installed, `md2pdf` automatically renders standard math equations (like $E = mc^2$) locally in milliseconds, with no network requests.
- 3 If the formula contains unsupported environments (like `\begin{cases}` or TikZ code) or if Matplotlib is not installed, it automatically and gracefully falls back to Kroki.

Mermaid Syntax



LaTeX Math Syntax

Use double dollar signs ($\$ \$$) to define block equations:

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2}$$

Caching and Layout Protection:

- **Concurrent Pre-fetching:** To avoid sequential request bottlenecks on formula-heavy documents, the pipeline scans the parsed AST and pre-populates the cache for all diagrams and math formulas concurrently in a thread pool of 5 workers before document mapping.
- **SHA-256 Caching:** Script inputs are hashed using SHA-256. If a diagram has already been compiled, it is read directly from disk with zero network latency.
- **Margin Trimming:** Graphics are automatically cropped to eliminate unnecessary white margins before embedding.
- **1-Retry Limit:** Remote Kroki compilation requests will retry exactly once on failure (2 attempts total).
- **Robust Fallback:** If math rendering fails or is offline, inline math falls back to showing raw LaTeX inline, and block math falls back to displaying the raw LaTeX within a monospace block (`Preformatted`) instead of a generic gray placeholder box.
- **KeepTogether Bond:** Elements are bound to their preceding heading. If the graphic cannot fit at the bottom of the page, the header and the graphic move together, avoiding orphans.

Unicode & Fonts

md2pdf bundles the **DejaVu Sans** font family directly within the package. This ensures broad out-of-the-box Unicode coverage on all operating systems without requiring system-installed fonts.

Supported Unicode blocks include:

- **Latin Extended:** café, naïve, résumé, Ångström, Zürich, façade, Łódź.
- **Greek Alphabet:** Both uppercase and lowercase characters (α β γ ... A B Γ).
- **Cyrillic Script:** Russian, Ukrainian, Bulgarian, Serbian.
- **Math Operators:** \sum \prod \int ∂ $\sqrt{\quad}$ $\sqrt[3]{\quad}$ ∞ \emptyset \in \leq \geq \neq \approx \equiv .
- **Arrows:** \leftarrow \rightarrow \uparrow \downarrow \leftrightarrow \Leftrightarrow \Uparrow \Downarrow .
- **Box Drawing:** Vertical and horizontal rules used to build ASCII diagrams.
- **Currency:** \$ € £ ¥ ₹ ₪ ₩ ₪ ₪ ₪ ₪.
- **Typographic Symbols:** Em dash (—), ellipses (...), curly quotes (“ ”), registered (®), trademark (™).

Custom Font Validation

When you configure a custom TTF font via `font_file_body`, `font_file_heading`, or `font_file_mono` in the `[theme]` section, md2pdf performs a **pre-flight path check** before attempting to register the font with ReportLab. If the file does not exist on disk, a `ConfigError` is raised immediately with a clear diagnostic message:

```
ConfigError: [theme] font_file_body points to a missing font file: /path/to/MyFont.ttf
Please verify the path exists and is a valid TrueType (.ttf) file.
```

This prevents the cryptic low-level ReportLab errors that would otherwise surface mid-pipeline when font registration fails.

Themes & Styling System in md2pdf

md2pdf uses a centralized theme system to control colors, fonts, and spacing across all document elements. The stylesheet registry compiles layers of styles into a single unified stylesheet dictionary before rendering.

Built-In Themes

md2pdf bundles several pre-built themes selectable via the CLI `--theme / -t` flag or the `theme` configuration setting in `md2pdf.toml`:

- **default**: The standard layout using DejaVu fonts, a corporate dark-blue table header, gray grid, and default spacing metrics.
- **academic**: A formal serif layout using Times-Roman and Times-Bold fonts, Courier for code, slightly larger font sizes/spacing, and formal dark table styling.
- **minimal**: A clean, modern sans-serif layout using Helvetica and Helvetica-Bold fonts, generous line margins, thick black blockquote accent bar, black hyperlinks, and plain white table styling.
- **dark**: A dark mode layout painting a dark charcoal background (`#1e1e1e`) across all pages, light-gray body text, soft-blue hyperlinks, dark table structures, and pygments syntax highlighting styled via `monokai`.

