RexxPub: A Rexx Publishing Framework

A Work in Progress

Josep Maria Blasco
Espacio Psicoanalítico de Barcelona
Balmes, 32, 2º 1ª — 08007 Barcelona
jose.maria.blasco@gmail.com
+34 93 454 89 78

May the 6th, 2026

This document[1] was written using RexxPub, a Rexx Publishing Framework.[2]

1. Introduction

The Rexx Parser was developed to support deep, semantically accurate analysis of Rexx source code, and its child project, the Rexx Highlighter, followed naturally as its first visible application. Both were presented at last year's Symposium [1], [2], and a revised account, with the features added since, is given in a companion article [3]. With the Highlighter in place, something new became possible: writing serious technical documents in which Rexx code could appear as a first-class citizen, including the work currently under way on the official ooRexx DocBook documentation, covered in that same companion article. But producing complete, well-formatted documents — web pages, articles, slides, or PDF files — needs more than a highlighter: an authoring format, a document transformation pipeline, and some way to deliver the result. RexxPub is a framework that provides all of this.

The authoring format is Markdown. Markdown is easy to write, easy to read, easy to learn, and easy to maintain. It is far simpler than HTML or LaTeX, yet powerful enough for most publishing needs, especially when combined with Pandoc, a universal document converter that extends Markdown with footnotes, tables, citations, and many other features.

RexxPub provides four pipelines that share a common two-stage transformation core; what varies is what happens before and after it:

- The **Web** pipeline serves the HTML dynamically through a CGI program running under Apache httpd. The pages under `https://rexx.epbcn.com/rexx-parser/` are a live example of this pipeline. It also supports a print variant: by appending `?print=pdf` to the URL, the same page is rendered as a paginated, print-ready article, letter, or slide deck directly in the browser, using Paged.js, a JavaScript polyfill that implements most of the CSS Paged Media specification. The document you are reading is itself a live example of this variant: it was generated from `https://rexx.epbcn.com/rexx-parser/doc/publications/37/2026-05-06-RexxPub-A-Rexx-Publishing-Framework/?print=pdf` and printed from the browser.
- The **HTML** pipeline uses the md2html utility to convert Markdown files — either a single file or an entire directory tree — into static HTML, suitable for publication on any web server or hosting service.
- The **PDF** pipeline uses the md2pdf utility to produce print-quality PDF files directly from the command line, from a single file or an entire directory tree, without requiring a web server or opening a browser. It relies on pagedjs-cli, a Node.js command-line tool that runs paged.js outside the browser using a headless Chromium, and, optionally, on pikepdf to activate the document outline in the resulting PDF.
- The **EPUB** pipeline uses the md2epub utility to produce EPUB ebooks directly from Markdown, from a single file or an entire directory tree. Pandoc generates the EPUB directly, without requiring pagedjs-cli, a headless browser, or any tool beyond Pandoc itself.

The next section describes the common core shared by all the pipelines. The following four sections describe each of these pipelines in turn. A further section covers document classes and the pagination infrastructure shared by the Web (in print mode) and PDF pipelines. The final section discusses the design decisions that shape the framework as a whole.

---

1. This document was presented to the 37th International Rexx Language Symposium, held at the Espacio Psicoanalítico de Barcelona and online from the 3rd to the 6th of May, 2026. URL of this document: `https://www.epbcn.com/pdf/josep-maria-blasco/2026-05-06-RexxPub-A-Rexx-Publishing-Framework.pdf`; HTML version: `https://rexx.epbcn.com/rexx-parser/doc/publications/37/2026-05-06-RexxPub-A-Rexx-Publishing-Framework/`.↩
2. See `https://rexx.epbcn.com/rexx-parser/doc/rexxpub/`.↩

2. The Common Core

Every RexxPub pipeline begins with the same preprocessing: when Rexx fenced code blocks are present, `FencedCode.cls` replaces them with their highlighted HTML; otherwise the Markdown source is passed through unchanged. The result is

then handed to Pandoc, which converts it to the format each pipeline requires — HTML for Web, HTML, and PDF, EPUB directly for EPUB.

2.1. FencedCode.cls

`FencedCode.cls` is a Rexx file containing a public routine, `FencedCode`, that acts as a Markdown preprocessor. It receives an array of lines (typically the contents of a Markdown file), scans it for Rexx fenced code blocks, and returns a new array where every such block has been replaced by its highlighted HTML version. Everything else —paragraphs, headings, non-Rexx code blocks— is passed through untouched.

A Rexx fenced code block is a standard Markdown fenced code block whose language tag is `rexx`:

```
~~~rexx
Say "Hello, world!"
~~~
```

Both backticks and twiddles are accepted, with three or more characters, but the opening and closing markers must match in style and length. Code blocks with a different language tag (or no tag at all) are silently ignored.

