Vivliostyle CLI Documentation

Table of Contents

Getting Started

Vivliostyle CLI is a command-line interface for typesetting HTML and Markdown documents. It includes the Vivliostyle Viewer (https://docs.vivliostyle.org/en/viewer/vivliostyle-viewer/) and generates high-quality PDFs suitable for publications.

Creating a Vivliostyle Project

Run one of the following commands. Answer the prompts, and your project will be created automatically.

```
npm create book
yarn create book # For yarn users
pnpm create book # For pnpm users
```

> *[!NOTE] You need to have Node.js (https://nodejs.org/) (v20 or later) installed beforehand. If you use a runtime other than Node.js, such as Bun or Deno, follow the instructions for your chosen runtime.*

When you create a project, a configuration file named `vivliostyle.config.js` will be generated automatically. For details on usage, please see Using Config File.

Community Templates

Vivliostyle CLI provides several default templates: Minimal, Basic, Documentation, Novel, Academic, and Magazine. If the Vivliostyle Theme you want to use offers its own template, you can select `Use templates from the community theme` during project creation to start your project from that template.

You can also use your own template. Specify the template with the `--template` option when creating a project.

```
npm create book -- --template gh:org/repo/templates/awesome-template
```

For details on referencing templates, see the giget (https://github.com/unjs/giget#readme) documentation.

Manual Installation

You can install Vivliostyle CLI with the following command:

```
npm install -g @vivliostyle/cli
```

This command makes Vivliostyle available system-wide. If you want to use Vivliostyle CLI only in the current directory, use the following command instead:

```
npm install @vivliostyle/cli
```

Generating PDFs

Generate PDFs from HTML or Markdown

Use the `vivliostyle build` command to generate a PDF from an HTML file. The default output PDF file name is "output.pdf".

```
vivliostyle build index.html
```

Similarly, specify a Markdown file to generate a PDF from the Markdown.

```
vivliostyle build manuscript.md -s A4 -o paper.pdf
```

For the Markdown syntax available in Vivliostyle CLI, refer to VFM: Vivliostyle Flavored Markdown (https://vivliostyle.github.io/vfm/#/).

Specifying the output PDF file

Specify the PDF file name with the `-o` (`--output`) option.

```
vivliostyle build book.html -o book.pdf
```

Specifying a web URL

You can also specify a web URL in addition to local HTML files.

```
vivliostyle build https://vivliostyle.github.io/vivliostyle_doc/samples/gutenberg/Alice.ht
```

Generate PDFs from other formats

Vivliostyle CLI supports reading EPUB, unzipped EPUB OPF files, `pub-manifest` (web publication manifest JSON files), and `webbook` (HTML files with links to a table of contents or web publication manifest).

```
vivliostyle build epub-sample.epub -o epub.pdf
vivliostyle build publication.json -o webpub.pdf
```

Previewing the typesetting result

Preview the typesetting result in a browser with the `vivliostyle preview` command. The browser will launch, allowing you to view the typesetting result with the Vivliostyle Viewer.

```
vivliostyle preview index.html
vivliostyle preview manuscript.md
vivliostyle preview epub-sample.epub
```

Quickly preview publications composed of many documents

To quickly preview publications composed of many documents, use the `-q` (`--quick`) option. This option uses rough page count estimation to load documents quickly (page numbers will be inaccurate).

```
vivliostyle preview index.html --quick
vivliostyle preview publication.json --quick
vivliostyle preview epub-sample.epub --quick
```

Output formats other than PDF

Vivliostyle CLI supports output in EPUB format and Web publications (WebPub) in addition to PDF format. For details, see Special Output Settings.

The matrix of supported output formats is as follows:

Input \ Output	`pdf`	`webpub`	`epub`
`pub-manifest`	🔵	🔵	🔵
`markdown`	🔵	🔵	🔵
`html` `webbook` (including external HTML)	🔵	🔵	🔵
`epub` `epub-opf`	🔵	🚧	🚧

Other options

Display a list of available options in Vivliostyle CLI with the `vivliostyle help` command.

```
vivliostyle help
vivliostyle help init
vivliostyle help build
vivliostyle help preview
```

Secret feature: Instead of the `vivliostyle` command, you can also use the command name `vs` to reduce the number of keystrokes slightly.

See also:

- Vivliostyle CLI (README) (https://github.com/vivliostyle/vivliostyle-cli/blob/main/README.md)

Using Config File

To compile multiple articles or chapter files into a single publication, use a configuration file. When you run the `vivliostyle build` or `vivliostyle preview` command, if there is a configuration file named `vivliostyle.config.js` in the current directory, it will be used. You can also create a configuration file in JSONC (https://code.visualstudio.com/docs/languages/json#_json-with-comments) (JSON with comments) format with the filename `vivliostyle.config.json`.

Creating a Configuration File

You can create a configuration file `vivliostyle.config.js` with the following command:

```
vivliostyle init
```

This will generate a `vivliostyle.config.js` file in the current directory. The created `vivliostyle.config.js` file will look like this:

```
// @ts-check
import { defineConfig } from '@vivliostyle/cli';

export default defineConfig({
  title: 'My Book Title',
  author: 'John Doe',
  language: 'en',
  image: 'ghcr.io/vivliostyle/cli:latest',
  entry: ['manuscript.md'],
});
```

Configuration File Settings

The settings in the configuration file are explained in the comments within the file (starting with `//`). Here

are the main settings for each item. For all settings, refer to the Config Reference.

- **title**: The title of the publication (e.g., `title: 'Principia'`).
- **author**: The author's name (e.g., `author: 'Isaac Newton'`).
- **language**: The language (e.g., `language: 'en'`). This will be reflected in the `lang` attribute of the HTML.
- **size**: The page size (e.g., `size: 'A4'`). For how to specify, refer to Specifying Page Size.
- **theme**: The package name of the Vivliostyle Themes to apply to the entire document, or the path to a CSS file. You can specify the following values and also specify multiple themes in an array format:
  - npm-style package name (e.g., `@vivliostyle/theme-techbook`, `./local-pkg`)
  - URL or local file path specifying a single CSS (e.g., `./style.css`, `https://example.com/style.css`)
- **entry**: Specify an array of input Markdown or HTML files. `js entry: [ { path: 'about.md', title: 'About This Book', theme: 'about.css' }, 'chapter1.md', 'chapter2.md', 'glossary.html' ],` You can specify the input as a string or in object format. In the case of object format, you can add the following properties:
  - `path`: Specify the path of the entry. This property is required, but not needed when specifying `rel: 'contents'` or `rel: 'cover'`. Refer to the following items for these specifications:
    - To output the table of contents in a location other than the beginning of the publication
    - To output the cover page to a location other than the beginning of the publication
  - `title`: Specify the title of the entry. If this property is not set, the title will be obtained from the content of the entry.
  - `theme`: Specify the package name of the Vivliostyle Themes to apply to the entry, or the path to a CSS file.
- **output**: Specify the output destination. Example: `output: 'output.pdf'`. The default is `{title}.pdf`. You can also specify multiple outputs in an array format as follows: `js output: [ './output.pdf', { path: './book', format: 'webpub', }, ],` You can specify the output as a string or in object format. In the case of object format, you can add the following properties:
  - `path`: Specify the path of the output destination. This property is required.
  - `format`: Specify the format to output (available options: `pdf`, `epub`, `webpub`). For EPUB output, refer to Output in EPUB Format, and for WebPub output, refer to Output in Web Publication (WebPub) Format.
- **workspaceDir**: Specify the directory to save intermediate files. If not specified, the default is the current directory, and the HTML files converted from Markdown will be saved in the same location as the Markdown files. Example: `workspaceDir: '.vivliostyle'`
- **toc**: If this property is specified, an HTML file `index.html` containing the table of contents will be output. For details, refer to Creating Table of Contents Page.

- **cover**: If this property is specified, an HTML file `cover.html` for the cover page will be output. For details, refer to Creating Cover Page.
- **image**: Change the Docker image to use. For details, refer to Generating with Docker.