The ThemeConfig Dataclass

The styling system is parameterized by `ThemeConfig`, which is the single source of truth for colors and fonts. Sensible defaults are provided so the theme section is completely optional.

Here is the default schema with its default settings:

```
[theme]
# --- Typography ---
font_body          = "Helvetica"
font_heading       = "Helvetica-Bold"
font_mono          = "Courier"
font_size_body     = 10
font_size_small    = 9
spacing_base       = 8

# --- Prose Colors ---
color_body_text    = "#000000"
color_blockquote_text = "#555555"
color_link         = "#0366d6"
color_hr           = "#cccccc"

# --- Table Colors ---
color_table_header_bg = "#2c3e50"
color_table_header_text = "#ffffff"
color_table_grid      = "#cccccc"
color_table_row_odd   = "#ffffff"
color_table_row_even  = "#f5f5f5"

# --- Blockquote Accent Bar ---
color_blockquote_bar = "#cccccc"

# --- Code Highlighting ---
color_code_bg       = "#f5f5f5"
syntax_style        = "default"
```

```
# --- Page Background ---
color_page_bg = "#ffffff"
```

Base Stylesheet Keys

The default base stylesheet built from `ThemeConfig` contains the following keys, which map to ReportLab `ParagraphStyle` objects or raw scalar values:

Key	Type	Description
h1	ParagraphStyle	Large document heading style (H1).
h2	ParagraphStyle	Section heading style (H2).
h3	ParagraphStyle	Subsection heading style (H3).
h4	ParagraphStyle	Minor heading style (H4).
body	ParagraphStyle	Normal body paragraphs.
blockquote	ParagraphStyle	Indented text for blockquotes.
list_item	ParagraphStyle	Text within lists.
code_inline	ParagraphStyle	Monospace inline code fragments.
table_header	ParagraphStyle	Bold column headers inside tables.
table_cell	ParagraphStyle	Normal text inside table cells.
table_style	list	ReportLab table styles (backgrounds, padding, grid lines).
color_hr	HexColor	Color for <code><hr></code> thematic breaks.
color_link	str	Color for hyperlinks (used in XML markup tag rendering).
color_blockquote_bar	HexColor	Left vertical rule color for blockquotes.
code_block	ParagraphStyle	Multi-line syntax-highlighted code block style.
footnote	ParagraphStyle	Text style for footnote definitions.
color_code_bg	HexColor	Background color for code blocks.
syntax_style	str	Name of Pygments syntax highlighting theme.
spacing_base	int	Base spacing metric in points.
color_page_bg	HexColor	Background color for document pages.

All paragraph styles include:

- `allowWidows = 0`
- `allowOrphans = 0`

These ensure that single lines of paragraphs are never orphaned at the top or bottom of page boundaries.

Styling Override Plugins

You can register styling layers by implementing a class with a `get_stylesheet()` method and registering it under the `md2pdf.stylesheets` entry-point group.

A style layer returns a partial dictionary. Later layers are merged with earlier layers, overriding keys:

```
class ElegantLawTheme:
    def get_stylesheet(self) -> dict:
        return {
            "font_body": "Times-Roman",
            "font_heading": "Times-Bold",
            "color_body_text": "#1a1a1a",
            "color_link": "#000080",
        }
```

Register this layer in your plugin package's `pyproject.toml`:

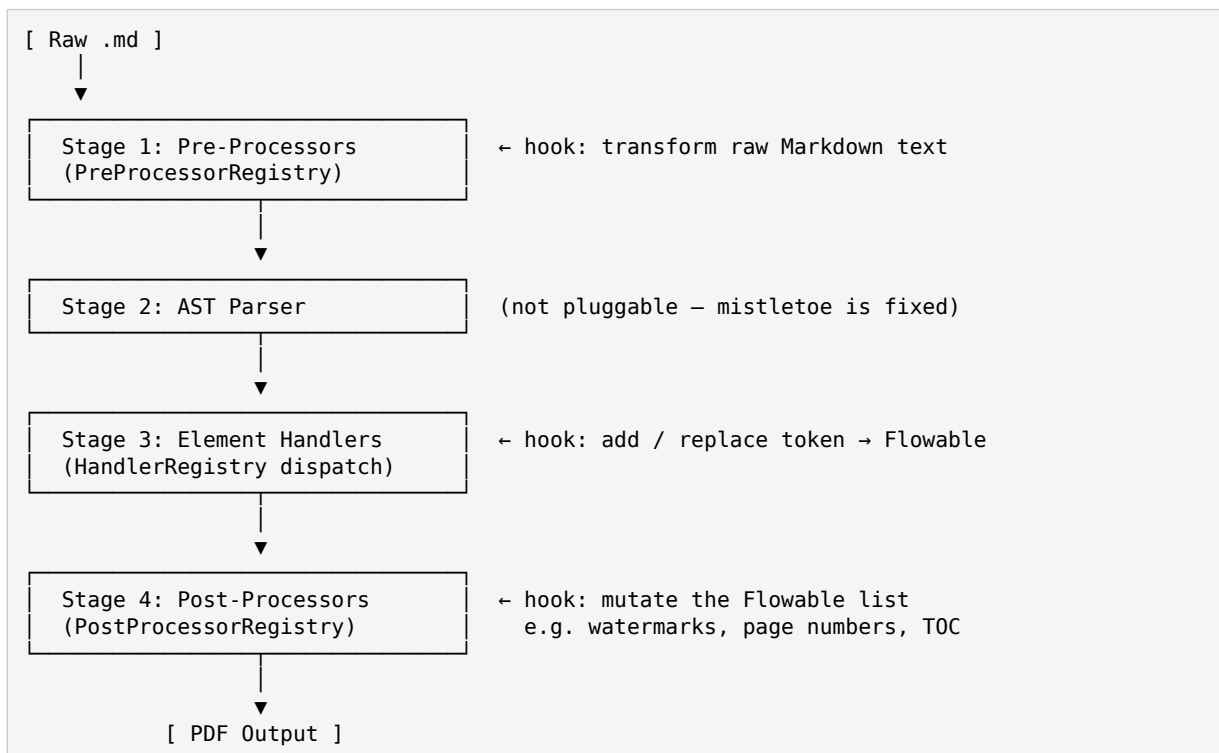
```
[project.entry-points."md2pdf.stylesheets"]
law_theme = "my_plugin.themes:ElegantLawTheme"
```

Plugin Authoring Guide for md2pdf

This document describes the public API available to third-party plugin authors, the four hook stages, and how to publish a plugin package.

Overview

md2pdf processes a Markdown document through four sequential stages. Plugins can intercept at any of the first three stages, plus supply stylesheet overrides:



Additionally, plugins can supply a **stylesheet layer** that is merged with the default styles before any handler sees the `styles` dict.

Public API

Only the following modules are part of the **public, stable API**. Do not import from `md2pdf.core.*` private internals.

Module	Public symbols
<code>md2pdf.core.registry</code>	<code>ElementHandler</code>
<code>md2pdf.core.preprocessors</code>	<code>PreProcessor</code> , <code>PreProcessorRegistry</code>
<code>md2pdf.core.postprocessors</code>	<code>PostProcessor</code> , <code>PostProcessorRegistry</code>
<code>md2pdf.core.styles</code>	<code>StyleRegistry</code>

Hook 1 — Pre-Processor

Subclass `PreProcessor` and implement `process(raw_md: str) -> str`.

```
from md2pdf.core.preprocessors import PreProcessor

class MyPreProcessor(PreProcessor):
    def process(self, raw_md: str) -> str:
        # Replace all occurrences of ":warning:" with Δ
        return raw_md.replace(":warning:", "Δ")
```

Priority: Declare under `md2pdf.preprocessors` entry points. The `register()` call accepts a `priority` keyword (default 50). Built-ins run at priorities 10 and 20, so plugin pre-processors at priority 50 always run after them.