Attributes

Fenced code blocks accept optional attributes enclosed in braces after the `rexx` tag:

```
~~~rexx {.numberLines style=dark pad=80 executor}
```

Attributes control how the Highlighter processes the block. They include:

- `style=`*name* — selects a predefined highlighting style (default: `dark`).
- `source=`*filename* — reads the code from an external file instead of the block contents, which allows a Markdown document to include a program listing that always reflects the current version of the source.
- `patch=`*"patches"* and `patchfile=`*filename* — apply style patches to customize the appearance of specific element categories.
- `caption=`*"text"* — adds a caption to the code block. Captions are rendered as `<figcaption>` elements, with "Listing *n*:" numbering following the LaTeX convention. Listing captions are placed above the code block by default.
- `size=`*size* — overrides the font size for an individual code block (e.g., `size=10pt` or `size=0.8em`).
- `.numberLines`, `startFrom=`*n*, `numberWidth=`*w* — enable and configure line numbering.
- `pad=`*column* — pads `::Resource` data lines and doc-comments to a fixed

width, useful for producing clean rectangular backgrounds.

- `doccomments=`*mode* — controls doc-comment highlighting: `"detailed"` (the default) highlights each part individually; `"block"` highlights the entire doc-comment as a single block.
- `executor` — enables Executor extensions.
- `experimental` — enables Experimental features.
- `tutor` (or `unicode`) — enables TUTOR-flavoured Unicode support.
- `operator`, `special`, `constant`, `assignment` — control the granularity of HTML class assignments (`"group"`, `"detail"`, or `"full"`), allowing fine-grained or coarse-grained control over how different element categories are styled.

Under the hood, `FencedCode.cls` instantiates the Rexx Highlighter for each code block, passing it the source lines and the parsed options. The Highlighter invokes the Rexx Parser to produce a full AST, walks the tree, and assigns one or more CSS classes to every parsed element based on its syntactic role — keywords, variables, operators, strings, comments, directives, and dozens of other categories are all distinguished. The result is a `<pre>` block containing richly annotated `<span>` elements, wrapped in a `<div>` whose class encodes the selected style.

Predefined styles

The Highlighter ships with a palette of predefined styles: `dark` and `light` (the author's original styles), `rgfdark` and `rgflight` (Rony Flatscher's adaptations), `print` (optimised for paper output with high-contrast colours that remain distinguishable in greyscale), `tokio-night` and `tokio-day` (dark and light styles inspired by the Tokio colour palette), `electric` (a vivid dark style), and a collection of styles derived from Vim colour schemes (`vim-dark-blue`, `vim-dark-desert`, `vim-light-morning`, and many others), also contributed by Rony Flatscher. The style is selected per code block via the `style` attribute. A global default for the whole document can also be set; the mechanism for doing so depends on the pipeline.

2.2. Pandoc

After `FencedCode.cls` has processed all the Rexx fenced code blocks, the result is still a Markdown document, but with HTML fragments in place of the Rexx code blocks. Pandoc then converts it into a complete HTML document.

Pandoc provides many features that plain Markdown lacks: footnotes (essential for conference articles), smart typography (curly quotes, em-dashes), tables, definition lists, and citation support.

3. The Web pipeline: Dynamic Publishing

The Web pipeline, the first and most feature-rich, delivers highlighted Markdown documents dynamically through the web, using the Common Gateway Interface (CGI) protocol under Apache httpd.

This section walks through the Web pipeline in detail: the Apache directives that activate it, the structure of the CGI script that processes each request, the page layout it produces, the style chooser it offers to readers, the live example that runs at `https://rexx.epbcn.com/rexx-parser/`, and the print variant that turns any page into a paginated, print-ready document.

3.1. Apache Configuration

The Apache configuration required to activate the Web pipeline is minimal. Two directives are sufficient[3]:

```
Action Markdown /cgi-bin/CGI.markdown.rex
```

```
<Files *.md>
  SetHandler Markdown
</Files>
```

The first directive defines an action that associates the name `Markdown` with the CGI script `CGI.markdown.rex`. The second tells Apache to apply this action to all files with a `.md` extension. From this point on, every Markdown file served by Apache will be processed by the CGI pipeline instead of being delivered as plain text.

A third, optional directive further simplifies the site's URL structure. By adding the Markdown filenames to the `DirectoryIndex` list, Apache will serve them automatically when a directory is requested, just as it would serve an `index.html`:

```
DirectoryIndex readme.md article.md letter.md index.html
```

Such an approach allows all URLs on a site to use clean directory paths — for example, `https://rexx.epbcn.com/rexx-parser/doc/highlighter/` instead of `https://rexx.epbcn.com/rexx-parser/doc/highlighter/readme.md`. The order of the filenames in the directive determines the priority: if a directory contains both a `readme.md` and an `article.md`, the `readme.md` will be served. The trailing `index.html` provides a fallback for directories that contain a conventional HTML index instead of a Markdown file. Combined with the two directives above, this means that the entire site can be navigated using directory URLs, with Apache transparently selecting and processing the appropriate Markdown file through the CGI pipeline.

---

3. The directives shown here assume that the package is installed as distributed, for an initial test; for a production site, paths and filenames should be adapted accordingly. See the installation tutorial for details.↩

3.2. The CGI Script

The CGI script, `CGI.markdown.rex`, is an ooRexx program that subclasses `Rexx.CGI`, an abstract class that encapsulates the generic machinery of a CGI program: it creates request and response objects, redirects program output to an in-memory array (so that HTTP headers can be emitted before the body, even though the body is generated first), performs basic security checks, and manages the response lifecycle.

The subclass implements a single method, `Process`, which contains the pipeline-specific logic:

1. **Parameter parsing.** The query string is inspected for recognized parameters: `style=`*name* selects a highlighting style, `size=`*n* selects the base font size, `sections=`*n* overrides the section numbering depth, `numberfigures=0` disables automatic figure/listing numbering, `print=pdf` activates paged.js for print-ready output (see *The Print Variant* below), and `view=highlight` enables source viewing for `.rex` and `.cls` files. Unrecognized parameters produce a 404 response.
2. **File reading.** The file identified by Apache's `PATH_TRANSLATED` environment variable is read into an array.
3. **Rexx source files.** If the requested file is a `.rex` or `.cls` file and `view=highlight` has been specified, the file contents are wrapped in a Rexx fenced code block and passed through `FencedCode.cls`. The result is inserted into a minimal HTML template and served directly. This allows any Rexx source file on the site to be viewed with full syntax highlighting simply by appending `?view=highlight` to its URL.
4. **Markdown files.** For `.md` files, the standard two-stage core is applied: `FencedCode.cls` processes the Rexx fenced code blocks, and then Pandoc converts the enriched Markdown to HTML. Pandoc is invoked from ooRexx using `Address Command` with `With Input Using (source) Output Using (contents)`, which pipes the array directly to Pandoc's standard input and captures the output in another array, without intermediate files.
5. **Template insertion.** Pandoc converts Markdown into an HTML *fragment* — just the body content, without `<html>`, `<head>`, or the surrounding page structure. To become a complete page, that fragment must be wrapped in something that provides the title, navigation, sidebar, footer, and CSS links. Doing this with string concatenation in the CGI would entangle content with presentation; using a template keeps them separate, lets every page share the same layout, and makes it possible to restyle the entire site by editing a single file. The template contains placeholder tags (`%title%`, `%header%`, `%contents%`, `%sidebar%`, `%footer%`, and others) that are replaced at runtime, and is stored as an ooRexx `::Resource` embedded in the CGI script itself.
6. **Dynamic CSS injection.** As a final step, the pipeline includes the highlight-

ing stylesheets actually used in the page, avoiding the cost of loading the rest.

3.3. Page Layout and Customization

The default page template uses a Bootstrap 3 grid layout with a content area (nine columns) and a sidebar (three columns):

```
+-------------------------------------------------------------+
|              page header, including the title               |
+---------------------------------------------+---------------+
|                content header                |   side bar    |
+---------------------------------------------+               |
|                                             |               |
|                  contents                   |               |
|                                             |               |
|                [9 columns]                  |  [3 columns]  |
+---------------------------------------------+---------------+
|                          page footer                        |
+-------------------------------------------------------------+
```

Each region of the page — header, content header, sidebar, and footer — is generated by an optional routine that can be provided in a separate file. The CGI script loads this file using a `Requires` directive, and calls each routine through an optional call mechanism: if the routine exists, it is called; if not, the corresponding region is simply left empty. Any of these regions other than the content area may be omitted, allowing the CGI script to serve sites with very different layouts — or none at all — without modification.

The package includes a sample customization file, `rexx.epbcn.com.optional.cls`, which provides the routines used by the `https://rexx.epbcn.com/` site: a navigation bar with a top menu and the site logo, a breadcrumb trail with the style chooser dropdown, a context-sensitive sidebar, and a copyright footer. This file is distributed as a sample to illustrate how site-specific customizations can be implemented without modifying the core CGI script.

3.4. The Style Chooser

The HTML rendered by the Web pipeline includes a style chooser dropdown that allows the reader to select a highlighting style for Rexx fenced code blocks. The mechanism is deliberately simple: the dropdown reloads the page with a `style=`*name* parameter in the query string. This parameter sets the default highlighting style for all Rexx fenced code blocks on the page. Individual blocks can still override the default by specifying `{style=`*name*`}` in the fenced code block attributes. The page reload is necessary because the highlighting style is implemented as a CSS class on the `<pre>` element, and changing it requires the page to be regenerated by the CGI.

3.5. A Live Example

The pages under `https://rexx.epbcn.com/rexx-parser/` are a live example of the Web pipeline in operation. All its documentation pages are Markdown files served through the CGI, with full Rexx syntax highlighting, a navigation sidebar, breadcrumb navigation, and the style chooser dropdown. A step-by-step installation tutorial[4] is available for anyone wishing to set up a similar configuration.

4. `https://rexx.epbcn.com/rexx-parser/doc/highlighter/cgi/`↩

3.6. The Print Variant

The same pipeline can also serve the document in a paginated, print-ready form. When the query string parameter `?print=pdf` is added to the URL of a Markdown page, the page is rendered as a print-ready article, letter, or slide deck — directly in the browser, with no additional tools required. This requires that the page use a document class that supports pagination (`article`, `letter`, `slides`, or `book`; see the Document Classes section below). The `default` class does not support pagination, and pages without a class set cannot be paginated.

The CGI script detects the `print=pdf` parameter and makes three adjustments to the normal flow:

1. **Paged.js is loaded.**[5] A `<script>` tag for `paged.polyfill.js` is injected into the page. Once loaded, paged.js takes the HTML content of the page and paginates it into discrete pages with margins, headers, footers, footnotes, and page numbers.
2. **A document class stylesheet is loaded.** The CGI selects a CSS file based on the Markdown filename: `article.md` loads `print/article.css`, `letter.md` loads `print/letter.css`, `slides.md` loads `print/slides.css`. The document class can also be specified in the YAML front matter with `docclass:` under `rexxpub:`, which takes precedence over the filename convention. These stylesheets define the document class — the page size, margins, typography, heading hierarchy, and overall visual structure.
3. **Footnotes are inlined.** Instead of the normal Pandoc invocation, the CGI uses a Pandoc Lua filter called `inline-footnotes.lua`. Normally, Pandoc collects footnotes and places them at the end of the section or the document. The Lua filter transforms each footnote into an inline `<span class="footnote">` element instead, which paged.js recognizes and moves to the footnote area at the bottom of the corresponding page, following the CSS Paged Media specification.

5. Paged.js is a JavaScript polyfill that implements most of the CSS Paged Media specification. See *Why paged.js* below for a fuller discussion.↩

4. The HTML pipeline: Static HTML Generation

The Web pipeline serves Markdown dynamically through a CGI script: every request triggers a Pandoc invocation. The cost is negligible — rexx.epbcn.com runs this way without trouble — and the model has real advantages, such as picking up edits the moment they are saved. But it requires a web server with CGI enabled, which is not always available or convenient. The HTML pipeline addresses those cases: it converts a tree of Markdown files into static HTML in a single batch run, ready to be published by any means the user prefers — a hosting service, a file share, a static-only web server, or even a local filesystem.

The md2html utility was originally written to support Jean Louis Faucher's work on Executor.[6] The Executor documentation needed full Rexx syntax highlighting, but setting up a local web server was not a practical option in that context. A batch tool that could produce static HTML with the same quality of highlighting as the Web pipeline was the natural solution. The tool soon proved useful beyond its original purpose, and it was made public as a general-purpose component of RexxPub.

4.1. Usage

