Docco: Feature Showcase

Welcome to this demonstration of Docco's capabilities. This document showcases all features through practical examples and explanations.

1 Getting Started

1.1 Frontmatter Configuration

YAML frontmatter at the beginning of the document (between `---` delimiters) configures document processing:

```
---
css:
  - "css/page.css"
  - "css/toc.css"
multilingual: true
base_language: en
---
```

1.1.1 Supported Fields

`css` - CSS stylesheet(s) for PDF styling. Can be:

- Single file (string): `css: "style.css"`
- Multiple files (inline array): `css: ["page.css", "theme.css"]`
- Multiple files (multiline list):

```
css:
  - "css/page.css"
  - "css/toc.css"
```

- External CSS URLs (e.g., Google Fonts):

```
css:
  - "css/page.css"
  - "https://fonts.googleapis.com/css?family=Raleway:400,600&display=s
```

File paths are relative to the markdown file and are embedded in the generated HTML document within `<style>` tags. External CSS URLs (starting with `http://` or `https://`) are included as `<link>` tags in the HTML, allowing WeasyPrint to fetch them during PDF generation. This enables use of web fonts like Google Fonts directly in your PDF documents.

`multilingual` - Enable multilingual mode (boolean, default: `false`). When set to `true`, Docco automatically extracts translatable strings to a POT file and generates PDFs for the base language plus all discovered translations.

`base_language` - The language code of the source document (required when `multilingual: true`). Example: `base_language: en`. This will be used as the suffix for the base language PDF (e.g., `Document_EN.pdf`).

`dpi` - Maximum image resolution in dots per inch (integer, optional). Controls image downsampling in the generated PDF:

- Default (no `dpi` specified): Preserves original image resolution. High-quality images remain full resolution, which is excellent for print but may result in larger file sizes.
- `dpi: 300`: Recommended for professional printing. Images are downsampled to a maximum of 300 DPI, which is the industry standard for high-quality print output while keeping file sizes reasonable.
- `dpi: 150`: Suitable for screen viewing and digital distribution. Produces smaller files with adequate quality for on-screen reading.

Example:

```
---
css: "style.css"
dpi: 300
---
```

Note: The DPI setting works most effectively when images have CSS constraints. For optimal file size reduction, include CSS rules like:

```
img {
    max-width: 100%;
    height: auto;
}
```

This setting applies to all raster images (PNG, JPEG, etc.) embedded in the document. It does not affect vector graphics (SVG). If images are already at or below the specified DPI, they remain unchanged.

2 Core Concepts

2.1 Understanding Directives

Docco extends markdown with powerful **directives** - special HTML comments that trigger processing. Directives enable:

- File inclusion and composition
- Dynamic content generation
- Page layout control
- Placeholder substitution
- Python code execution

2.1.1 General Directive Rule

Directives can appear anywhere in the document, including in the middle of lines. However, directives inside code blocks (both inline `` `code` `` and fenced blocks) are **protected** and will not be processed. This allows you to show directive syntax as examples in documentation without triggering them.

Example (this won't execute):

```
<!-- inline:"file.md" -->
<!-- python -->print("hello")<!-- /python -->
```

This protection ensures compatibility with standard markdown parsing and allows you to safely demonstrate directive syntax in tutorials and documentation.

3 Docco Directives

3.1 Inline Content

The `inline` directive embeds external markdown or HTML files with optional placeholder substitution.

Syntax: `<!-- inline:"path/to/file" key1="value1" key2="value2" -->`

3.1.1 Basic Usage

All attributes after the file path become placeholders. For example, `author="Docco Team"` replaces all `{{author}}` occurrences in the inlined file:

This content is inlined, with arguments.

Author: Docco Team Date: 2025-10-26

3.1.2 Recursive Inlining

Inlined files can themselves contain inline directives, enabling multi-level composition (up to 10 levels maximum to prevent infinite recursion). This allows modular document structures where content is composed from nested files.

3.1.3 Important: HTML Content and Indentation

When inlining HTML files, be aware that **all content is parsed as markdown**:

- **Leading indentation matters**: Lines starting with 4+ spaces are treated as code blocks. If your inlined HTML has indentation, it will be wrapped in `<code>` tags, breaking the structure.
- **Solution**: HTML files must start at column 0 (no leading indentation). Inline directives themselves should also not be indented.