Hook 2 — Element Handler

Subclass `ElementHandler`, set `token_type`, and implement `render()`.

```
from md2pdf.core.registry import ElementHandler
from reportlab.platypus import Paragraph
from reportlab.lib.styles import getSampleStyleSheet

class CalloutHandler(ElementHandler):
    """Render custom ``Callout`` token blocks as styled paragraphs."""

    token_type = "Callout"

    def render(self, token: dict, styles: dict) -> list:
        text = token.get("raw", "")
        style = getSampleStyleSheet()["Normal"]
        return [Paragraph(f" {text}", style)]
```

The `styles` dict is the merged stylesheet (defaults + plugin layers). You can add your own style keys to it via a stylesheet plugin (Hook 4).

Last-writer-wins: registering a handler for a `token_type` that already has one (e.g. "Heading") replaces the built-in handler.

Hook 3 — Post-Processor

Subclass `PostProcessor` and implement `process(doc: SimpleDocTemplate, flowables: list) -> list`.

```
from reportlab.platypus import SimpleDocTemplate
from md2pdf.core.postprocessors import PostProcessor

class WatermarkPostProcessor(PostProcessor):
    """Stamps 'DRAFT' diagonally on every page."""

    def __init__(self, text: str = "DRAFT") -> None:
        self.text = text

    def process(self, doc: SimpleDocTemplate, flowables: list) -> list:
        # Attach an onPage callback to the document.
        text = self.text

    def _stamp(canvas, doc):
        canvas.saveState()
```

```

        canvas.setFont("Helvetica-Bold", 72)
        canvas.setFillAlpha(0.1)
        canvas.translate(doc.pagesize[0] / 2, doc.pagesize[1] / 2)
        canvas.rotate(45)
        canvas.drawCentredString(0, 0, text)
        canvas.restoreState()

    doc.md2pdf_on_later_pages = _stamp # type: ignore
    return flowables

```

Post-processors run in **registration order**. The flowables list returned by each processor is passed as input to the next.

Hook 4 — Stylesheet Override

Implement a class with a `get_stylesheet() -> dict` method. Return a dict whose keys override the default stylesheet values.

```

class DarkTheme:
    def get_stylesheet(self) -> dict:
        return {
            "color_body_text": "#e0e0e0",
            "color_link": "#80bfff",
            "font_size_body": 11,
        }

```

Entry-Point Declaration

In your plugin package's `pyproject.toml`:

```

[project.entry-points."md2pdf.handlers"]
callout = "my_package.handlers:CalloutHandler"
timeline = "my_package.handlers:TimelineHandler"

[project.entry-points."md2pdf.preprocessors"]
emoji = "my_package.preprocessors:EmojiPreProcessor"

[project.entry-points."md2pdf.postprocessors"]
watermark = "my_package.postprocessors:WatermarkPostProcessor"

[project.entry-points."md2pdf.stylesheets"]
dark_theme = "my_package.themes:DarkTheme"

```

After installation (`pip install my-plugin-package`), md2pdf auto-discovers and loads the plugin on every run — no config file changes needed.

Config-File Alternative

If you cannot use entry points (e.g. local scripts), declare plugins in

`md2pdf.toml`:

```

[plugins]
handlers = [
    "my_package.handlers:CalloutHandler",
]
preprocessors = [
    "my_package.preprocessors:EmojiPreProcessor",
]
postprocessors = [

```

```
] "my_package.postprocessors:WatermarkPostProcessor",
```

Config-file plugins are loaded **after** entry-point plugins, so they can override built-ins declared via entry points.

Error Handling Contract

- **All plugin loading errors are caught and logged** — a bad plugin never aborts a conversion run.
- Errors appear at `ERROR` level in the `md2pdf.core.plugin_loader` logger.
- Enable logging in your application to see plugin load failures:

```
import logging
logging.basicConfig(level=logging.DEBUG)
```

Rules for Plugin Authors

- 1 **Only subclass public ABCs** (`ElementHandler`, `PreProcessor`, `PostProcessor`). Do not subclass internal implementation classes.
- 2 **Return valid `Flowable` instances** from `ElementHandler.render()`.
- 3 **Never raise from `render()` or `process()`** — return a fallback value instead. An unhandled exception will propagate and crash the conversion.
- 4 **Declare dependencies** in your `pyproject.toml` — do not assume md2pdf's dependencies (`reportlab`, `mistletoe`, `requests`) are available beyond their published API.
- 5 **Pin to `md2pdf >= 0.1.0`** — the public API is stable from this version.

