makepdf

User Manual

Introduction

This document contains instructions for use of the `makepdf` tool. The source for this document is available in the `sample/` folder of the project's repository on Github.

The `makepdf` software package uses the pdfmake package, which is itself based on pdfkit. All packages are available under an open source license, and are available for installation through NPM.

Installation

Generating PDF files with makepdf requires installing Node.JS and the accompanying NPM package manager. Editing source files (Markdown) does not require Node.JS or NPM.

Running makepdf. To generate a PDF file from Markdown files in the current folder, run the following command in a Command Prompt (Windows) or Terminal (Mac OS/Linux) window:

```
npx makepdf
```

This looks for `makepdf` on NPM, and runs the code locally. By default, the `index.md` file is converted, but it's recommended to create a configuration file to customize input and output settings (see **Configuration**). Configuration files with the name `makepdf.json` will be used automatically, but you can also pass the name of a configuration file by appending a file name to the end of the command above.

Installation as a global executable. This method makes the `makepdf` command available without the use of `npx`.

To install `makepdf` on your computer, run the following command:

```
npm install -g makepdf
```

Now you can create PDF files using the `makepdf` command, with an optional parameter for the configuration file name.

Configuration

Project configuration files are usually named `makepdf.json`, but any other file name can be passed to the `makepdf` command, including files located in other folders.

The configuration is stored using the JSON file format — either as a single object, or as an array of objects for multiple output files. For an example configuration file, refer to the `sample/` folder of the Github repository.

The following options are available. None or required, but leaving out the 'info' values results in a warning.

Option	Description
skip	**boolean** True if this document should not be generated (useful if the configuration file contains an array for multiple documents).
define	**object** Key-value (string) map of texts that can be inserted using the `\\insert` tag.
input.baseDir	Path name for the directory that contains all other files (if not the same as the directory that contains the configuration file itself).
input.entry	Path and/or file name for the Markdown file that is converted. Defaults to `index.md`.
output.debug	**boolean** True if an additional file called `debug.json` should be generated, which contains the document structure that is sent to `pdfmake`.
output.file	Path and file name for the resulting PDF file.
output.info	**object** Key-value (string) map for meta data that gets saved to the PDF file. This includes `title`, `author`, `subject`, and `keywords`.
output.footerStartPage	First page (1-n) that should include a footer with the document title and subject, and page number.
output.footerEndPage	Last page (1-n, *or* negative value) that should include a footer. Negative values indicate pages *before* the end, i.e. -1 is the last page, -2 the page before that, etc.

Option	Description
output.tocLevel	Lowest heading level (1-n) to be included in the automatically generated Table of Contents.
output.autonumSuffix	Suffix to be appended to all automatically generated numbers. Defaults to a single space, but some might prefer a dot instead.
output.pageSize	**object** with `height` and `width` properties (in points, i.e. 1/72"), for the total size of the generated page(s). May also be set to a 'named' page size, e.g. `"A4"` (as a string). The default page size is 6" by 9" for improved readability on screens, or printing at the size of a pocket-sized book.
output.pageOrientation	Either `portrait` (default) or `landscape`.
output.pageMargins	**array** (left, top, right, bottom values), page margins in points.
output.footerMargins	**array** (left, top, right, bottom values), margins around the footer area, within the page area. The top value is used for the offset between text and footer, the bottom value is usually 0.
fonts	**object** Key-object map that defines a font by name, and lists font *files* for different styles (glob, e.g. `"fonts/**/Arial.ttf"` to find the Arial font within any folder that is located within the `fonts` folder). Styles should include `normal`, `bold`, `italics`, and `bolditalics`. Default font names include "Headings", "Headings Bold", "Headings Light", "Body", "Body Bold", "Body Light", "Monospaced", and "Symbol".
styles	**object** Key-object map that defines text styles by name, and lists properties for each style. All properties are passed to the `pdfmake` library directly, refer to its documentation for more information. The default set is defined in the `src/config/styles.ts` file in the makepdf Github repository.

Inline Formatting

Makepdf supports inline Markdown formatting, as well as a number of extensions that are mostly based on LaTeX-like syntax.

Automatic replacements

Some punctuation and other characters are replaced to ‘typographic’ versions automatically. This includes the following characters:

- Straight double quotes are replaced with “curly quotes”.
- Straight single quotes and apostrophe (e.g. in **it’s**) are also replaced with ‘curly quotes’.
- Double dash `--` is replaced with an em-dash — like so.
- Three dots `...` are replaced with a single character ‘…’ for better spacing.