Handling Multiple Inputs and Outputs at Once

- Example: multiple-input-and-output (https://github.com/vivliostyle/vivliostyle-cli/tree/main/examples/multiple-input-and-output)

The above configuration file options can be specified in an array. By setting them as an array, you can conveniently handle multiple inputs and outputs at once. The following example is a configuration file that converts Markdown files in the `src` directory to PDF files with the same name.

```
const fs = require('fs');
const path = require('path');

const inputDir = path.join(__dirname, 'src');
const outputDir = path.join(__dirname, 'output');
const files = fs.readdirSync(inputDir);

const vivliostyleConfig = files
  .filter((name) => name.endsWith('.md'))
  .map((name) => ({
    title: `Article ${path.basename(name, '.md')}`,
    entry: name,
    entryContext: inputDir,
    output: path.join(outputDir, `${path.basename(name, '.md')}.pdf`),
  }));
module.exports = vivliostyleConfig;
```

Config Reference

The configuration files `vivliostyle.config.js` and `vivliostyle.config.json` accept the `VivliostyleConfigSchema` for configuring the Vivliostyle CLI. You can reference the configuration's type scheme from TypeScript files.

```
import { VivliostyleConfigSchema } from '@vivliostyle/cli';
```

Alternatively, you can use the `defineConfig` helper function to define the configuration.

```
import { defineConfig } from '@vivliostyle/cli';

export default defineConfig({ ... });
```

Config API

VivliostyleConfigSchema

Type definition

```
type VivliostyleConfigSchema =
  | BuildTask[]
  | BuildTask;
```

BuildTask

Properties

- `BuildTask`
  - `entry`: (string | ContentsEntryConfig | CoverEntryConfig | ArticleEntryConfig)[] | ArticleEntryConfig | string
    Entry file(s) of the document.
  - `title`: string
    Title of the document.
  - `author`: string
    Author of the document.
  - `theme`: (ThemeConfig | string)[] | ThemeConfig | string
    Theme package path(s) or URL(s) of the CSS file.
  - `entryContext`: string
    Directory containing the referenced entry file(s).
  - `output`: (OutputConfig | string)[] | OutputConfig | string
    Output options.
  - `workspaceDir`: string
    Directory where intermediate files (e.g., manuscript HTMLs, publication.json) are saved. (default: `.vivliostyle`)
  - ~~`includeAssets`~~ *Deprecated*
    Use `copyAsset.includes` instead.
  - `copyAsset`: CopyAssetConfig
    Options for asset files to be copied when exporting output.
  - `size`: string
    PDF output size. (default: `letter`)
    - Preset: `A5`, `A4`, `A3`, `B5`, `B4`, `JIS-B5`, `JIS-B4`, `letter`, `legal`, `ledger`
    - Custom (comma-separated): `182mm,257mm` or `8.5in,11in`
  - ~~`pressReady`~~ *Deprecated*
    Use `pdfPostprocess.preflight: "press-ready"` instead
  - `pdfPostprocess`: PdfPostprocessConfig
    PDF post-processing options. When both pdfPostprocess and legacy options (pressReady, preflight, etc.) are specified, pdfPostprocess takes precedence.
  - `language`: string
    Language of the document.

- `readingProgression`: "ltr" | "rtl"
  Specifies the reading progression of the document. This is typically determined automatically by the CSS writing-mode. Use this option only if explicit configuration is needed.

- `toc`: TocConfig | boolean | string
  Options for Table of Contents (ToC) documents.

- ~~`tocTitle`~~ *Deprecated*
  Use `toc.title` instead

- `cover`: CoverConfig | string
  Options for cover images and cover page documents.

- `timeout`: number
  Timeout limit for waiting for the Vivliostyle process (in ms). (default: `300000`)

- `documentProcessor`: (option: import("@vivliostyle/vfm").StringifyMarkdownOptions, metadata: import("@vivliostyle/vfm").Metadata) => import("unified").Processor
  Custom function to provide a unified Processor for converting the source document to HTML.

- `documentMetadataReader`: (content: string) => import("@vivliostyle/vfm").Metadata
  Custom function to extract metadata from the source document content.

- `vfm`: VfmConfig
  Options for converting Markdown into a stringified format (HTML).

- `image`: string
  Docker image used for rendering.

- ~~`http`~~ *Deprecated*
  This option is enabled by default, and the file protocol is no longer supported.

- `viewer`: string
  URL of a custom viewer to display content instead of the default Vivliostyle CLI viewer. Useful for using a custom viewer with staging features (e.g., `https://vivliostyle.vercel.app/`).

- `viewerParam`: string
  Parameters for the Vivliostyle viewer (e.g., `allowScripts=false&pixelRatio=16`).

- `browser`: string
  Specify a browser type and version to launch the Vivliostyle viewer.

- `base`: string
  Base path of the served documents. (default: `/vivliostyle`)

- `server` : ServerConfig
  Options for the preview server.

- `static` : [key: (string)]: (string)[]
  Specifies static files to be served by the preview server. `js export default { static: { '/static': 'path/to/static', '/': ['root1', 'root2'], }, };`

- `temporaryFilePrefix` : string
  Prefix for temporary file names.

- `vite` : import("vite").UserConfig
  Configuration options for the Vite server.

- `viteConfigFile` : string | boolean
  Path to the Vite config file. If a falsy value is provided, Vivliostyle CLI ignores the existing Vite config file.

Type definition

```
type BuildTask = {
  entry:
    | (
        | string
        | ContentsEntryConfig
        | CoverEntryConfig
        | ArticleEntryConfig
      )[]
    | ArticleEntryConfig
    | string;
  title?: string;
  author?: string;
  theme?:
    | (ThemeConfig | string)[]
    | ThemeConfig
    | string;
  entryContext?: string;
  output?:
    | (OutputConfig | string)[]
    | OutputConfig
    | string;
  workspaceDir?: string;
  includeAssets?: string[] | string;
  copyAsset?: CopyAssetConfig;
  size?: string;
  pressReady?: boolean;
  pdfPostprocess?: PdfPostprocessConfig;
  language?: string;
  readingProgression?: "ltr" | "rtl";
  toc?: TocConfig | boolean | string;
  tocTitle?: string;
  cover?: CoverConfig | string;
  timeout?: number;
  documentProcessor?: (
    option: import("@vivliostyle/vfm").StringifyMarkdownOptions,
    metadata: import("@vivliostyle/vfm").Metadata,
  ) => import("unified").Processor;
  documentMetadataReader?: (
    content: string,
  ) => import("@vivliostyle/vfm").Metadata;
  vfm?: VfmConfig;
  image?: string;
  http?: boolean;
  viewer?: string;
  viewerParam?: string;
  browser?: string;
  base?: string;
  server?: ServerConfig;
  static?: {
    [key: string]: string[] | string;
```

```
  };
  temporaryFilePrefix?: string;
  vite?: import("vite").UserConfig;
  viteConfigFile?: string | boolean;
};
```

ContentsEntryConfig

Properties

- `ContentsEntryConfig`
  - `rel`: "contents"
  - `path`: string
  - `output`: string
  - `title`: string
  - `theme`: (ThemeConfig | string)[] | ThemeConfig | string
  - `pageBreakBefore`: "left" | "right" | "recto" | "verso"
    Specifies the page break position before this document. Useful for determining which side the first page of the document should be placed on in a two-page spread.
  - `pageCounterReset`: number
    Resets the starting page number of this document to the specified integer. Useful for controlling page numbers when including a page.

Type definition

```
type ContentsEntryConfig = {
  rel: "contents";
  path?: string;
  output?: string;
  title?: string;
  theme?:
    | (ThemeConfig | string)[]
    | ThemeConfig
    | string;
  pageBreakBefore?:
    | "left"
    | "right"
    | "recto"
    | "verso";
  pageCounterReset?: number;
};
```

ThemeConfig

Properties

- `ThemeConfig`
  - `specifier`: string
    The specifier name for importing the theme package or the path to a CSS file.
    - An npm-style package argument is allowed (e.g., `@vivliostyle/theme-academic@1`, `./local-pkg`).
    - A URL or a local path to a CSS file is allowed (e.g., `./style.css`, `https://example.com/style.css`).
  - `import`: (string)[] | string
    The path(s) to the CSS file(s) to import from the package. Specify this if you want to import files other than the default.

Type definition

```
type ThemeConfig = {
  specifier: string;
  import?: string[] | string;
};
```

CoverEntryConfig

Properties

- `CoverEntryConfig`
  - `rel`: "cover"
  - `path`: string
  - `output`: string
  - `title`: string
  - `theme`: (ThemeConfig | string)[] | ThemeConfig | string
  - `imageSrc`: string
  - `imageAlt`: string
  - `pageBreakBefore`: "left" | "right" | "recto" | "verso"
    Specifies the page break position before this document. Useful for determining which side the first page of the document should be placed on in a two-page spread.