```
[rexx] md2html [options] filename [destination]
[rexx] md2html [options] source [destination]
```

When the argument is a file, md2html converts that single file to HTML. When the argument is a directory, it recursively scans it for `.md` files and produces a corresponding `.html` file for each one, preserving the directory structure. *Destination* defaults to the current directory.

The available options include `--css` and `--js` to specify the location of stylesheets and JavaScript files, `--default` to set default attributes for all Rexx fenced code blocks, `--continue` to keep processing when a code block contains an error, `--path` to specify a search path for configuration files, `--section-numbers` *n* to override the section numbering depth, and `--no-number-figures` to disable automatic figure/listing numbering.

---

6. https://jlfaucher.github.io/executor/↩

4.2. Workflow

The processing of each Markdown file follows the same two-stage common core as the Web pipeline, with the output written to a file instead of being served over HTTP:

1. `FencedCode.cls` expands all Rexx fenced code blocks, if any, in the Markdown source.

2. Pandoc converts the enriched Markdown to HTML.
3. The HTML is inserted into a template (`default.md2html`) using the same placeholder mechanism as in the Web pipeline.
4. The same dynamic CSS injection mechanism as in the Web pipeline is applied.

4.3. Customization

The template and the page regions are fully customizable through two files:

- `default.md2html` defines the HTML page structure. Lines beginning with `--` are treated as comments. Placeholder markers are expanded at runtime.
- `md2html.custom.rex` provides the routines that generate the optional page regions: header, content header, sidebar, and footer. Additional routines allow the user to skip specific files, translate filenames, or specify a custom file extension for the output.

Both files are searched in order: first in a path specified via the `--path` option, then in the current directory, the destination directory, the source directory, and finally using the standard Rexx external file search order. Sample versions of both files are distributed with the Rexx Parser.

The architecture mirrors the Web pipeline's optional call mechanism: if a customization routine exists, it is called; if not, the corresponding region is silently left empty. This makes it easy to produce anything from a minimal, unstyled HTML page to a fully branded documentation site, simply by providing the appropriate configuration files.

5. The PDF pipeline: Offline PDF Generation

The Web pipeline can produce print-ready PDF through the browser's print variant, but doing so requires a running web server and a browser to print from. The PDF pipeline eliminates both requirements: the md2pdf utility converts Markdown files directly to print-quality PDF from the command line, with no server and no browser window. It can process a single file or an entire directory tree.

5.1. Usage