Additionally, some special characters can be inserted using special codes:

- Use `\\newline` to insert a newline character. Alternatively, add a single backslash character `\` to the very end of a line.
- Use `\~` to insert a non-breaking space.

Special characters `` `\*_{}[]()<>#+-.!'" `` can be inserted verbatim by preceding them with a backslash. For example the text C:\* should be written as `C:\\\*`, *except* if this text occurs within **Code** (see below). Similarly, to display measurements in feet and inches (6'10"), write 6\'10\".

Automatic numbering is supported within inline text as well, refer to **Automatic numbering**.

Bold, italics, strikethrough

Text can be made **bold**, set in *italics* font, or ~~struck through~~ using ‘paired’ characters around the text.

```
This is **bold** style.
This is _italics_ style.
This is ~~strikethrough~~ style.
```

Note that the symbols for bold and italics are interchangeable: asterisks and underscores will both work. A single pair changes to italics font, a double pair changes to bold font. These can be used ***together*** as well, even including ***~~strikethrough~~***.

```
This is **_really_ important**.
```

Code

Surround inline code with backticks (i.e. `` `code` ``) to set it in a monospaced font, and disable parsing of tags and automatic replacements within this text.

```
Boolean values are either `true` or `false`.
```

For blocks of code, refer to **Pre-formatted blocks (code)**.

Links

External links (i.e. URLs) can be inserted using Markdown syntax. Most PDF readers will make these 'clickable', and the link text will be styled using the `link` style, which can be overridden using the `style` object in the configuration file.

For example, a link to the Google home page can be inserted as `[Google](https://www.google.com)`, which results in the following link: Google.

Internal (document) links can be inserted as well, refer to **References & TOC**.

LaTex-style tags

Alternatively, you can use LaTeX-style tags for inline styling, which enable a few more possibilities for styling parts of your text.

```
This is \\bold{\\italics{really} important}.
```

The following tags are available.

- `\\bold{...}` or `\\b{...}` for **bold text**.
- `\\italics{...}` or `\\i{...}` for *italics*.
- `\\underline{...}` for underlined text.

- `\\overline{...}` for overlined text.
- `\\linethrough{...}` for ~~struck-out text~~.
- `\\code{...}` for inline `code` (different from `` `code` `` since the text inside this tag is still parsed for other tags or markup).
- `\\color(red){...}` for colored text. Some colors are available by name, but you can also use CSS-style colors like `#0022FF`.
- `\\style(name){...}` to apply a predefined style to a part of the text, e.g. link style (not a link), or **Heading 4**.

Options for underline, overline, and strikethrough can be specified in brackets. This includes a line 'style' (i.e. `solid`, `dashed`, `dotted`, `wavy`, or `double`) followed by an optional color.

For example, `\\underline(wavy red){this}` results in this.

Verbatim

To include text in the output without parsing it at all (such as for the examples shown in this document), you can use the `\\verbatim+...+` tag, or the shorter `\\v+...+` version. The plus symbol can be changed to any other character, as long as it doesn't appear within the text itself.

```
Use \\code{\\v+\\code{...}+} for inline \\code{code}.
               ^^^^^^^^^^^ not parsed
```

Document properties

To insert document (meta) properties, such as the document author or title, you can use the `\\insert(...)` tag.

In addition to predefined meta properties, you can also define your own properties on the `define` object in the **configuration file**. This can be useful for inserting dynamic text fields such as a customer names or project dates.

```
Prepared for: \\insert(customerName)
By: \\insert(author)
```

Blocks

Text in the document is grouped into ‘block-level’ elements, or blocks. Some blocks can also be nested.

The following types of blocks are supported:

- Paragraphs
- Headings
- Lists
- Separators (horizontal lines, and page breaks)
- Code blocks
- Other blocks (quote/note style)

Paragraphs

The most common block is the paragraph. This is simply a continuous line of text that is automatically wrapped across multiple lines in the resulting document if needed.

Paragraphs are separated by empty lines, *or* other types of blocks (e.g. separators or lists).

In the source document, a single paragraph can be split across multiple lines. A single line break in the source document has no effect on the output. Insert manual line breaks within the text if needed (see **Automatic replacements**].

Headings

Markdown-style headings using hash characters `#` are supported. One hash character at the start of a line denotes a level-1 heading, two characters denotes a level-2 heading, and so on. Only 4 levels are supported.

```
# Level 1 heading
## Level 2 heading
Body text (optionally separated using a blank line).
```

Automatic numbering and references are handled specially within heading blocks. Refer to References & TOC for details.