[SetextHeading block – not implemented]

title: "md2pdf Feature Showcase & Rendering Test"
author: "pymd2pdf"
date: "June"

md2pdf Feature Showcase & Rendering Test

This document serves as both a user guide and a comprehensive rendering test suite for the md2pdf typesetting engine. It includes all supported Markdown and layout elements to verify visual layout quality, vertical spacing, and formatting.

1. Quick Start Guide

md2pdf is a pure-Python Markdown-to-PDF compiler that translates standard Markdown and extended features (like LaTeX equations and Mermaid diagrams) directly into print-ready PDF files using ReportLab.

Command Line Interface

Convert this document to a PDF:

```
md2pdf docs/showcase.md -o docs/showcase.pdf
```

Enable validation checks without writing a PDF:

```
md2pdf docs/showcase.md --validate-only
```

Run in offline mode to use placeholders for network-dependent elements (e.g. diagrams):

```
md2pdf docs/showcase.md -o docs/showcase.pdf --offline
```

Programmatic Python API

```
from md2pdf import convert, Config, Pipeline

# Direct conversion
convert("docs/showcase.md", "docs/showcase.pdf")

# Custom pipeline settings
config = Config(
    input_file="docs/showcase.md",
    output_file="docs/showcase.pdf",
    offline=False,
    min_image_scale=0.75
)
pipeline = Pipeline(config)
pipeline.run(raw_md="# My Doc\nSome text.")
```

2. Heading Typography Hierarchy

Below is the vertical stack of headings from H1 through H6. Heading levels 5 and 6 fall back to the H4 style for clean presentation since standard PDF engines support up to 4 heading levels out-of-the-box.

Heading Level 1 (H1)

Heading Level 2 (H2)

Heading Level 3 (H3)

Heading Level 4 (H4)

Heading Level 5 (H5 - fallback to H4 style)

Heading Level 6 (H6 - fallback to H4 style)

3. Inline Elements and Styles

This section tests inline layout and inline style parsing:

- **Strong/Bold:** This is a **strongly emphasized text block** using double asterisks.
 - **Emphasis/Italic:** This is an emphasized text block using single asterisks.
 - **Nested Styles:** You can combine styles like **bold-italic nested runs**.
 - **Code Inline:** Use backticks for monospace symbols like `Pipeline`, `ThemeConfig`, or `build_default_stylesheet()`.
 - **Hyperlinks:** Clickable links like the [md2pdf GitHub page](#) are automatically colored.
 - **Strikethrough:** This is a ~~strikethrough text run~~ showing a horizontal line through text.
 - **Highlight:** This is a **highlighted text run** showcasing text with a yellow background.
 - **Superscript:** This is a superscript text run: $x^2 + y^2 = r^2$.
 - **Subscript:** This is a subscript text run: H_2O , CO_2 .
 - **Footnotes:** Clickable footnote references¹ linked to their definitions² at the bottom of the page.
-

4. Lists

md2pdf supports ordered, unordered, and multi-level nested lists.

Unordered Bullet List

- Top-level item A
- Top-level item B
 - Nested sub-item B.1
 - Nested sub-item B.2
 - Deeply nested sub-item B.2.a

¹ This is the first footnote definition, explaining reference 1.

² This is the second footnote definition, containing inline formatting like **bold text** and ~~strikethrough~~.

- Top-level item C

Ordered Numbered List

- 1 First step in the instructions
- 2 Second step in the instructions
 - 1 Sub-step 2.a
 - 2 Sub-step 2.b
- 3 Final step

Task List Checkboxes

- ☐ Uncompleted task list item
 - ☒ Completed task list item (lowercase)
 - ☒ Completed task list item (uppercase)
 - ☐ Task list item with a link: [md2pdf GitHub page](#)
 - ☐ Task list item with inline code: `x = 1`
-

5. Blockquotes

Blockquotes are styled with a left vertical accent bar and an indented block format.