Type definition

```
type CoverEntryConfig = {
  rel: "cover";
  path?: string;
  output?: string;
  title?: string;
  theme?:
    | (ThemeConfig | string)[]
    | ThemeConfig
    | string;
  imageSrc?: string;
  imageAlt?: string;
  pageBreakBefore?:
    | "left"
    | "right"
    | "recto"
    | "verso";
};
```

ArticleEntryConfig

Properties

- `ArticleEntryConfig`
  - `path`: string
  - `output`: string
  - `title`: string
  - `theme`: (ThemeConfig | string)[] | ThemeConfig | string
  - `encodingFormat`: string
  - `rel`: (string)[] | string
  - `documentProcessor`: (option: import("@vivliostyle/vfm").StringifyMarkdownOptions, metadata: import("@vivliostyle/vfm").Metadata) => import("unified").Processor
    Custom function to provide a unified Processor for converting the source document to HTML.
  - `documentMetadataReader`: (content: string) => import("@vivliostyle/vfm").Metadata
    Custom function to extract metadata from the source document content.

Type definition

```
type ArticleEntryConfig = {
  path: string;
  output?: string;
  title?: string;
  theme?:
    | (ThemeConfig | string)[]
    | ThemeConfig
    | string;
  encodingFormat?: string;
  rel?: string[] | string;
  documentProcessor?: (
    option: import("@vivliostyle/vfm").StringifyMarkdownOptions,
    metadata: import("@vivliostyle/vfm").Metadata,
  ) => import("unified").Processor;
  documentMetadataReader?: (
    content: string,
  ) => import("@vivliostyle/vfm").Metadata;
};
```

OutputConfig

Properties

- `OutputConfig`
  - `path`: string
    Specifies the output file name or directory. (default: `<title>.pdf`)
  - `format`: "pdf" | "epub" | "webpub"
    Specifies the output format.
  - `renderMode`: "local" | "docker"
    If set to `docker`, Vivliostyle will render the PDF using a Docker container. (default: `local`)
  - ~~`preflight`~~ *Deprecated*
    Use `pdfPostprocess.preflight` instead
  - ~~`preflightOption`~~ *Deprecated*
    Use `pdfPostprocess.preflightOption` instead
  - `pdfPostprocess`: PdfPostprocessConfig
    PDF post-processing options. When both pdfPostprocess and legacy options (pressReady, preflight, etc.) are specified, pdfPostprocess takes precedence.

Type definition

```
type OutputConfig = {
  path: string;
  format?: "pdf" | "epub" | "webpub";
  renderMode?: "local" | "docker";
  preflight?:
    | "press-ready"
    | "press-ready-local";
  preflightOption?: string[];
  pdfPostprocess?: PdfPostprocessConfig;
};
```

PdfPostprocessConfig

PDF post-processing options. When both pdfPostprocess and legacy options (pressReady, preflight, etc.) are specified, pdfPostprocess takes precedence.

Properties

- `PdfPostprocessConfig`
  - `preflight`: "press-ready" | "press-ready-local"
    Apply the process to generate a print-ready PDF.
  - `preflightOption`: (string)[]
    Options for the preflight process (e.g., `gray-scale`, `enforce-outline`). Refer to the press-ready documentation for more information: press-ready (https://github.com/vibranthq/press-ready)
  - `cmyk`: boolean | CmykConfig
    Convert device-cmyk() colors to CMYK in the output PDF. Can be a boolean or a config object with overrideMap and warnUnmapped options.
  - `replaceImage`: (ReplaceImageEntry)[]
    Replace images in the output PDF. Each entry specifies a source image path and its replacement image path. Useful for replacing RGB images with CMYK versions.

Type definition

```
type PdfPostprocessConfig = {
  preflight?:
    | "press-ready"
    | "press-ready-local";
  preflightOption?: string[];
  cmyk?: boolean | CmykConfig;
  replaceImage?: ReplaceImageEntry[];
};
```

CmykConfig

Properties

- `CmykConfig`
  - `overrideMap`: ("{tuple(Array)}")[]
    Custom RGB to CMYK color mapping. Each entry is a tuple of [rgb, {c, m, y, k}]. RGB can be an object {r, g, b} with integers (0-10000) or a hex color string (e.g. "#ff0000").
  - `reserveMap`: ("{tuple(Array)}")[]
    Pre-register RGB to CMYK color mappings for use in SVG or other non-CSS contexts. Each entry is a tuple of [rgb, {c, m, y, k}]. RGB can be an object {r, g, b} with integers (0-10000) or a hex color string (e.g. "#ff0000").
  - `warnUnmapped`: boolean
    Warn when RGB colors not mapped to CMYK are encountered. (default: true)
  - `mapOutput`: string
    Output the CMYK color map to a JSON file at the specified path.

Type definition

```
type CmykConfig = {
  overrideMap?: "{tuple(Array)}"[];
  reserveMap?: "{tuple(Array)}"[];
  warnUnmapped?: boolean;
  mapOutput?: string;
};
```

ReplaceImageEntry

Properties

- `ReplaceImageEntry`
  - `source`: string | RegExp
    Path to the source image file, or a RegExp pattern to match multiple files.
  - `replacement`: string
    Path to the replacement image file. When source is a RegExp, supports $1, $2, etc. for captured groups.

Type definition

```
type ReplaceImageEntry = {
  source: string | RegExp;
  replacement: string;
};
```

CopyAssetConfig

Properties

- `CopyAssetConfig`
  - `includes`: (string)[]
    Directories and files to include as asset files. Supports wildcard characters for glob patterns.
  - `excludes`: (string)[]
    Directories and files to exclude from asset files. Supports wildcard characters for glob patterns.
  - `includeFileExtensions`: (string)[]
    File extensions to include as asset files. (default: `[css, css.map, png, jpg, jpeg, svg, gif, webp, apng, ttf, otf, woff, woff2]`)
  - `excludeFileExtensions`: (string)[]
    File extensions to exclude from asset files.

Type definition

```
type CopyAssetConfig = {
  includes?: string[];
  excludes?: string[];
  includeFileExtensions?: string[];
  excludeFileExtensions?: string[];
};
```

TocConfig

Properties

- `TocConfig`
  - `title`: string
    Title of the generated ToC document.
  - `htmlPath`: string
    Location where the generated ToC document will be saved. (default: `index.html`)
  - `sectionDepth`: number
    Depth of sections to include in the ToC document. (default: `0`)
  - `transformDocumentList`: (nodeList: StructuredDocument[]) => (propsList: { children: any }[]) => any
    Function to transform the document list.
  - `transformSectionList`: (nodeList: StructuredDocumentSection[]) => (propsList: { children: any }[]) => any
    Function to transform the section list.

Type definition

```
type TocConfig = {
  title?: string;
  htmlPath?: string;
  sectionDepth?: number;
  transformDocumentList?: (
    nodeList: StructuredDocument[],
  ) => (
    propsList: { children: any }[],
  ) => any;
  transformSectionList?: (
    nodeList: StructuredDocumentSection[],
  ) => (
    propsList: { children: any }[],
  ) => any;
};
```

StructuredDocument

Properties

- `StructuredDocument`
  - `title` : string
  - `href` : string
  - `children` : (StructuredDocument)[]
  - `sections` : (StructuredDocumentSection)[]

Type definition

```
type StructuredDocument = {
  title: string;
  href: string;
  children: StructuredDocument[];
  sections?: StructuredDocumentSection[];
};
```

StructuredDocumentSection

Properties

- `StructuredDocumentSection`
  - `headingHtml` : string
  - `headingText` : string
  - `level` : number
  - `children` : (StructuredDocumentSection)[]
  - `href` : string
  - `id` : string

Type definition

```
type StructuredDocumentSection = {
  headingHtml: string;
  headingText: string;
  level: number;
  children: StructuredDocumentSection[];
  href?: string;
  id?: string;
};
```

CoverConfig

Properties

- `CoverConfig`
  - `src`: string
    Path to the cover image for the cover page.
  - `name`: string
    Alternative text for the cover image.
  - `htmlPath`: string | boolean
    Path where the generated cover document will be saved. (default: `cover.html`) If set to a falsy value, the cover document will not be generated.