Lists

Single or multi-level lists can be formatted using Markdown syntax. Use either dashes `-` or asterisks `*` followed by one space, or numbers followed by a dot and a space to demarcate list items.

Empty lines between list items are allowed, and number ranges do not need to be contiguous.

```
- This is an example.
- Nest lists by indenting each nested item:
  1. One
  2. Two
  1. Three
```

- This is an example.
- Nest lists by indenting each nested item:
  1. One
  2. Two
  3. Three

To add other blocks within lists, make sure that the block is indented at least as deeply as the list text itself.

- This is a list item

  ```
  This is a code block within a list item.
  ```

- Another list item.

  A second paragraph.

Separators

Horizontal line — Insert a horizontal line using three or more dashes on a single line (i.e. `---`). The inserted line is surrounded by a significant amount of padding. Override the predefined `separator` style's `lineWidth`, `lineColor`, and `margin` properties to change the separator's appearance.

Page break — Insert a manual page break at any point using the `\\pagebreak` tag. This tag must appear on a separate line as well.

Pre-formatted blocks (code)

To format code in blocks of monospaced text, wrap the text between lines consisting of three or more backticks.

~~~
```
This text will be formatted using a monospaced font,
  indentation and newlines are preserved.
```
~~~

Note: Code blocks of five lines or less are automatically made ‘unbreakable’, i.e. they will not be split across multiple pages in the final document.

Other blocks

Similar to code blocks, other text blocks can be made to stand out from the rest using indentation, background color, or borders. Use the Markdown ‘quote’ syntax to format text as a separate block:

```
This text is not in a block.

> This text is in a block.
> This is on the same line.
```

This text is not in a block.

> This text is in a block. This is on the same line.

By default, the text is only indented slightly. To make the block stand out more, use block styles and properties (see below).

Block styling

All types of blocks (paragraphs, headings, lists, separators, code, other blocks, and also Tables) can be formatted using additional style properties, which are passed directly to the `pdfmake` tool.

You can either pass these properties individually, or use one or more of the named styles that are defined in the configuration file.

Block property tags

To specify individual properties for a block, use one or more ‘property tags’ on a separate line before the block. These tags are formatted using JSON:

```
\\{"alignment": "right", "color": "red"}
Red and right aligned paragraph.
```

Red and right aligned paragraph.

One or more blank lines may appear between a property tag and the block itself, which will not introduce additional space in the output document.

Block property tags can be formatted to span multiple lines if that makes the JSON structure easier to read — especially for ‘quote’ style blocks, which can have multiple properties for border widths and colors as illustrated below.

```
\\{
\\  "margin": [15, 10, 0, 10],
\\  "fillColor": "#ffeeee",
\\  "color": "red",
\\  "borderWidth": [5, 0, 1, 1],
\\  "borderColor": ["red", "#ccc", "#ccc", "#ccc"]
\\}
> This block looks very different.
```

> This block looks very different.

Block style tags

Instead of specifying individual properties for each block, you can also use named block styles. Named styles are predefined groups of properties.

A number of built-in styles are available as well, such as `bold`, `italics`, `left`, `right`, `center`, `small`, `nomargin`, `caption`, `h1-h4`, `unbreakable`, and `note`.

Styles can be added or overridden using the Configuration file.

Specify one or more styles using a single tag using curly braces, e.g.:

```
\\{note}
> **Note**: This is a note block. It stands out from the rest of
the text.

\\{right small unbreakable}
This style works well for smallprint, aligned to the right side
of the page and set using a smaller font and slightly faded
color. The text will not be broken up across multiple pages if it
occurs near the bottom of a page.
```

> **Note**: This is a note block. It stands out from the rest of the text.

This style works well for small print, aligned to the right side of the page and set using a smaller font and slightly faded color. The text will not be broken up across multiple pages if it occurs near the bottom of a page.

Tables & Columns

Two methods for generating tables are supported by makepdf. The first is based on the syntax used by most Markdown parsers, while the second is based on list syntax, which is better suited for tables with large amounts of text. This method can also be used to create ‘columns’, which are a lot like tables with a single row and different padding styles.

Markdown syntax

Tables are created using pipe characters `|` around cells (the first and last ones are optional), and a ‘separator’ line between the table heading and other rows. The separator line consists of three or more dashes for each cell, separated by pipe characters.

Consider the following Markdown source text:

```
| One | Two | Three |
| --- | --- | ----- |
| X   | Y   | Z     |
| 1   | 2   | 3     |
```

This becomes a table with a heading row and two rows, and three columns.