"Simplicity is the ultimate sophistication."

— Leonardo da Vinci

This is a multi-paragraph blockquote. It maintains indentation and the vertical border across all blocks.

Here is the second paragraph inside the same blockquote.

6. Admonition & Callout Blocks

md2pdf supports Obsidian/MkDocs style fenced admonitions (`:::type`, etc.) and GitHub Markdown alerts (`> [!TYPE]`, etc.) with customizable titles and distinct border/background styling.

Fenced Admonitions

Blue / Slate Themes (`note`, `info`, `todo`)

Note

This is a standard `note` callout block.

Information

This is an `info` callout block with a custom title.

Task Checklist

This is a `todo` callout block for keeping track of tasks.

Green Themes (`tip`, `success`, `check`)

Pro Tip

This is a `tip` callout block to share helpful suggestions.

Completed successfully

This is a `success` callout block indicating successful completion.

Status Checked

This is a `check` callout block verifying a positive status.

Amber / Orange Themes (`warning`, `attention`)

Potential Risk

This is a `warning` callout block indicating a warning condition.

Attention Required

This is an `attention` callout block for items needing care.

Red Themes (`danger`, `error`, `failure`, `bug`, `caution`)

Critical Danger

This is a `danger` callout block for serious threats or data loss alerts.

Error Encountered

This is an `error` callout block showing critical execution issues.

Build Failed

This is a `failure` callout block indicating a failed pipeline run.

Active Bug

This is a `bug` callout block to highlight software defects.

Precautionary Note

This is a `caution` callout block with red theme styling.

Teal / Cyan Theme (`important`)

Crucial Note

This is an `important` callout block highlighting key details.

Unknown Fallback Theme

Custom Type

This is a fallback custom type admonition showing the default theme color.

GitHub-style Markdown Alerts

Note

This is a GitHub-style `NOTE` alert block.

Tip

This is a GitHub-style `TIP` alert block.

Important

This is a GitHub-style `IMPORTANT` alert block.

Warning

This is a GitHub-style `WARNING` alert block.

Caution

This is a GitHub-style `CAUTION` alert block.

7. Code Blocks (Syntax Highlighting)

Monospaced code blocks are typeset with syntax highlighting powered by Pygments (theme customizable via the stylesheet configuration).

```
import os
from reportlab.platypus import Paragraph

def hello_world(name: str) -> None:
    # A simple hello world script to verify syntax colors
    message = f"Hello, {name}!"
    print(message)
```

8. Tables

Tables split cleanly across page boundaries. Table columns automatically distribute width evenly across the printable area, and the header repeats at the top of every page.

Parameter	Type	Default	Description
font_body	str	"DejaVuSans"	Body typeface.
font_heading	str	"DejaVuSans-Bold"	Heading typeface.
spacing_base	int	8	Base vertical spacing metric.
color_link	str	"#0366d6"	Hex color string for links.

Table Alignment Showcase

Below is a table demonstrating left, center, right, and default column alignments:

Left Aligned	Center Aligned	Right Aligned	Default Aligned
Text Left	Text Center	Text Right	Text Default
Left text long	Center text long	Right text long	Default text long

9. Diagrams (Mermaid)

Mermaid diagram blocks are rendered as cropped PNG images using the Kroki API and cached locally.



10. Math & Equations (LaTeX)

LaTeX math expressions are compiled to transparent images and centered:

$$f(x) = \int_{-\infty}^{\infty} \hat{f}(\xi) e^{2\pi i x \xi} d\xi$$

11. Images & Placeholders

Standard images are scaled down automatically to fit within the printable page area.