Correct:

```
<!-- inline:"header.html" -->
```

Incorrect:

```
    <!-- inline:"header.html" -->
```

This follows the CommonMark specification for compatibility with standard markdown parsing.

3.2 Table of Contents

The `<!-- TOC -->` directive generates a hierarchical, automatically numbered table of contents (1, 1.1, 1.2, 1.2.1, etc.).

To exclude a heading from the TOC and remove its numbering, use `<!-- toc:exclude -->` before the heading:

```
<!-- toc:exclude -->
## Appendix (not numbered, not in TOC)
```

3.3 Page Layout

Control page breaks and orientation within your document using layout directives.

3.3.1 Page Breaks

The `<!-- pagebreak -->` directive starts a new page:

This section starts on a **new page** using the `<!-- pagebreak -->` directive. Use page breaks to organize content into logical sections with clear visual separation.

3.3.2 Orientation Control

The `<!-- landscape -->` and `<!-- portrait -->` directives control page orientation, useful for wide content like tables:

This section uses **landscape orientation** with the `<!-- landscape -->` directive, providing more horizontal space:

Q1 Revenue	Q1 Expenses	Q1 Profit	Q2 Revenue	Q2 Expenses	Q2 Profit	Q3 Revenue	Q3 Expenses	Q3 Profit	Q4 Revenue	Q4 Expenses
$50,000	$35,000	$15,000	$55,000	$37,000	$18,000	$62,000	$40,000	$22,000	$71,000	$45,000

This section returns to **portrait orientation** using the `<!-- portrait -->` directive.

3.4 Headers & Footers

Docco supports page headers and footers added via **inline directives**. Headers and footers are processed through the same pipeline as the main document, supporting:

- Placeholder substitution
- Dynamic content with `<!-- python -->` directives
- File inclusion with `<!-- inline -->` directives

3.4.1 Adding Headers & Footers

Include headers and footers at the beginning of your document using inline directives with placeholder attributes:

```
<!-- inline:"header.html" title="Docco Feature Showcase" author="Docco Tea
<!-- inline:"footer.html" title="Docco" -->
```

The attributes replace `{{key}}` placeholders in the HTML files. For example, `title="My Title"` replaces all `{{title}}` instances.

3.4.2 CSS Requirements

Headers and footers require CSS Paged Media rules to position them on the page:

```
@page {
    margin-top: 2.5cm;
    margin-bottom: 2.5cm;
    @top-center {
        content: element(header);
    }
    @bottom-center {
        content: element(footer);
    }
}
```

See `examples/css/header_footer.css` for a complete example.

3.4.3 Example Files

Docco provides example header and footer HTML files in the `examples/` folder:

- `examples/header.html` - Example page header with placeholder support
- `examples/footer.html` - Example page footer with directive support

Examine these files to understand the structure and create your own headers and footers with placeholders (e.g., `{{title}}`, `{{author}}`) and directives.

3.5 Python Code Execution

The `<!-- python -->` directive executes Python code and inserts stdout output into the markdown. Useful for generating dynamic content.

Syntax: `<!-- python -->code<!-- /python -->`

Important: Python code execution is disabled by default for security reasons. Use the `--allow-python` flag to enable it:

```
docco input.md -o output/ --allow-python
```

3.5.1 Example

This code:

```
print("_", end='')
for i in range(10):
    print(i, end='')
print("_", end='')
```

Produces: 0123456789

The output can contain other directives (markdown, inline files, etc.), enabling complex dynamic content generation.

4 Document Formatting

Docco relies on MarkdownIt for rendering markdown to HTML. It fully supports CommonMark specs with table support. The (block) attributes plugin is also installed.

4.1 Images with Styling

Add images using standard Markdown syntax and use `{}` attributes for styling:

```
![](images/idea.svg){.icon}
```

This defines an image with CSS class `icon` (styled in `css/fancy.css`):

Or define styles directly: `![](images/idea.svg){style="width:2cm"}`

4.2 Tables

Markdown tables organize tabular data with optional styling:

A	B	C
Table	with borders	inside

A	B	C
Table	with borders	outside

A	B	C
Table	without	borders

4.3 More Markdown Features

Explore additional markdown features: https://markdown-it.github.io/

5 Styling & Layout

5.1 CSS for PDF Generation

WeasyPrint (version 66.0 at time of writing) is used to convert HTML to PDF. It supports CSS up to v2.1, with partial support for modern CSS features:

Supported:

- Basic layout: block, inline, float, positioning
- Some modern features: Flexbox, Grid (partially)
- Media queries and custom selectors
- CSS custom properties (`--variables`)

Not fully supported:

- Some CSS 3+ features (check WeasyPrint documentation)

For detailed CSS support information, see the WeasyPrint API reference.