Type definition

```
type CoverConfig = {
  src: string;
  name?: string;
  htmlPath?: string | boolean;
};
```

VfmConfig

Properties

- `VfmConfig`
  - `style`: (string)[] | string
    Path(s) or URL(s) to custom stylesheets.
  - `partial`: boolean
    Output markdown fragments instead of a full document.
  - `title`: string
    Title of the document (ignored in partial mode).
  - `language`: string
    Language of the document (ignored in partial mode).
  - `replace`: (test: RegExp; match: (result: RegExpMatchArray, h: any) => Object)[]
    string
    Handlers for replacing matched HTML strings.
  - `hardLineBreaks`: boolean
    Insert `<br>` tags at hard line breaks without requiring spaces.
  - `disableFormatHtml`: boolean
    Disable automatic HTML formatting.
  - `math`: boolean
    Enable support for math syntax.
  - `imgFigcaptionOrder`: "img-figcaption" | "figcaption-img"
    Order of img and figcaption elements in figure.
  - `assignIdToFigcaption`: boolean
    Assign ID to figcaption instead of img/code.
  - `footnote`: "pandoc" | "dpub" | "gcpm" | mode: "pandoc"
    "dpub" | "gcpm"
    Footnote output mode. Default is `'pandoc'` (endnote section).

Type definition

```
type VfmConfig = {
  style?: string[] | string;
  partial?: boolean;
  title?: string;
  language?: string;
  replace?: {
    test: RegExp;
    match: (
      result: RegExpMatchArray,
      h: any,
    ) => Object | string;
  }[];
  hardLineBreaks?: boolean;
  disableFormatHtml?: boolean;
  math?: boolean;
  imgFigcaptionOrder?:
    | "img-figcaption"
    | "figcaption-img";
  assignIdToFigcaption?: boolean;
  footnote?:
    | "pandoc"
    | "dpub"
    | "gcpm"
    | {
        mode:
          | "pandoc"
          | "dpub"
          | "gcpm";
      };
};
```

ServerConfig

Properties

- `ServerConfig`
  - `host`: boolean | string
    IP address the server should listen on. Set to `true` to listen on all addresses. (default: `true` if a PDF build with Docker render mode is required, otherwise `false`)
  - `port`: number
    Port the server should listen on. (default: `13000`)
  - `proxy`: [key: (string)]: import("vite").ProxyOptions[string]
    Custom proxy rules for the Vivliostyle preview server.
  - `allowedHosts`: (string)[] | boolean
    The hostnames that are allowed to respond to. Set to `true` to allow all hostnames. See `server.allowedHosts` option of Vite (https://vite.dev/config/server-options.html#server-allowedhosts) for more details.

Type definition

```
type ServerConfig = {
  host?: boolean | string;
  port?: number;
  proxy?: {
    [key: string]:
      | import("vite").ProxyOptions
      | string;
  };
  allowedHosts?: string[] | boolean;
};
```

Themes and CSS

To add styling such as fonts and text sizes to your manuscript, apply a Cascading Style Sheet (CSS), similar to how you would with an HTML file.

Adding Additional Stylesheets

To use additional stylesheets (CSS files) alongside those specified in the HTML file, use the `--style` option.

```
vivliostyle build example.html --style additional-style.css
```

The stylesheet specified this way will be treated the same as the author stylesheet (https://developer.mozilla.org/en-US/docs/Web/CSS/Cascade#author_stylesheets) specified in the HTML file. Since it is specified later, it will override the styles in the HTML file according to CSS cascading rules.

Specifying User Stylesheets

To use a user stylesheet (https://developer.mozilla.org/en-US/docs/Web/CSS/Cascade#user_stylesheets), specify the stylesheet with the `--user-style` option. User stylesheets do not override author stylesheets unless the style is specified with `!important`.

```
vivliostyle build example.html --user-style user-style.css
```

Specifying CSS Content Directly

By using the `--css` option, you can pass the stylesheet directly as CSS text. This option is useful for setting simple stylesheets or CSS variables.

```
vivliostyle build example.html --css "body { background-color: lime; }"
```

Specifying Page Size

You can specify the page size with the `-s` (`--size`) option. The sizes you can specify are A5, A4, A3, B5, B4, JIS-B5, JIS-B4, letter, legal, ledger, or you can specify the width and height separated by a comma.

```
vivliostyle build paper.html -s A4 -o paper.pdf
vivliostyle build letter.html -s letter -o letter.pdf
vivliostyle build slide.html -s 10in,7.5in -o slide.pdf
```

This option is equivalent to `--css "@page { size: <size>; }"`.

Specifying Crop Marks

By using the `-m` (`--crop-marks`) option, crop marks (indicators of the cutting position for printed materials) will be added to the output PDF.

```
vivliostyle build example.html -m
```

You can specify the bleed width when adding crop marks with the `--bleed` option. You can also specify the width outside the crop marks with the `--crop-offset` option.

```
vivliostyle build example.html -m --bleed 5mm
vivliostyle build example.html -m --crop-offset 20mm
```

This option is equivalent to `--css "@page { marks: crop cross; bleed: <bleed>; crop-offset: <crop-offset>; }"`.

About Vivliostyle Themes

- Vivliostyle Themes (https://vivliostyle.github.io/themes/)

Vivliostyle Themes is an official collection of style themes used when creating publications with Vivliostyle. By referring to Vivliostyle Themes, you can apply styles without preparing your own CSS.

Finding Themes

To find themes published as npm packages, search for the keyword "vivliostyle-theme" on npm (https://www.npmjs.com/):

- List of Themes (npm) (https://www.npmjs.com/search?q=keywords%3Avivliostyle-theme)

Using Themes

- Example: theme-css (https://github.com/vivliostyle/vivliostyle-cli/tree/main/examples/theme-css)
- Example: theme-preset (https://github.com/vivliostyle/vivliostyle-cli/tree/main/examples/theme-preset)

You can use a theme by specifying the `-T` (`--theme`) option or `theme` in the configuration file. If the theme file does not exist locally, it will be automatically installed in the `themes` directory on the first run.

```
vivliostyle build manuscript.md --theme @vivliostyle/theme-techbook -o paper.pdf
```

You can also use themes available in your local environment. If it is a single CSS file, you can specify the CSS file directly as follows.

```
vivliostyle build manuscript.md --theme ./my-theme/style.css -o paper.pdf
```

If there is a `package.json` file that conforms to npm in your local environment, you can also load the Vivliostyle Theme in that directory. The following is an example where a package available as a Vivliostyle Theme is placed in the `my-theme` directory.

```
vivliostyle build manuscript.md --theme ./my-theme -o paper.pdf
```

Using Create Book

By using Create Book, you can easily create a project with a theme already set. Refer to Create Book (https://docs.vivliostyle.org/en/cli/getting-started/).

Creating Cover Page

If the configuration file `vivliostyle.config.js` specifies `cover: 'image.png'`, a cover HTML file `cover.html` will be generated and added as the cover page of the publication.

The content of the generated cover HTML file will be as follows:

```
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>Book Title</title>
    <style data-vv-style>
      body {
        margin: 0;
      }
      [role="doc-cover"] {
        display: block;
        width: 100vw;
        height: 100vh;
        object-fit: contain;
      }
      @page {
        margin: 0;
      }
    </style>
  </head>
  <body>
    <section role="region" aria-label="Cover">
      <img role="doc-cover" src="image.png" alt="Cover image" />
    </section>
  </body>
</html>
```

The `cover` can also be specified as an object with the following properties:

- By specifying `src`, you can set the cover image to be loaded.
- By specifying `htmlPath`, you can output the cover HTML file to a file other than `cover.html`. Setting it to `false` prevents the cover HTML file from being output. In this case, the cover page will not be included in the PDF output but will be set as the cover image when output in EPUB or WebPub format.
- By specifying `name`, you can change the alt text of the cover image.

```
cover: {
  src: 'image.png',
  htmlPath: 'toc.html',
  name: 'My awesome cover image',
},
```

To output the cover page to a location other than the beginning of the publication

By specifying `{ rel: 'cover' }` as an element of the `entry` array in the configuration file `vivliostyle.config.js`, the cover HTML file will be generated at that position.

```
entry: [
  'titlepage.md',
  { rel: 'cover' },
  'chapter1.md',
  ...
],
cover: 'image.png',
```

With this, the first HTML file of the publication will be `titlepage.html`, followed by the cover HTML file `cover.html`.