Missing Image Placeholder

If an image file is missing or corrupt, a placeholder box is rendered instead of throwing a compiler crash error:

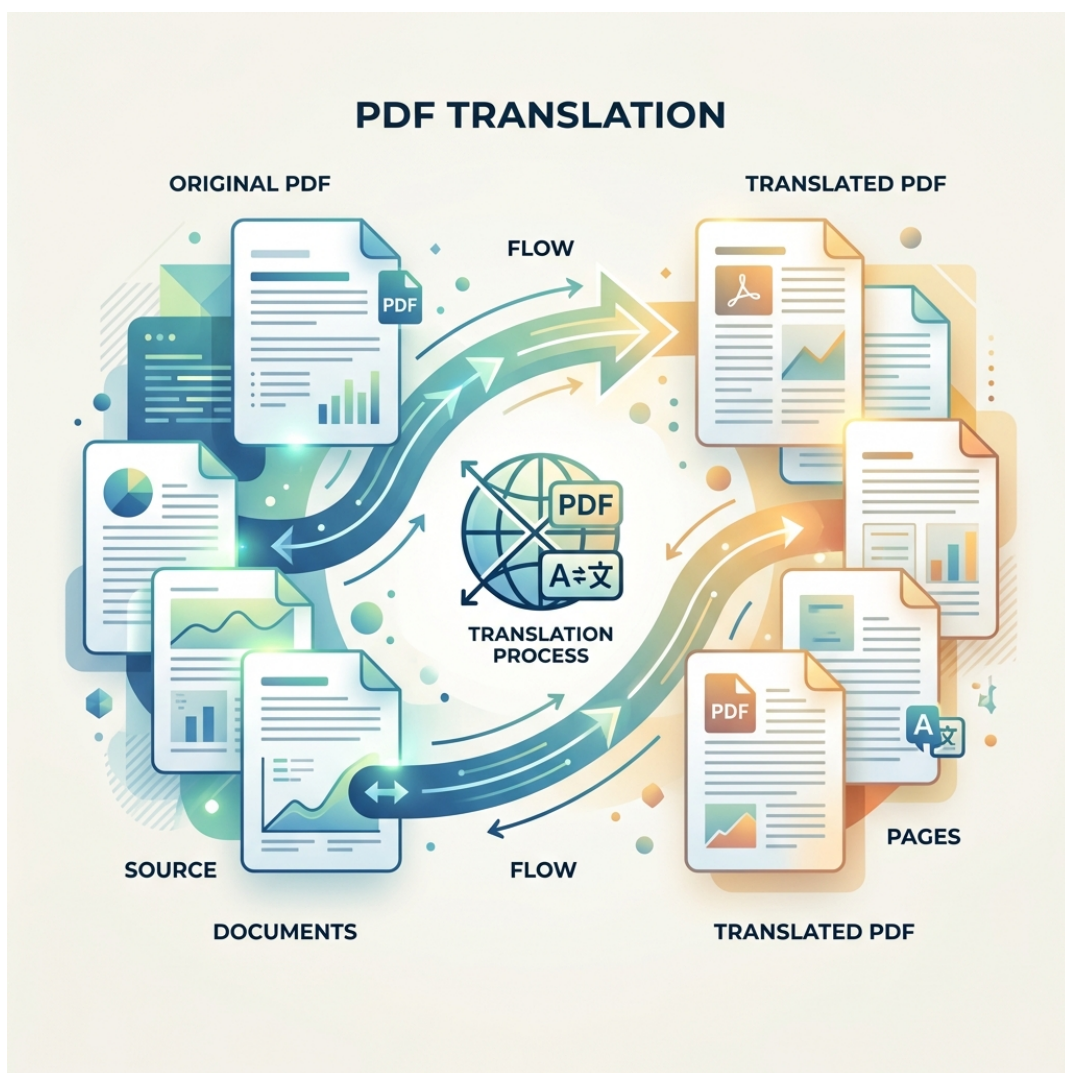
[image diagram — offline / render failed]

Missing: missing_image.png

Missing Image Example

Working Image Example

Successful image references are dynamically loaded, scaled, and centered:



Sample Document Translation Graphic

Typographic Punctuation

Em dash —, en dash –, ellipsis ..., curly quotes " ' ', guillemets « »,
bullet •, section §, pilcrow ¶, dagger †, double dagger ‡,
trademark ™, copyright ©, registered ®, degree °