```
[rexx] md2pdf [options] filename
[rexx] md2pdf [options] source-directory [destination-directory]
```

When the argument is a file, the input is a single Markdown file (the `.md` extension can be omitted) and the output is a PDF file with the same name, placed in the same directory as the input. When the argument is a directory, all `.md` files in it and its subdirectories are converted to PDF.

5.2. Workflow

The processing follows the same common core as the other pipelines, but with a different final stage:

1. `FencedCode.cls` expands all Rexx fenced code blocks, if any.
2. Pandoc converts the enriched Markdown to HTML, using the `inline-footnotes.lua` Lua filter (the same one used by the Web pipeline's print variant) so that footnotes are placed correctly in the paginated output. Pandoc is also invoked with `--citeproc` and a Citation Style Language file for processing bibliographic references.
3. The HTML is assembled into a self-contained file: all CSS — Bootstrap, the document class stylesheet, and the highlighting style — is embedded directly in a `<style>` block, rather than linked externally. This produces a single HTML file that carries everything it needs.
4. pagedjs-cli is invoked to convert the HTML file to PDF.
5. Optionally, pikepdf is used to activate the document outline in the resulting PDF (the `--fix-outline` option), so that the table of contents panel opens automatically in PDF readers.

Because pagedjs-cli bundles an older version of Chromium that does not support CSS nesting, the CSS stylesheets must use flattened (un-nested) versions, generated automatically from the nested originals.

5.3. Options

- `--docclass` *class* — selects the document class. When not specified, the class is inferred from the YAML front matter (`docclass:` under `rexxpub:`) or from the source filename; if neither specifies a class, the `default` class is used as a fallback. The document class determines the page geometry, typography, and overall visual structure, just as when printing from the Web pipeline.
- `--size` *n* — selects the base font size (default: 12). The value must correspond to an existing size override stylesheet (e.g., `article-10pt.css`). Prebuilt overrides are provided at 10pt and 14pt; additional sizes can be added by creating the corresponding CSS file.
- `-c` *dir* or `--css` *dir* — sets the CSS base directory, allowing custom styles and document classes to be used independently of the Rexx Parser installation.
- `--style` *name* — sets the default highlighting style for Rexx fenced code blocks.
- `--section-numbers` *n* — overrides the section numbering depth (default: 3 for article, 2 for book, 0 for slides).
- `--no-number-figures` — disables automatic figure/listing numbering.
- `--outline` *n* — generates a PDF outline (bookmarks) from headings `<h1>` through `<h`*n*`>`. The default is 3.
- `--fix-outline` — uses pikepdf (a Python library) to modify the PDF so that the document outline panel opens automatically.
- `--csl` *name|path* — selects a Citation Style Language style for bibliographic references (default: `rexxpub`). A plain name is looked up in the `csl/` directory; a path is used as-is.
- `--continue` — in batch mode, continues processing when a file fails, rather than aborting.
- `--language` *code* — sets the document language (default: `en`).
- `--default` *"options"* — sets default attributes for all Rexx fenced code blocks.
- `--check-deps` — verifies that all external dependencies (Pandoc, Node.js, npm, pagedjs-cli) are installed.

5.4. Two Routes to PDF

RexxPub offers two ways to produce print-ready PDF. They use the same source, the same stylesheets, the same Lua filter, and the same paged.js polyfill, and in normal use produce the same output — though small rendering differences can occur.[7] The Web pipeline's print variant uses the browser as the rendering engine: the user opens the page with `?print=pdf`, paged.js paginates it in the browser, and the user prints to PDF. The PDF pipeline automates the same process from the command line: pagedjs-cli runs paged.js inside a headless Chromium, producing the PDF

without user interaction.

The main practical difference, apart from the obvious one of requiring or not requiring a server, is that the PDF pipeline needs the flattened CSS stylesheets because of the older Chromium bundled with pagedjs-cli, whereas the browser route uses the browser's own (typically up-to-date) rendering engine and can work with the original nested stylesheets.

---

7. The two routes use different versions of Chromium: the user's browser (typically current) for the Web pipeline's print variant, and an older version bundled with pagedjs-cli for the PDF pipeline. Differences are normally cosmetic — slight variations in font metrics, antialiasing, or how edge-case CSS features are rendered — but they can be visible when the same document is rendered both ways. Readers coming from LaTeX, where rendering is bit-reproducible across versions, should be aware of this trade-off.↩

6. The EPUB pipeline: EPUB Generation

The three pipelines described so far produce HTML or PDF. The EPUB pipeline addresses a different delivery format: EPUB, the standard ebook format supported by virtually every ebook reader, tablet, and smartphone.

The md2epub utility converts Markdown files to EPUB ebooks from the command line. Like md2html and md2pdf, it can process a single file or an entire directory tree.

6.1. Usage