You can also add multiple cover pages. The following example adds different cover pages at the beginning and end of the publication.

```
entry: [
  {
    rel: 'cover',
    output: 'front-cover.html',
  },
  ...
  {
    rel: 'cover',
    output: 'back-cover.html',
    imageSrc: 'back.png',
  },
],
cover: 'front.png',
```

To customize the cover page

- Example: customize-generated-content (https://github.com/vivliostyle/vivliostyle-cli/tree/main/examples/customize-generated-content)

To customize the cover page, specify the `path` and `output` of the cover file and `rel: 'contents'` as elements of the `entry` array in the configuration file as follows:

```
entry: [
  {
    path: 'cover-template.html',
    output: 'cover.html',
    rel: 'cover'
  },
  ...
],
```

Then, prepare the HTML file `cover-template.html` as the cover template. By including a tag `<img role="doc-cover" />` in `cover-template.html`, the cover image will be inserted at that part and the cover HTML file will be output to `cover.html`.

Creating Table of Contents Page

- Example: table-of-contents (https://github.com/vivliostyle/vivliostyle-cli/tree/main/examples/table-of-contents)

If the configuration file `vivliostyle.config.js` specifies `toc: true`, a table of contents HTML file named `index.html` will be generated as the first file of the publication.

The content of the generated table of contents HTML file will include the `title` and `h1` elements, which will output the title of the publication as specified in the configuration file's `title`.

```
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <title>Book Title</title>
    <link href="publication.json" rel="publication" type="application/ld+json" />
    <link href="style.css" rel="stylesheet" type="text/css" />
  </head>
  <body>
    <h1>Book Title</h1>
    <nav id="toc" role="doc-toc">
      <h2>Table of Contents</h2>
      <ol>
        <li><a href="prologue.html">Prologue</a></li>
        <li><a href="chapter1.html">Chapter 1</a></li>
        <li><a href="chapter2.html">Chapter 2</a></li>
        <li><a href="chapter3.html">Chapter 3</a></li>
        <li><a href="epilogue.html">Epilogue</a></li>
      </ol>
    </nav>
  </body>
</html>
```

The `toc` can also be specified as an object with the following properties:

- By specifying `htmlPath`, the table of contents HTML file will be output to a file other than `index.html`.
- By specifying `title`, the table of contents title (the content of the `h2` element within the `nav` element) will be changed.
- To include headings within the entries in addition to references to each entry, specify a value from `1` to `6` for `sectionDepth` (indicating which levels from `h1` to `h6` to include).

```
toc: {
  htmlPath: 'toc.html',
  title: 'Contents',
  sectionDepth: 6,
},
```

To output the table of contents in a location other than the beginning of the publication

By specifying `{ rel: 'contents' }` as an element of the `entry` array in the configuration file `vivliostyle.config.js`, the table of contents HTML file will be generated at that position.

```
entry: [
  'titlepage.md',
  { rel: 'contents' },
  'chapter1.md',
  ...
],
toc: 'toc.html',
```

As a result, the first HTML file of the publication will be `titlepage.html`, followed by the table of contents HTML file `toc.html`.

To customize the table of contents

- Example: customize-generated-content (https://github.com/vivliostyle/vivliostyle-cli/tree/main/examples/customize-generated-content)

To customize the table of contents, specify the `path`, `output`, and `rel: 'contents'` of the table of contents file as elements of the `entry` array in the configuration file as follows:

```
entry: [
  {
    path: 'toc-template.html',
    output: 'index.html',
    rel: 'contents'
  },
  ...
],
```

Then, prepare the HTML file `toc-template.html` as a template for the table of contents. By including the tag `<nav role="doc-toc"></nav>` in `toc-template.html`, the table of contents items will be inserted into that part, and the table of contents HTML file will be output to `index.html`.

For detailed instructions on creating a table of contents, refer to the W3C Publication Manifest (https://www.w3.org/TR/pub-manifest/) specification's Machine-Processable Table of Contents (https://www.w3.org/TR/pub-manifest/#app-toc-structure).

Special Output Settings

Output in EPUB Format

To output in EPUB format, use the `-f` (`--format`) option with the `vivliostyle build` command, specifying `epub`, or use the `.epub` extension with the `-o` (`--output`) option.

When outputting in EPUB format, it is recommended to set the table of contents. Refer to Creating Table of Contents Page and configure `toc` in the configuration file. Then, set EPUB in the output options as follows:

```
vivliostyle build -o output.epub
```

You can also generate both PDF and EPUB in a single `vivliostyle build` command as follows (this applies to other output formats as well):

```
vivliostyle build -o pdfbook.pdf -o epubbook.epub
```

Vivliostyle CLI generates EPUB files compliant with EPUB 3. However, the CSS files applied to EPUB are output as they are, which may cause display issues depending on the EPUB viewer. To support more EPUB viewers, apply themes or CSS files compatible with each viewer. For Japanese EPUBs, we provide the Vivliostyle Theme "@vivliostyle/theme-epub3j" (https://github.com/vivliostyle/themes/tree/main/packages/%40vivliostyle/theme-epub3j) compliant with the EBPAJ EPUB 3 File Creation Guide (https://dpfj.or.jp/counsel/guide).

Output in Web Publication (WebPub) Format

To generate a Web Publication (WebPub), specify `webpub` with the `-f` (`--format`) option in the `vivliostyle build` command. Specify the directory to place the WebPub with the `-o` (`--output`) option.

```
vivliostyle build -o webpub/ -f webpub
```

The generated WebPub directory contains a publication manifest `publication.json` file, which describes information such as the loading order of the HTML files of the content. This complies with the W3C standard specification Publication Manifest (https://www.w3.org/TR/pub-manifest/).

WebPub can be used to create publications that can be read on the web. You can also generate a PDF from WebPub by specifying the `publication.json` file to the `vivliostyle build` command as follows:

```
vivliostyle build webpub/publication.json -o pdfbook.pdf
```

Generating Print-Ready PDF (PDF/X-1a Format)

- Example: preflight (https://github.com/vivliostyle/vivliostyle-cli/tree/main/examples/preflight)

To output in PDF/X-1a format suitable for print submission, specify the `--preflight press-ready` option in the `vivliostyle build` command, or specify `preflight: 'press-ready'` in the configuration file. To use this feature, you need to install Docker (https://docs.docker.com/get-started/get-docker/) in advance.

By specifying the `--preflight-option` option, you can add options to press-ready (https://github.com/vibranthq/press-ready) that performs this processing.

```
# Output in grayscale
vivliostyle build manuscript.md --preflight press-ready --preflight-option gray-scale
# Force outline fonts and output
vivliostyle build manuscript.md --preflight press-ready --preflight-option enforce-outline
```

You can also specify the `--preflight press-ready-local` option to execute the output to PDF/X-1a format in the local environment. However, it is generally recommended to execute it in the Docker environment.

Generating with Docker

- Example: render-on-docker (https://github.com/vivliostyle/vivliostyle-cli/tree/main/examples/render-on-docker)

To specify Docker as the environment for PDF output, use the `--render-mode docker` option in the `vivliostyle build` command (the above option only executes post-processing on Docker, but this option executes all processing on Docker). This option ensures that all processing is executed on Docker, fixing the environment at the time of output and ensuring consistent results across different environments and OS.

When using Docker render mode, please note the following points:

- Docker is isolated from the host environment, so you cannot use fonts installed on the host. The fonts available by default in the Docker container are limited. You usually need to place local font files and specify them in CSS, or use web fonts such as Google Fonts.
- The files mounted on Docker are only the project workspace directory (usually the directory containing `vivliostyle.config.js`), and other files cannot be referenced from inside the Docker container. All files referenced in the document, such as images, must be included in the workspace directory.

Generating PDF Bookmarks

The PDF output by the `vivliostyle build` command generates bookmarks based on the table of contents. PDF bookmarks can be used for table of contents navigation in PDF viewing software such as Adobe Acrobat.

This bookmark generation feature is enabled when the publication includes a table of contents. When generating PDF from EPUB, the table of contents included in the EPUB is used. For other cases, refer to Creating a Table of Contents.

Frontend Framework Support

Vivliostyle CLI creates publications using web technologies. By combining it with other web frontend frameworks, you can leverage powerful features. For example, you can export the same manuscript as both a web page and a publication.

Referencing Statically Built HTML Files

If your framework has a static site build feature, you can reference the built HTML files using the `static` option in Vivliostyle CLI. This allows you to load those files as entries.