One	Two	Three
X	Y	Z
1	2	3

Literal pipe characters — To include literal pipe characters within your table, prefix them with a backslash character, i.e. `\|`.

Column alignment — To align all cells in a table column to the left (default), right, or center, add colon characters on the left side, right side, or both sides of the separator dashes respectively.

```
| Right | Center | Left |
| ----: | :----: | :--- |
| X     | Y      | Z    |
| 1     | 2      | 3    |
```

Right	Center	Left
X	Y	Z
1	2	3

Column width — Normally, each column takes up as little or as much space as it needs to fit its contents. The table doesn't necessarily take up the full width of the page. Markdown doesn't provide a way to set column widths, but the **block properties tag** (or block style tag) can be used to specify widths explicitly as an array. The width of each column should be specified in the following manner:

- A number, which specifies the width in points (1/72").
- `auto` — makes the column take up as much space as needed, within as little space as available.
- `*` — which makes the column take up as much space as available.

Example:

```
\\{"widths": [50,"*"]}
|Amount|Description|
|---:|---|
|$128|Lorem ipsum dolor sit amet|
|$1,000|Adipiscit|
```

Amount	Description
$128	Lorem ipsum dolor sit amet
$1,000	Adipiscit

No heading row — To insert a table without a heading row, simply start the table with a separator line:

```
| --- | --- | ----- |
| X   | Y   | Z     |
| 1   | 2   | 3     |
```

X	Y	Z
1	2	3

Decorations — The `layout` block property can be used to specify a specific combination of line widths, line colors, and padding.

The following predefined (named) table layouts are available:

- `default` — as illustrated above, horizontal lines only.
- `noBorders` — no borders at all, no padding.
- `allBorders` — all borders (thin), regular padding.

(Note that the header row background and font are defined using the `tableHeader` style, and are not affected by the table 'layout').

```
\\{"layout": "noBorders"}
| One | Two | Three |
| --- | --- | ----- |
| X   | Y   | Z     |
| 1   | 2   | 3     |
```

One	Two	Three
X	Y	Z
1	2	3

Note: You can also use a **block style tag** to set layout, widths, and other properties in one go, if you add a custom style definition to the configuration file.

By default, the `defs` (definitions table) style can be used to create tables with a narrow first column and a full-width second column — which is useful for tables that describe a set of terms:

Element	Description
Input field	Enter a number between 1-100.
OK button	Click here to approve the transaction.
Cancel button	Click here to close the dialog and cancel the transaction.

List syntax

Alternatively, you can turn a nested list into a table using a block property tag. Set the `table` property to `true`, or to a specific table layout name to turn list items into rows and nested list items into cells.

By default, the first list item becomes the table's heading row. If you do not want the table to include a heading row, set the `headerRows` property to zero.

This is especially useful if your table contains a lot of text, or nested (styled) blocks, as in the example below.

```
\\{"table": "noBorders", "widths": ["*", 50], "headerRows": 0}
- - Lorem ipsum dolor sit amet, consectetur adipiscing elit.
    Suspendisse elementum tellus dui, ac pretium nisl sodales in.

    Aliquam non est ut libero dapibus sagittis vitae quis dolor.
  - 00:05
- - Ut id est et enim laoreet condimentum in blandit tortor.
    Suspendisse nec nulla at elit lobortis blandit id in justo.
    Sed tempus dui ut mauris euismod dapibus.
    Morbi quis convallis mauris.
  - 01:10
- - Vestibulum ante ipsum primis in faucibus orci luctus et
    ultrices posuere cubilia Curae;
    Pellentesque habitant morbi tristique senectus et netus et
    malesuada fames ac turpis egestas.

    Habitant morbi tristique senectus et netus et malesuada
    fames ac turpis egestas.
  - 02:25
```

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse elementum tellus dui, ac pretium nisl sodales in. Aliquam non est ut libero dapibus sagittis vitae quis dolor.	00:05

Ut id est et enim laoreet condimentum in blandit tortor. Suspendisse nec nulla at elit lobortis blandit id in justo. Sed tempus dui ut mauris euismod dapibus. Morbi quis convallis mauris.	01:10
Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.	02:25
Habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.	

Note that nested list items can be formatted as above with two bullets on a single line, or with items on a separate line (see below). Any content before the nested list is ignored — however, list items cannot be completely blank (after the `-` or `*` symbol), so it's a good idea to add a single character to indicate new rows.