5.2 Best Practices

- Keep CSS focused on print-friendly layouts
- Test CSS features before heavy use
- Reference the WeasyPrint documentation for edge cases
- Use CSS Paged Media rules for headers, footers, and page styling

6 Multilingual Documents

Create professional multilingual documents with automatic language-specific PDF generation.

6.1 Automatic Multilingual Mode

Use the `multilingual: true` flag in frontmatter with `base_language` to automatically generate PDFs for the base language plus all available translations:

```
---
multilingual: true
base_language: en
---
```

6.1.1 How It Works

When enabled, Docco will:

1. Extract a POT file to a `{document_name}/` subfolder
2. Discover all `.po` translation files in that subfolder
3. Generate a PDF for the base language (e.g., `Document_EN.pdf`)
4. Generate translated PDFs for each `.po` file (e.g., `Document_DE.pdf`, `Document_FR.pdf`)

Example: See `Multilingual_Document_Example.md` which generates:

- `Multilingual_Document_Example_EN.pdf`
- `Multilingual_Document_Example_DE.pdf`
- `Multilingual_Document_Example_NL.pdf`

6.2 Manual Translation Workflow

For single-language builds with specific translations, use the `--po` flag:

```
docco myfile.md --po translations/de.po -o output/
```

This generates a single PDF with the specified translation applied.

6.3 Professional Translation Workflow with POT/PO Files

Docco integrates with professional translation tools and services for enterprise workflows.

6.3.1 Step 1: Enable Multilingual Mode

Add to your frontmatter:

```
---
multilingual: true
base_language: en
---
```

6.3.2 Step 2: Generate Initial PDFs and POT File

```
docco myfile.md -o output/
```

This automatically:

- Generates `myfile_EN.pdf` (base language)
- Creates `myfile/myfile.pot` (translation template)

File structure:

```
myfile.md
myfile/
  └── myfile.pot        (template)
output/
  └── myfile_EN.pdf
```

6.3.3 Step 3: Create Language-Specific Translations

Translators create `.po` files for each language using professional tools:

- **poedit** (desktop application)
- **Weblate** (web-based, collaborative)
- **Crowdin, Lokalise, POEditor** (professional translation platforms)
- Any gettext-compatible tool

Place them in the `myfile/` directory:

```
myfile.md
myfile/
  ├── myfile.pot        (template)
  ├── de.po             (German translation)
  ├── fr.po             (French translation)
  └── nl.po             (Dutch translation)
```

6.3.4 Step 4: Generate Multilingual PDFs

```
docco myfile.md -o output/
```

This automatically:

- Updates POT file
- Updates all existing PO files with new/changed strings
- Generates PDFs for all languages

```
Extracted POT: myfile/myfile.pot
Updated de.po: 40 translated, 2 fuzzy, 5 untranslated
Updated fr.po: 38 translated, 1 fuzzy, 6 untranslated
Updated nl.po: 45 translated, 0 fuzzy, 0 untranslated
```

```
Processing base language: EN
Processing language: DE
WARNING - Translation incomplete for DE: 5 untranslated, 2 fuzzy
Processing language: FR
WARNING - Translation incomplete for FR: 6 untranslated, 1 fuzzy
Processing language: NL
Generated 4 output file(s)
```

6.3.5 Translation Maintenance

When the source document changes, simply run the same command:

```
docco myfile.md -o output/
```

Docco automatically:

- Updates POT file with new content
- Merges changes into all existing PO files
- Preserves existing translations
- Marks new strings as untranslated
- Marks changed strings for review (fuzzy)
- Regenerates all PDFs

This allows translators to focus only on new/modified content rather than re-translating the entire document.

7 Conclusion

This document demonstrates all of Docco's core capabilities: configuration, directives, formatting, styling, and multilingual support. Use these features to create professional, multilingual documents with dynamic content and flexible layouts.

For more information, consult the main Docco documentation.