For example, consider a case where HTML files are built in the `dist` directory. This directory contains `index.html` and, within the `blog` directory, `my-first-post.html` and `another-post.html`. To create a publication from these three HTML files, configure it as follows:

```
{
  static: {
    '/': 'dist',
  },
  entry: [
    '/index.html',
    '/blog/my-first-post.html',
    '/blog/another-post.html',
  ],
};
```

> *[!IMPORTANT] Entries hosted with `static` must be referenced using absolute paths (paths starting with `/`). Otherwise, the HTML files will be loaded relative to `vivliostyle.config.js` instead of the directory specified in `static`.*

> *[!NOTE] While it is also possible to specify HTML files directly in `entry` using relative paths, using the `static` option is more convenient. For example, directly specifying files does not allow external files (such as images) referenced by the HTML to be loaded, but using the `static` option ensures that all files under the directory are referenced and displayed correctly.*

Using Frameworks Based on Vite

Vivliostyle CLI is developed based on Vite, a web frontend development tool. Therefore, it can be used in combination with frameworks based on Vite.

Vivliostyle CLI provides the following two usage methods depending on the type of framework.

1. Using Vite Plugins

Vivliostyle CLI allows you to use Vite or Rollup plugins. You can configure plugins and Vite settings in the `vite` option.

```
{
  vite: {
    plugins: [...],
  },
}
```

Vivliostyle Viewer differs from typical web applications in that it has restrictions on JavaScript execution. Please note the following points:

- Vivliostyle Viewer does not support client-side routing technologies (Single Page Applications; SPA), so each page must be statically rendered. If the environment does not work with `appType` set to `mpa`, this method cannot be used.
- Vivliostyle Viewer cannot detect changes to HTML content on the client side. Therefore, the UI framework used must support server-side rendering (SSR). If client-side JavaScript is included, it may not function correctly.

2. Using Vivliostyle CLI as a Vite Plugin

- Example: with-astro (https://github.com/vivliostyle/vivliostyle-cli/tree/main/examples/with-astro)
- Example: with-eleventy (https://github.com/vivliostyle/vivliostyle-cli/tree/main/examples/with-eleventy)

Some frameworks (e.g., Astro (https://astro.build/)) can accept external Vite plugins but cannot be used as Vite plugins themselves. In such cases, Vivliostyle CLI is used as a Vite plugin.

For example, in Astro, you can pass the Vite plugin of Vivliostyle CLI to the `vite.plugins` option in `astro.config.js` to use Astro and Vivliostyle CLI simultaneously. In the example below, the `openViewer: true` option is specified to automatically open the Vivliostyle Viewer when the Astro dev server starts.

```
import { createVitePlugin } from '@vivliostyle/cli';
import { defineConfig } from 'astro/config';

export default defineConfig({
  vite: {
    plugins: [
      createVitePlugin({
        openViewer: true,
      }),
    ],
  },
});
```

JavaScript API

Exported members

Functions

- `build`
- `create`
- `createVitePlugin`
- `defineConfig`
- `preview`
- `VFM`

Interfaces

- `StringifyMarkdownOptions`
- `TemplateVariable`

Type Aliases

- `Metadata`
- `StructuredDocument`
- `StructuredDocumentSection`
- `VivliostyleConfigSchema`
- `VivliostylePackageMetadata`
- `VivliostylePackageMetadata`

Variables

- `readMetadata`

Functions

build()

> ***build***(`options`): `Promise` < `void` >

Build publication file(s) from the given configuration.

```
import { build } from '@vivliostyle/cli';
build({
  configPath: './vivliostyle.config.js',
  logLevel: 'silent',
});
```

Parameters

options

author?

`string` = `...`

bleed?

`string` = `...`

browser?

`string` = `...`

cmyk?

`boolean` | { `mapOutput?`: `string`; `overrideMap?`: [`string` | { `b`: `number`; `g`: `number`; `r`: `number`; }, { `c`: `number`; `k`: `number`; `m`: `number`; `y`: `number`; }][]; `reserveMap?`: [`string` | { `b`: `number`; `g`: `number`; `r`: `number`; }, { `c`: `number`; `k`: `number`; `m`: `number`; `y`: `number`; }][]; `warnUnmapped?`: `boolean`; } = `CmykSchema`

config?

`string` = `...`

configData?

`VivliostyleConfigSchema` | `null` = `...`

createConfigFileOnly?

`boolean` = `...`

cropMarks?

`boolean` = `...`

cropOffset?

`string` = `...`

css?

`string` = `...`

cwd?

`string` = `...`

disableServerStartup?

`boolean` = `...`

enableStaticServe?

`boolean` = `...`

enableViewerStartPage?

`boolean` = `...`

executableBrowser?

`string` = `...`

host?

`string` | `boolean` = `...`

ignoreHttpsErrors?

`boolean` = `...`

image?

`string` = `...`

input?

`string` = `...`

installDependencies?

`boolean` = `...`

language?

`string` = `...`

logger?

`LoggerInterface` = `...`

logLevel?

`"info"` | `"silent"` | `"verbose"` | `"debug"` = `...`

openViewer?

`boolean` = `...`

output?

`string` | `object` & `object` | (`string` | `object` & `object`)[] = `...`

port?

`number` = `...`

preflight?

`"press-ready"` | `"press-ready-local"` = `...`

preflightOption?

`string` | `string`[] = `...`

pressReady?

`boolean` = `...`

projectPath?

`string` = `...`

proxyBypass?

`string` = `...`

proxyPass?

`string` = `...`

proxyServer?

`string` = `...`

proxyUser?

`string` = `...`

quick?

`boolean` = `...`

readingProgression?

`"ltr"` | `"rtl"` = `...`

renderMode?

`"local"` | `"docker"` = `...`

sandbox?

`boolean` = `...`

signal?

`AbortSignal` = `...`

singleDoc?

`boolean` = `...`

size?

`string` = `...`

stderr?

`Writable` = `...`

stdin?

`Readable` = `...`

stdout?

`Writable` = `...`

style?

`string` = `...`

template?

`string` = `...`

theme?

`string` | `false` | `object` & `object` | (`string` | `object` & `object`)[] = `...`

timeout?

`number` = `...`

title?

`string` = `...`

userStyle?

`string` = `...`

viewer?

`string` = `...`

viewerParam?

`string` = `...`

vite?

`UserConfig` = `...`

viteConfigFile?

`string` | `boolean` = `...`

Returns

`Promise` < `void` >

---

create()

> ***create***(*`options`*): *`Promise`* < *`void`* >

Scaffold a new Vivliostyle project.

Parameters

options

author?

`string` = `...`

bleed?

`string` = `...`

browser?

`string` = `...`

cmyk?

`boolean` | { `mapOutput?`: `string`; `overrideMap?`: [`string` | { `b`: `number`; `g`: `number`; `r`: `number`; }, { `c`: `number`; `k`: `number`; `m`: `number`; `y`: `number`; }][]; `reserveMap?`: [`string` | { `b`: `number`; `g`: `number`; `r`: `number`; }, { `c`: `number`; `k`: `number`; `m`: `number`; `y`: `number`; }][]; `warnUnmapped?`: `boolean`; } = `CmykSchema`

config?

`string` = `...`

configData?

`VivliostyleConfigSchema` | `null` = `...`

createConfigFileOnly?

`boolean` = `...`

cropMarks?

`boolean` = `...`

cropOffset?

`string` = `...`

css?

`string` = `...`

cwd?

`string` = `...`

disableServerStartup?

`boolean` = `...`

enableStaticServe?

`boolean` = `...`

enableViewerStartPage?

`boolean` = `...`

executableBrowser?

`string` = `...`

host?

`string` | `boolean` = `...`

ignoreHttpsErrors?

`boolean` = `...`

image?

`string` = `...`

input?

`string` = `...`

installDependencies?

`boolean` = `...`

language?

`string` = `...`

logger?

`LoggerInterface` = `...`

logLevel?

`"info"` | `"silent"` | `"verbose"` | `"debug"` = `...`

openViewer?

`boolean` = `...`

output?

`string` | `object` & `object` | (`string` | `object` & `object`)[] = `...`

port?

`number` = `...`

preflight?

`"press-ready"` | `"press-ready-local"` = `...`

preflightOption?

`string` | `string`[] = `...`

pressReady?