Named styles that include a `table` property have the same effect, such as the built-in `defs` style.

```
\\{defs}
- |
  - Element
  - Description
- |
  - Input field
  - Enter a number between 1-100.
- |
  - **OK** button
  - Click here to approve the transaction.
- |
  - **Cancel** button
  - Click here to close the dialog and cancel the transaction.
```

Element	Description
Input field	Enter a number between 1-100.
OK button	Click here to approve the transaction.
Cancel button	Click here to close the dialog and cancel the transaction.

Columns

A list with two or more items can also be converted into blocks that are placed next to each other on the horizontal axis, using the `columns` property.

Set this property to `true` to enable column mode. By default, all columns are equally sized.

```
\\{"columns": true}
- Left
- Middle
- Right
```

Left Middle Right

To change column padding and sizing, add optional `columnGap` and/or `widths` properties. If `columnGap` is specified, the `column` property is no longer needed.

```
\\{"columnGap": 6, "widths": ["auto", "*"]}
- **Note**
- Nullam vitae urna metus. Vivamus quis justo eros. Nullam
auctor purus at tincidunt bibendum. Praesent non blandit arcu,
eget lacinia metus.
```

Note Nullam vitae urna metus. Vivamus quis justo eros. Nullam auctor purus at tincidunt bibendum. Praesent non blandit arcu, eget lacinia metus.

Tables and columns can be nested inside of other block-level elements, and the other way around, as illustrated by the following example.

```
\\{note}
> \\{"columnGap": 6, "widths": ["auto"], "margin": [0, 0]}
> - **Note**
> - Nullam vitae urna metus. Vivamus quis justo eros.
>   Nullam auctor purus at tincidunt bibendum.
>   \\{small right nomargin}
>   Aliquam non est ut libero dapibus sagittis vitae quis dolor.
```

Note Nullam vitae urna metus. Vivamus quis justo eros. Nullam auctor purus at tincidunt bibendum.

Aliquam non est ut libero dapibus sagittis vitae quis dolor.

References & TOC

Automatic numbering

Automatic numbering is supported using tags that insert a number *and* automatically increment an internal counter. Different counters are supported, to be able to keep track of various (nested) sequences.

Section numbering

Regular numbers (1, 2, 3, etc.) can be inserted using the `\\().` tag, which is reserved for chapter numbering. Each section can then be numbered using `\\().().` (the final dot is mandatory, but will not be inserted — depending on the value of `autonumSuffix` in the **Configuration file**).

Alphabetic sequences (e.g. appendix numbering) can be inserted using `\\[].`. These can be mixed an matched, such that a section of an appendix chapter can be numbered using `\\[].().`.

Example:

```
# \\(). Introduction
## \\().(). Definitions
Lorem ipsum...

# Appendix \\[]. \\newline Examples
**Example \\[].().**
```

Automatically generated numbers within headings are styled using the special `autonum_h1` through `autonum_h4` styles. By default, numbers within level 1 headings are made bold and much larger than the rest of the text. This can be overridden by changing the properties for these styles in the **Configuration file**.

Other sequences

Any (nested) numbering sequence can be explicitly named, by placing a name within the tag's brackets, e.g. `\\(fig).`. Nested numbering sequences are also supported, for numbering within sections (i.e. `\\().(ex).`) or sequences within named sequences (i.e. `\\(clause).[].`).

Examples:

- First `\\(ex).` is 1
- Second `\\(ex).` is 2
  - Nested `\\(ex).[].` is 2.A
  - Nested `\\(ex).[].` is 2.B with further sequences of `\\(ex).[].().` as 2.B.1 , 2.B.2 , 2.B.3
  - A nested named sequence starts at 1 again, e.g. `\\(ex).[].(sub).`: 2.B.1 , 2.B.2 , 2.B.3
- Alpha `\\[ex].` is independent of numeric: A
- Second `\\[ex].` is B

Cross-references

To be able to add cross-references within the document, first the relevant target sections need to be marked.

Heading markers

Headings (both with and without numbers) can be marked by adding a tag at the end of the heading line, as follows.

```
## \\().(). Sample heading {#foo}
```

This heading can then be referenced using a Markdown-style link. Most PDF readers will make the link 'clickable', and also when printed the link can be made to look different (since links are styles using the `doclink` style; different from the `link` style which is applied to 'external' links). Example:

```
Refer to [Sample](#foo) for details.
```

An advantage of marking the entire heading is that the heading text can be used dynamically as the text of the reference. Leave out all text from the link to insert the heading title automatically.

```
Refer to section [](#foo) for details.
```

If the section number is e.g. 2.1, the above text will read "Refer to section 2.1 Sample heading for details".