Superscript & Subscript Lookalikes

$$\text{H}_2\text{O}, \text{CO}_2, E = mc^2, x^2 + y^2 = r^2, A_n \rightarrow B_n$$

Custom Font via md2pdf.toml

To use a different font, add the following to your `md2pdf.toml`. The engine registers the TTF file automatically — no plugin or Python code required:

```
[theme]
font_body      = "NotoSans"
font_file_body = "/path/to/NotoSans-Regular.ttf"

font_heading   = "NotoSans-Bold"
font_file_heading = "/path/to/NotoSans-Bold.ttf"

font_mono      = "NotoSansMono"
font_file_mono = "/path/to/NotoSansMono-Regular.ttf"
```

Tilde expansion (~ / fonts / MyFont . ttf) is supported.

Colour Emoji (Twemoji) 🕶️

Emoji codepoints are automatically detected and replaced with full-colour **Twemoji** PNG images,
cached locally on first use. No configuration required — it works out of the box.

Single emoji: 🌐🚀🎉✅❌🔥💡🧠🐸📄

Emoji in prose: The build passed 🎉 and all tests are green ✅. Deploy to production 🚀?

Faces and people: 😊 🕶️ 🤔 🧐 🙌 🙏 🍷 👍 🤝 🧑

Nature and travel: 🌸 🌊 🏔️ 🌅 🌙 ⭐ 🌈 🦋 🌿 🍀

Food and objects: 🍕 🍵 🎸 📦 🔑 📌 📁 📊 🔬 🛠️

Emoji mixed with code references: Use `convert()`  and Pipeline  together.

To disable emoji substitution, pass `--no-emoji` on the CLI or set `emoji = false` in `md2pdf.toml`:

```
md2pdf docs/showcase.md --no-emoji
```

```
[md2pdf]
emoji = false
```

13. Colour Emoji Showcase





This section verifies end-to-end rendering of emoji characters across multiple contexts.

Inline Paragraph Emoji





All systems nominal  — pipeline complete . Found **3 warnings**  and **0 errors** .

Emoji in Headings and Lists

Today's Task List

-  Run the full test suite
-  Update the changelog
-  Tag and publish the release
-  Celebrate!

Status Summary



Component	Status
Core pipeline	 OK
Emoji support	 OK
LaTeX rendering	 OK
Mermaid diagrams	 OK

ZWJ Sequences

Complex multi-codepoint emoji sequences (joined with a Zero-Width Joiner) are handled as a single unit so they map to the correct Twemoji image:

 (person: technologist) ·  (family) ·  (rainbow flag)

Emoji in Blockquotes

 **Key insight:** Colour emoji in PDFs previously required replacing the entire ReportLab rendering backend. The Twemoji approach slots in as a lightweight pre-processor with zero pipeline changes — emoji PNGs are cached after the first download .

14. File Inclusion (!include)

You can split documents into multiple reusable files and combine them at compile-time:

This text is imported dynamically from `docs/included_feature.md` using the new `!include` feature. It demonstrates that formatting, styles, and other features (like inline math $E = mc^2$) work perfectly inside included documents.

15. Page Breaks

You can manually control pagination using either the standard HTML comment directive or a custom backslash directive. This compiles directly into a PDF PageBreak flowable.

HTML Comment Syntax

```
<!-- pagebreak -->
```

Backslash Syntax

```
\pagebreak
```

Both directives must reside on their own line (leading/trailing whitespace is allowed, and they are case-insensitive).

16. Cover Page Generation

You can automatically prepend a beautifully formatted cover/title page before the Table of Contents by passing the `--cover` CLI flag:

```
md2pdf docs/showcase.md -o docs/showcase.pdf --cover
```

The cover page is generated dynamically using keys declared in the document's YAML front-matter metadata:

- `title`: Extracted and displayed in a large, bold format. Defaults to the source filename if not explicitly provided.
- `author`: Rendered only if explicitly declared in the front-matter metadata (omitting the default library branding).
- `date`: Rendered only if explicitly declared in the front-matter metadata.

When a cover page is generated, page-level running headers, header rules, and footers (page numbers) are automatically suppressed on the first page.