`boolean` = `...`

projectPath?

`string` = `...`

proxyBypass?

`string` = `...`

proxyPass?

`string` = `...`

proxyServer?

`string` = `...`

proxyUser?

`string` = `...`

quick?

`boolean` = `...`

readingProgression?

`"ltr"` | `"rtl"` = `...`

renderMode?

`"local"` | `"docker"` = `...`

sandbox?

`boolean` = `...`

signal?

`AbortSignal` = `...`

singleDoc?

`boolean` = `...`

size?

`string` = `...`

stderr?

`Writable` = `...`

stdin?

`Readable` = `...`

stdout?

`Writable` = `...`

style?

`string` = `...`

template?

`string` = `...`

theme?

`string` | `false` | `object` & `object` | (`string` | `object` & `object`)[] = `...`

timeout?

`number` = `...`

title?

`string` = `...`

userStyle?

`string` = `...`

viewer?

`string` = `...`

viewerParam?

`string` = `...`

vite?

`UserConfig` = `...`

viteConfigFile?

`string` | `boolean` = `...`

Returns

`Promise` < `void` >

---

createVitePlugin()

> ***createVitePlugin***(*`inlineConfig`*): *`Promise`* < *`Plugin`* < *`any`* >*[]*>

Parameters

inlineConfig

author?

`string` = `...`

bleed?

`string` = `...`

browser?

`string` = `...`

cmyk?

`boolean` | { `mapOutput?` : `string` ; `overrideMap?` : [`string` | { `b` : `number` ; `g` : `number` ; `r` : `number` ; }, { `c` : `number` ; `k` : `number` ; `m` : `number` ; `y` : `number` ; }][]; `reserveMap?` : [`string` | { `b` : `number` ; `g` : `number` ; `r` : `number` ; }, { `c` : `number` ; `k` : `number` ; `m` : `number` ; `y` : `number` ; }][]; `warnUnmapped?` : `boolean` ; } = `CmykSchema`

config?

`string` = `...`

configData?

`VivliostyleConfigSchema` | `null` = `...`

createConfigFileOnly?

`boolean` = `...`

cropMarks?

`boolean` = `...`

cropOffset?

`string` = `...`

css?

`string` = `...`

cwd?

`string` = `...`

disableServerStartup?

`boolean` = `...`

enableStaticServe?

`boolean` = `...`

enableViewerStartPage?

`boolean` = `...`

executableBrowser?

`string` = `...`

host?

`string` | `boolean` = `...`

ignoreHttpsErrors?

`boolean` = `...`

image?

`string` = `...`

input?

`string` = `...`

installDependencies?

`boolean` = `...`

language?

`string` = `...`

logger?

`LoggerInterface` = `...`

logLevel?

`"info"` | `"silent"` | `"verbose"` | `"debug"` = `...`

openViewer?

`boolean` = `...`

output?

`string` | `object` & `object` | (`string` | `object` & `object`)[] = `...`

port?

`number` = `...`

preflight?

`"press-ready"` | `"press-ready-local"` = `...`

preflightOption?

`string` | `string` [] = `...`

pressReady?

`boolean` = `...`

projectPath?

`string` = `...`

proxyBypass?

`string` = `...`

proxyPass?

`string` = `...`

proxyServer?

`string` = `...`

proxyUser?

`string` = `...`

quick?

`boolean` = `...`

readingProgression?

`"ltr"` | `"rtl"` = `...`

renderMode?

`"local"` | `"docker"` = `...`

sandbox?

`boolean` = `...`

signal?

`AbortSignal` = `...`

singleDoc?

`boolean` = `...`

size?

`string` = `...`

stderr?

`Writable` = `...`

stdin?

`Readable` = `...`

stdout?

`Writable` = `...`

style?

`string` = `...`

template?

`string` = `...`

theme?

`string` | `false` | `object` & `object` | (`string` | `object` & `object`)[] = `...`

timeout?

`number` = `...`

title?

`string` = `...`

userStyle?

`string` = `...`

viewer?

`string` = `...`

viewerParam?

`string` = `...`

vite?

`UserConfig` = `...`

viteConfigFile?

`string` | `boolean` = `...`

Returns

`Promise` < `Plugin` < `any` >[]>

---

defineConfig()

> ***defineConfig***(`config`): `VivliostyleConfigSchema`

Define the configuration for Vivliostyle CLI.

Parameters

config

`VivliostyleConfigSchema`

Returns

`VivliostyleConfigSchema`

---

preview()

> ***preview***(*`options`*): *`Promise`* < *`ViteDevServer`* >

Open a browser for previewing the publication.

Parameters

options

author?

`string` = `...`

bleed?

`string` = `...`

browser?

`string` = `...`

cmyk?

`boolean` | { `mapOutput?` : `string` ; `overrideMap?` : [`string` | { `b` : `number` ; `g` : `number` ; `r` : `number` ; }, { `c` : `number` ; `k` : `number` ; `m` : `number` ; `y` : `number` ; }][]; `reserveMap?` : [`string` | { `b` : `number` ; `g` : `number` ; `r` : `number` ; }, { `c` : `number` ; `k` : `number` ; `m` : `number` ; `y` : `number` ; }][]; `warnUnmapped?` : `boolean` ; } = `CmykSchema`

config?

`string` = `...`

configData?

`VivliostyleConfigSchema` | `null` = `...`

createConfigFileOnly?

`boolean` = `...`

cropMarks?

`boolean` = `...`

cropOffset?

`string` = `...`

css?

`string` = `...`

cwd?

`string` = `...`

disableServerStartup?

`boolean` = `...`

enableStaticServe?

`boolean` = `...`

enableViewerStartPage?

`boolean` = `...`

executableBrowser?

`string` = `...`

host?

`string` | `boolean` = `...`

ignoreHttpsErrors?

`boolean` = `...`

image?

`string` = `...`

input?

`string` = `...`

installDependencies?

`boolean` = `...`

language?

`string` = `...`

logger?

`LoggerInterface` = `...`

logLevel?

`"info"` | `"silent"` | `"verbose"` | `"debug"` = `...`

openViewer?

`boolean` = `...`

output?

`string` | `object` & `object` | (`string` | `object` & `object`)[] = `...`

port?

`number` = `...`

preflight?

`"press-ready"` | `"press-ready-local"` = `...`

preflightOption?

`string` | `string` [] = `...`

pressReady?

`boolean` = `...`

projectPath?

`string` = `...`

proxyBypass?

`string` = `...`

proxyPass?

`string` = `...`

proxyServer?

`string` = `...`

proxyUser?

`string` = `...`

quick?

`boolean` = `...`

readingProgression?

`"ltr"` | `"rtl"` = `...`

renderMode?

`"local"` | `"docker"` = `...`

sandbox?

`boolean` = `...`

signal?

`AbortSignal` = `...`

singleDoc?

`boolean` = `...`

size?

`string` = `...`

stderr?

`Writable` = `...`

stdin?

`Readable` = `...`

stdout?

`Writable` = `...`

style?

`string` = `...`

template?

`string` = `...`

theme?

`string` | `false` | `object` & `object` | (`string` | `object` & `object`)[] = `...`

timeout?

`number` = `...`

title?

`string` = `...`

userStyle?

`string` = `...`

viewer?

`string` = `...`

viewerParam?

`string` = `...`

vite?

`UserConfig` = `...`

viteConfigFile?

`string` | `boolean` = `...`

Returns

`Promise`<`ViteDevServer`>

VFM()

> ***VFM***(*`options?`*, *`metadata?`*): *`Processor`*

Create Unified processor for Markdown AST and Hypertext AST.

Parameters

options?

`StringifyMarkdownOptions`

Options.

metadata?

`Metadata`

Returns

`Processor`

Unified processor.

Interfaces

StringifyMarkdownOptions

Option for convert Markdown to a stringify (HTML).