```
[rexx] md2epub [options] filename
[rexx] md2epub [options] source-directory [destination-directory]
```

When the argument is a file, md2epub converts that single file to EPUB. When the argument is a directory, it recursively scans it for `.md` files and produces a corresponding `.epub` file for each one.

6.2. Workflow

The processing shares the same front-end as the other pipelines, but the back-end is simpler:

1. `FencedCode.cls` expands all Rexx fenced code blocks, if any.
2. Pandoc converts the enriched Markdown directly to EPUB with `--to epub`, using `--citeproc` for bibliographic references. No intermediate HTML file is produced: Pandoc packages the result into a standard EPUB file (internally XHTML + CSS + metadata in a ZIP archive).

Rexx syntax highlighting is preserved in the EPUB through embedded CSS stylesheets. Because EPUB readers do not support CSS nesting, md2epub uses the flattened versions of the highlighting stylesheets.

6.3. Differences from md2pdf

Md2epub shares the same front-end as md2pdf — `FencedCode.cls` processes Rexx fenced code blocks, `YAMLFrontMatter.cls` and `RexxPubOptions.cls` parse options — but the back-end is substantially lighter:

- **No pagedjs-cli.** Pandoc generates the EPUB directly. No headless Chromium is needed, and no Node.js installation is required.
- **No document classes.** EPUB readers control the page layout, so the `doc-class`, `size`, and `outline` options have no effect.
- **No Bootstrap.** The EPUB contains only the highlighting stylesheet and the Pandoc syntax highlighting CSS.

- **Cover image.** An optional cover image can be specified with `--cover` or via the `cover:` key under `rexxpub:` in the YAML front matter.
- **Chapter splitting.** The `--chapter-level` option (default: 1) controls at which heading level Pandoc splits the EPUB into separate internal chapters.

Unlike md2pdf, which requires Node.js, pagedjs-cli, and a headless Chromium, md2epub depends on Pandoc alone.

7. Document Classes

Document classes describe the typographic specification of a paginated document — page size, margins, typography, heading hierarchy, footnote placement, and so on. They are used by the Web pipeline when rendering pages with `?print=pdf`, and by the PDF pipeline. The HTML and EPUB pipelines do not use document classes: HTML output is unpaginated, and EPUB readers control the page layout themselves.

7.1. The Five Classes

RexxPub currently provides five document classes, each with its own CSS stylesheet:

- **`article`** — produces output resembling the LaTeX `article` document-class: DIN A4 portrait, Times New Roman at 12pt, justified text with automatic hyphenation, 1.5em first-line paragraph indent, LaTeX-style heading hierarchy, centred page numbers, and footnotes at the bottom of each page. The article you are reading was produced using this class.
- **`book`** — the most complete class, designed for long-form documents with chapters, parts, facing pages, running headers, a table of contents, and asymmetric margins for binding.
- **`default`** — a general-purpose class for Markdown files that do not match any of the named classes, intended primarily for web display rather than print. It uses serif typography with block-style paragraphs and left-aligned text, suitable for README files, tutorials, and technical documentation.
- **`letter`** — produces formal correspondence following the LaTeX `letter` conventions: DIN A4 portrait, Times New Roman at 12pt, block letter style (no indent, paragraph separation via vertical space), ragged right alignment, structured address blocks using Pandoc fenced divs (`::::: sender`, `::::: recipient`, `::::: closing`, `::::: signature`, etc.), and no page number on the first page.
- **`slides`** — produces conference presentation decks: FHD-compatible 16:9 pages (254mm × 142.875mm), Helvetica/Arial at 20pt, left-aligned text, generous list spacing for projection readability, blue accent colour for headings and rules, and discrete slide numbers in the bottom-right corner. Each top-level heading (a Markdown `#` or `===` title) marked with `{.slide}` starts a new slide.

The five classes follow the LaTeX document-class tradition: each is a self-contained typographic specification.

Parametric sizing

The `article`, `book`, `default`, and `letter` classes support parametric sizing

through CSS custom properties (`--doc-font-size`, `--doc-line-height`, `--doc-footnote-size`, `--doc-fn-marker-size`, `--doc-pre-size`). The default is 12pt. Pre-built size override stylesheets are provided at 10pt and 14pt (e.g., `article-10pt.css` or `letter-14pt.css`), and additional sizes can be added by creating the corresponding CSS file (e.g., `article-11pt.css`).

The size is selected with the `size:` option in the YAML front matter under `rexxpub:` (e.g., `size: 10`). Like the document class, the size is a writer-side decision: it belongs to the document, not to how it happens to be served or rendered.

Section numbering

Headings are automatically numbered by default, following the LaTeX convention (e.g., 1., 1.1, 1.1.1). The default depth depends on the document class: 3 for `article` (and `default`, `letter`), 2 for `book`, and 0 for `slides`. The depth can be overridden with the `sections=`*n* query string parameter in the CGI, or the `--section-numbers` *n* option in md2pdf and md2html; use 0 to disable numbering. Headings marked with Pandoc's `{.unnumbered}` or `{-}` attribute are excluded from numbering, as are headings with the `.part` or `.chapter` classes.

Figures and listing captions

Figures (Pandoc images with captions) and code listings (fenced code blocks with the `caption` attribute) are automatically numbered following the LaTeX convention ("Figure 1:", "Listing 1:", etc.). Figures and listings use independent counters. Listing captions are placed above the code block and figure captions below the image, matching the LaTeX defaults. Numbering can be disabled with `numberfigures=0` in the CGI, or `--no-number-figures` in md2pdf and md2html.

The caption system is driven by `numberFigures.js`, which operates as a paged.js handler when output is paginated (the Web pipeline in print mode and the PDF pipeline) and as an immediate script when output is plain HTML (the Web pipeline in its default mode and the HTML pipeline). Code blocks receive their captions from the `caption=` attribute, which works uniformly for both Rexx and non-Rexx blocks:

```
def greet(name):
    print(f"Hello, {name}!")