Auto-number markers

To insert cross-references to *other* parts of the document (i.e. targets that are not headings), you'll need to use automatic numbering. This ensures that references still make sense even if the document is printed.

Markers can be added to automatically numbered text, by adding the tag directly after the numbering tag. Example:

```
\\{caption}
**Table \\(tbl).{#resultsTable}** -- Final results
```

Table 1 — Final results

The marker can then be referenced in the same way as headings, however in this case leaving out the heading text only inserts the number itself, not all of the surrounding text.

```
Refer to table [](#resultsTable) for the final results.
```

Refer to table 1 for the final results.

Table of contents

A Table of Contents (TOC) can be inserted automatically, as a list of cross-references with heading titles and page numbers.

To insert a TOC, simply use the `\\toc` tag, on a separate line.

Heading levels — The maximum level for headings to appear in the TOC can be configured in the Configuration file.

TOC table style — The TOC itself is styled using the `toc` style, which can be overridden. For example, you could change the overall font size, outer margin, or column widths.

TOC entry styles — Each heading level is styled independently, which is what makes it possible to show the TOC as a hierarchy instead of a single list. Override the `toc1` through `toc4` styles to change the appearance of headings for each level within the TOC (e.g. font, style, or margin).

Transclusion

Transclusion refers to the inclusion of content from another file that is placed alongside the current file. In documents that are processed by makepdf, images and text can be transcluded in almost the same way.

Images

While it's currently not possible to include images within paragraph text (i.e. 'inline'), it is possible to include images as a separate block-level element.

The `\\image(...)` tag can be used to refer to an image file (PNG or JPG), and transclude the image itself into the document.

```
Normal text goes here.

\\image(plant.png)
```

Positioning — Use tables or columns to position images alongside parts of your document text, or use margins and/or alignment to offset the image on the page.

Properties — By default, the image is displayed using its 'natural' dimensions, or otherwise as large as possible. This can be changed by setting either the `width` property, `height` property, or both.

Captions — Any text that occurs after the `image` tag itself is considered to be part of the image *caption*, displayed underneath the image.

```
Normal text goes here.

\\image(plant.png) Figure \\(fig). -- A common house plant.
```

Folders — To include images from other folders, simply prepend the folder name (e.g. `other/plant.png`), or use `../` to go up a folder (e.g. `../img/plant.png`). As a shortcut, you can prepend the file/folder name with a slash `/` to find files from the folder that contains the configuration file (or base folder) instead of relative to the current file.

Multiple files — You can include multiple files in one go using 'glob' patterns.

For example, `../img/plants/**/*.png` will insert all 'PNG' images from any folder within the 'plants' folder at the given path. Files will be sorted by name. The caption, if any, will be used for all files (i.e. it is repeated).

Text files

Text from other files can be inserted using another type of transclusion tag. To include the full text of a file `other.md`, use the following tag:

```
\\include(other.md)
```

You can also include files from other folders, and multiple files at the same time using the same patterns described above for image files.

This makes it possible to split a long document into multiple files, e.g. one per chapter, and then transclude this content into the output document.

Dynamic content — Instead of the caption that's used for included images (see above), any text placed after the `\\include` tag, OR below the tag indented by at least one more space, will be passed to the file as its 'content'. Content can be inserted at any point in the file using the following tag:

```
Inserted content: \\insert(content)
```

This is mostly useful for files that are *templates*, which encapsulate a piece of content with e.g. block decorations or standard text.

For example, with the following text in a file `/includes/yellownote.md`:

```
\\{note}
\\{"fillColor": "#ffffee"}
> **Note:**
> \\insert(content)
```

The following text in another file will result in the note displayed below.

```
\\include(/include/yellownote.md)
    This is a paragraph that is displayed within a note
    that has a light yellow background tint.
```

Note: This is a paragraph that is displayed within a note that has a light yellow background tint.

The content text is inserted as-is (including line breaks), *unless* the `\\insert` tag exists on its own line — in which case all of the indentation and block marker (i.e. `>`) characters before the `\\insert` tag are duplicated for all inserted lines.

JavaScript code

As the ultimate solution for inserting any type of content into the PDF output, you can use the `\\include` tag to include the *exported value* of a JavaScript (CommonJS) module.

The exported data must be in the form of plain objects, strings, and arrays, as understood by the `pdfmake` library.