Properties

Property	Type	Description
`assignIdToFigcaption?`	`boolean`	Assign ID to figcaption instead of img/code.
`disableFormatHtml?`	`boolean`	Disable automatic HTML format.
`footnote?`	`FootnoteMode | { mode: "pandoc"; } | { body?: Properties | DpubBodyFactory; call?: Properties | DpubCallFactory; mode: "dpub"; } | { body?: Properties | GcpmBodyFactory; duplicatedCall?: Properties | GcpmDuplicatedCallFactory; mode: "gcpm"; }`	Footnote output mode. Default is `'pandoc'` (endnote section).
`hardLineBreaks?`	`boolean`	Add ` ` at the position of hard line breaks, without needing spaces.
`imgFigcaptionOrder?`	`"img-figcaption" | "figcaption-img"`	Order of img and figcaption elements in figure.
`language?`	`string`	Document language (ignored in partial mode).
`math?`	`boolean`	Enable math syntax.
`partial?`	`boolean`	Output markdown fragments.
`replace?`	`ReplaceRule[]`	Replacement handler for HTML string.
`style?`	`string | string[]`	Custom stylesheet path/URL.
`title?`	`string`	Document title (ignored in partial mode).

TemplateVariable

Extends

- `Omit`<`ParsedVivliostyleInlineConfig`, `"theme"`>

Properties

Property	Type
`author`	`string`
`bleed?`	`string`
`browser?`	`object`
`browser.tag?`	`string`
`browser.type`	`"chrome"` | `"chromium"` | `"firefox"`
`cliVersion`	`string`
`cmyk?`	`boolean` | { `mapOutput?`: `string`; `overrideMap?`: [`string` | { `b`: `number`; `g`: `number`; `r`: `number`; }, { `c`: `number`; `k`: `number`; `m`: `number`; `y`: `number`; }][]; `reserveMap?`: [`string` | { `b`: `number`; `g`: `number`; `r`: `number`; }, { `c`: `number`; `k`: `number`; `m`: `number`; `y`: `number`; }][]; `warnUnmapped?`: `boolean`; }
`config?`	`string`
`configData?`	`VivliostyleConfigSchema` | `null`
`coreVersion`	`string`
`createConfigFileOnly?`	`boolean`
`cropMarks?`	`boolean`
`cropOffset?`	`string`
`css?`	`string`
`cwd?`	`string`
`disableServerStartup?`	`boolean`
`enableStaticServe?`	`boolean`
`enableViewerStartPage?`	`boolean`
`executableBrowser?`	`string`
`host?`	`string | boolean`
`ignoreHttpsErrors?`	`boolean`

Property	Type
image?	string
input?	object
input.entry	string
input.format	InputFormat
installDependencies?	boolean
language	string
logger?	LoggerInterface
logLevel?	"info" | "silent" | "verbose" | "debug"
openViewer?	boolean
output?	object & object & object[]
port?	number
preflight?	"press-ready" | "press-ready-local"
preflightOption?	string[]
pressReady?	boolean
projectPath	string
proxyBypass?	string
proxyPass?	string
proxyServer?	string
proxyUser?	string
quick?	boolean
readingProgression?	"ltr" | "rtl"
renderMode?	"local" | "docker"
sandbox?	boolean

Property	Type
`signal?`	`AbortSignal`
`singleDoc?`	`boolean`
`size?`	`string`
`stderr?`	`Writable`
`stdin?`	`Readable`
`stdout?`	`Writable`
`style?`	`string`
`template?`	`string`
`theme?`	`string` | `object` & `object` |(`string` | `object` & `object`)[]
`themePackage?`	`VivliostylePackageJson`
`timeout?`	`number`
`title`	`string`
`userStyle?`	`string`
`viewer?`	`string`
`viewerParam?`	`string`
`vite?`	`UserConfig`
`viteConfigFile?`	`string` | `boolean`

Type Aliases

Metadata

> ***Metadata*** = *`object`*

Metadata from Frontmatter.

Properties

base?

> `optional` **base**: `Attribute[]`

Attributes of `<base>`.

body?

> `optional` **body**: `Attribute[]`

Attributes of `<body>`.

class?

> `optional` **class**: `string`

Value of `<html class="...">`.

custom?

> `optional` **custom**: `object`

A set of key-value pairs that are specified in `readMetadata` not to be processed as `<meta>`. The data types converted from Frontmatter's YAML are retained. Use this if want to add custom metadata with a third party tool.

Index Signature

[`key` : `string`]: `any`

dir?

> `optional` **dir**: `string`

Value of `<html dir="...">`. e.g. `ltr`, `rtl`, `auto`.

head?

> `optional` ***head***: `string`

`<head>...</head>`, reserved for future use.

html?

> `optional` ***html***: `Attribute` *[]*

Attributes of `<html>`. The `id`, `lang`, `dir`, and `class` specified in the root take precedence over the value of this property.

id?

> `optional` ***id***: `string`

Value of `<html id="...">`.

lang?

> `optional` ***lang***: `string`

Value of `<html lang="...">`.

link?

> `optional` ***link***: `Attribute` *[][]*

Attribute collection of `<link>`.

meta?

> `optional` ***meta***: `Attribute` *[][]*

Attribute collection of `<meta>`.

script?

> `optional` ***script***: `Attribute` *[][]*

Attribute collection of `<script>`.

style?

> `optional` ***style***: `string`

`<style>...</style>`, reserved for future use.

title?

> `optional` ***title***: `string`

Value of `<title>...</title>`.

vfm?

> `optional` ***vfm***: `VFMSettings`

VFM settings.

StructuredDocument

> ***StructuredDocument*** = `object`

See

https://github.com/vivliostyle/vivliostyle-cli/blob/main/docs/config.md (https://github.com/vivliostyle/vivliostyle-cli/blob/main/docs/config.md)

Properties

children

> ***children***: `StructuredDocument` *[]*

href

> ***href***: `string`

sections?

> `optional` ***sections***: `StructuredDocumentSection` *[]*

title

> ***title***: `string`

StructuredDocumentSection

> ***StructuredDocumentSection*** = `object`

See

https://github.com/vivliostyle/vivliostyle-cli/blob/main/docs/config.md (https://github.com/vivliostyle/vivliostyle-cli/blob/main/docs/config.md)

Properties

children

> ***children***: `StructuredDocumentSection` *[]*

headingHtml

> ***headingHtml***: `string`

headingText

> ***headingText***: `string`

href?

> `optional` ***href***: `string`

id?

> `optional` ***id***: `string`

level

> ***level***: `number`

VivliostyleConfigSchema

> ***VivliostyleConfigSchema*** = `BuildTask` *[]* | `BuildTask`

See

https://github.com/vivliostyle/vivliostyle-cli/blob/main/docs/config.md (https://github.com/vivliostyle/vivliostyle-cli/blob/main/docs/config.md)

VivliostylePackageMetadata

> ***VivliostylePackageMetadata*** = `SchemaWithPipe`<readonly [`Omit`< `ObjectSchema`<{ `template`: `SchemaWithPipe`<readonly [`RecordSchema`< `SchemaWithPipe`<readonly [..., ..., ...]>, `ObjectSchema`<{ `description`: ...; `name`: ...; `prompt`: ...; `source`: ...; }, `undefined`>, `undefined`>, `TitleAction`<{[`key`: `string`]: `object`; }, `"VivliostyleTemplateMetadata"`>]>; `theme`: `SchemaWithPipe`<readonly [`ObjectSchema`<{ `author`: `SchemaWithPipe`<...>; `category`: `SchemaWithPipe`<...>; `name`: `SchemaWithPipe`<...>; `style`: `SchemaWithPipe`<...>; `topics`: `SchemaWithPipe`<...>; }, `undefined`>, `TitleAction`<{ `author?`: ... | ...; `category?`: ... | ...; `name?`: ... | ...; `style?`: ... | ...; `topics?`: ... | ...; }, `"VivliostyleThemeMetadata"`>]>; }, `undefined`>, `"entries"` | `"~types"` | `"~run"` | `"~standard"`> & `object`, `TitleAction`<{ `template?`: {[`key`: `string`]: `object`; }; `theme?`: { `author?`: `string`; `category?`: `string`; `name?`: `string`; `style?`: `string`; `topics?`: `string`[]; }; }, `"VivliostylePackageMetadata"`>]>

VivliostylePackageMetadata

> ***VivliostylePackageMetadata*** = `v.InferInput`<typeof `VivliostylePackageMetadata`>

Variables

readMetadata()

> `const` ***readMetadata***: (`md`, `customKeys?`) => `Metadata`

Read metadata from Markdown frontmatter.

Keys that are not defined as VFM are treated as `meta`. If you specify a key name in `customKeys`, the key and its data type will be preserved and stored in `custom` instead of `meta`.

Parameters

md

`string`

Markdown.

customKeys?

`string`[]

A collection of key names to be ignored by meta processing.

Returns

`Metadata`

Metadata.