greet("Rexx")
```

Example 1: *A Python greeting function*

7.2. Per-File CSS

In addition to the document class stylesheet, the CGI checks for a CSS file with the same name as the Markdown file plus a `.css` extension (for example, `article.md.css` alongside `article.md`). If such a file exists, it is loaded as well.

This mechanism is useful for per-document customizations, such as running

headers and footers specific to a particular article. For example, paged.js supports the `string-set` property from the CSS Generated Content for Paged Media specification, which allows content from the document (such as the article title or the author name) to be captured and displayed in the page margins via `@page` rules with `@top-left`, `@top-right`, or other margin box selectors.

The per-file CSS can also be used for any other document-specific adjustments: custom styles for particular elements, overrides of the document class defaults, or experimental formatting that should not affect other documents on the site.

7.3. A Self-Referencing Example

The article you are reading is itself produced by the print variant of the Web pipeline. Its source is a Markdown file that uses the `article` document class. It can be viewed as a normal web page served by the Web pipeline, or as a print-ready paginated article by adding `?print=pdf` to the URL. The PDF version of this document was produced by printing the latter from the browser.

8. Shared Infrastructure and Design Decisions

8.1. A Single Source for All Outputs

RexxPub is built around a single architectural principle: a single Markdown source can be delivered in multiple forms — as a navigable web page, as a print-ready PDF, or as an EPUB — with no separate source, no duplicated content, and no synchronization work between versions. Technical documents typically live in two or three parallel formats — a LaTeX source for the PDF, an HTML version for the web, perhaps a separate slide deck — each requiring independent maintenance, and drifting apart over time as corrections are applied to one and forgotten in the others. The author maintains several sites, including `https://www.epbcn.com/`, where documents exist in LaTeX, PDF, and HTML simultaneously; RexxPub eliminates this burden by construction. The article you are reading is itself an instance of the principle: it is a node of `https://rexx.epbcn.com/`, served as a regular web page by the Web pipeline, and at the same time available as a print-ready PDF by appending `?print=pdf` to its URL (see A self-referencing example above).

8.2. Why Markdown + Pandoc

The choice of Markdown as the authoring format and Pandoc as the document transformation engine was driven by three requirements: simplicity of authoring, the ability to produce both HTML and print-ready PDF from the same source, and seamless Unicode support.

Pandoc emits clean HTML, which serves as the canonical intermediate format for every RexxPub pipeline. HTML is delivered directly by the Web and HTML pipelines, and converted to PDF or EPUB by the PDF and EPUB pipelines respectively. A single source thus produces every output format the framework supports.

LaTeX would have been the traditional choice for high-quality typesetting, and the Rexx Highlighter does include a LaTeX driver. However, experience with the TUTOR conference papers [4], [5] revealed serious practical difficulties when combining syntax highlighting with Unicode content. LaTeX's handling of Unicode is notoriously fragile: for example, displaying emojis requires a specific package, but using that package inside a highlighting environment leads to conflicts that are very hard to resolve. Since Unicode integration is one of the main areas of active development in the ooRexx ecosystem, this limitation is not a minor inconvenience — it makes LaTeX impractical as a target format for Rexx documentation going forward.

HTML, by contrast, handles Unicode natively: the entire character space is available through UTF-8 encoding, and syntax highlighting is simply a matter of CSS classes on `<span>` elements. There are no conflicts, no special packages, and no fragile configurations.

The Rexx Highlighter's LaTeX driver remains available as a proof-of-concept, and could be developed further by anyone interested in a pure LaTeX pipeline —

but the HTML path is where RexxPub invests its effort.

8.3. Why paged.js

Producing print-quality PDF from HTML requires a rendering engine that implements the CSS Paged Media specification. Several options exist, but most have significant drawbacks.

Prince XML is widely regarded as the most complete implementation of CSS Paged Media, but it is a commercial product: a server license costs $3,800 USD, putting it out of reach for an open-source project like RexxPub.

WeasyPrint is open-source, but it only implements a subset of the CSS Paged Media features — notably, it does not support footnotes, which are essential for conference articles like this one. It also lacks a JavaScript engine, which limits its ability to process dynamic content.

Paged.js takes a fundamentally different approach: it is a JavaScript polyfill that implements the W3C CSS Paged Media specification directly in the browser. This means that it works with exactly the same HTML and CSS that the Web pipeline already produces — no separate rendering engine, no additional transformations, no new dependencies beyond a browser.

This transparency is one of paged.js's greatest strengths. The author writes CSS Paged Media rules (for margins, running headers, footnotes, page breaks, and so on), and paged.js interprets them in the browser.

But the most compelling argument for paged.js is that it implements a W3C standard. This is an investment in the future: the CSS written today conforms to an open specification, and should continue to work regardless of what happens to any particular tool. Moreover, as browsers gradually implement more of the CSS Paged Media specification natively, paged.js will have less work to do as a polyfill — and in the limit, when browsers fully support the standard, the polyfill will no longer be needed at all, but the CSS will remain valid.

8.4. Bootstrap

The current implementation of the Web pipeline uses Bootstrap 3 for responsive layout and basic styling. This is a pragmatic choice driven by the fact that the `https://rexx.epbcn.com/` site already used Bootstrap 3 when the Web pipeline was developed, and it provides everything that is needed. The framework is not tied to any specific version of Bootstrap, nor indeed to Bootstrap at all — a different CSS framework could be used without changes to the core pipeline. A future version will likely drop the Bootstrap dependency altogether in favour of a lighter, self-contained layer.

8.5. YAML Front Matter

All RexxPub pipelines can read RexxPub options directly from the YAML front matter block of a Markdown document, alongside the standard Pandoc metadata fields (`bibliography`, `csl`, etc.). RexxPub-specific options are placed under a `rexxpub:` key to avoid any conflict with Pandoc's metadata:

```
---
bibliography: references.bib
csl: ../../../../csl/rexxpub.csl
highlight-style: pygments
rexxpub:
  docclass: article
  language: en
  section-numbers: 3
  number-figures: true
  size: 12
  outline: 4
  listings:
    caption-position: above
    caption-style: italic
    label-style: bold
    frame: tb
  figures:
    caption-position: below
---
```

The front matter is parsed by `YAMLFrontMatter.cls`, a Rexx routine that implements a subset of YAML: block mappings with scalar values, up to three levels of nesting. The parser handles blank lines, comments, and quoted values; lists, anchors, tags, flow style, and multiline scalars are not supported, as they are not needed for the current set of options.

The currently supported top-level options are `style` (the default Rexx highlighting style), `size` (the base font size), `section-numbers` (the heading numbering depth, from 0 to 4), `number-figures` (automatic figure and listing numbering, accepting `0`, `1`, `true`, or `false`), `docclass` (overrides the document class inferred from the filename or the CLI), `chapter` (sets the chapter number for the `book` class), `language` (sets the `<html lang>` attribute), and `outline` (the PDF bookmark depth, md2pdf only).

Two nested groups provide fine-grained control over captions. The `listings:` group configures code listing captions, with keys for `caption-position` (`above` or `below`, default `above`), `caption-style` (`normal` or `italic`), `label-style` (`bold`, `italic`, `bold-italic`, or `normal`), `label` (a custom text that overrides the localized label), and `frame` (`none`, `tb`, `single`, or `leftbar` — following the LaTeX `listings` package conventions — to control the border style of Pandoc-highlighted code blocks; the default is `none`). The `figures:` group offers the same four caption keys for image captions, with a default `caption-position` of `below`.

Two additional fields are standard Pandoc metadata and are placed at the top level of the YAML front matter, not under `rexxpub:`. The `csl` field selects the ci-

tation style for Pandoc's `--citeproc` processing. The `highlight-style` field selects the CSS theme for syntax highlighting of non-Rexx fenced code blocks (Python, Java, SQL, etc.); the available styles are `pygments` (the default), `kate`, `tango`, `espresso`, `zenburn`, `monochrome`, `breezeDark`, and `haddock`. This does not affect Rexx code blocks, which use the Rexx Highlighter and the `style` option under `rexxpub:`.

The precedence rules are designed to respect the intent of each actor. Structural options (`size`, `section-numbers`, `number-figures`) follow an *author-wins* policy: the YAML value takes precedence over URL parameters and command-line options, because the author knows how the document is designed. This is particularly important if cross-references are ever introduced: a reference such as "see Section 3.1 on page 12" would become meaningless if the numbering depth could be silently overridden from the URL. The highlighting `style`, by contrast, follows a *reader-wins* policy: the URL parameter (or the style chooser dropdown) takes precedence over the YAML value, because the choice of highlighting style depends on the delivery context — a document may need `dark` for screen display and `print` for conference proceedings.

9. Acknowledgements

The author wishes to thank The Rexx Language Association (RexxLA) for fostering a community where technical work can be both genuinely useful and genuinely enjoyable.

Thanks are due to Rony G. Flatscher for his generous help and support, and to Jean Louis Faucher for the many technical conversations we shared while implementing Executor support and beyond.

The author is also grateful to the members of the RexxLA Architecture Review Board, the developers' list, and the general membership list, for their varied contributions and for being such a welcoming community.

Special thanks go to the Espacio Psicoanalítico de Barcelona (EPBCN) for its generous support — computational resources, funding, time, and space — without which this work would not have been possible.

Finally, the author wishes to thank his colleagues at EPBCN — Laura Blanco, Carlos Carbonell, Silvina Fernández, Mar Martín, David Palau, Olga Palomino, and Amalia Prat — for their companionship, not only in the work of the EPBCN, but in life itself.

References

[1] Josep Maria Blasco, "The Rexx Highlighter," in *2025 International Rexx Language Symposium Proceedings*, https://www.epbcn.com/pdf/josep-maria-blasco/2025-05-06-The-Rexx-Highlighter.pdf; RexxLA Press, 2025. Held May 4–7, 2025, at the Wirtschaftsuniversität Vienna, Austria.

[2] Josep Maria Blasco, "The Rexx Parser," in *2025 International Rexx Language Symposium Proceedings*, https://www.epbcn.com/pdf/josep-maria-blasco/2025-05-05-The-Rexx-Parser.pdf; RexxLA Press, 2025. Held May 4–7, 2025, at the Wirtschaftsuniversität Vienna, Austria.

[3] Josep Maria Blasco, "The Rexx Parser Revisited: Highlights and New Features," in *2026 International Rexx Language Symposium Proceedings*, https://www.epbcn.com/pdf/josep-maria-blasco/2026-05-06-The-Rexx-Parser-Revisited.pdf; RexxLA Press, 2026. Held May 3–6, 2026, in Barcelona, Catalunya.

[4] Josep Maria Blasco, "The Unicode Tools of Rexx," in *2024 International Rexx Language Symposium Proceedings*, https://www.epbcn.com/pdf/josep-maria-blasco/2024-03-04-The-Unicode-Tools-Of-Rexx.pdf; RexxLA Press, 2024. Held March 3–6, 2024, in Brisbane, Australia.

[5] Josep Maria Blasco, "Unicode and Rexx," in *2025 International Rexx Language Symposium Proceedings*, https://www.epbcn.com/pdf/josep-maria-blasco/2025-05-04-Unicode-and-Rexx.pdf; RexxLA Press, 2025. Held May 4–7, 2025, at the Wirtschaftsuniversität Vienna, Austria.