Vivliostyle CLI Documentation

Table of Contents

Vivliostyle CLI Documentation

Vivliostyle CLI Documentation

Getting Started

Vivliostyle CLI is a command-line interface for typesetting HTML and Markdown documents. It includes the 1. https://docs.vivliostyle.org/#/vivliostyle-viewerVivliostyle Viewer[*1] 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* 2. https://nodejs.org/*Node.js*[*2] *(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 3. https://github.com/unjs/giget#readmegiget[*3] documentation.

Using Config File

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 1. https://code.visualstudio.com/docs/languages/json#_json-with-comments JSONC[*1] (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.

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.

Themes and CSS

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 1. https://developer.mozilla.org/en-US/docs/Web/CSS/Cascade#author_stylesheetsauthor stylesheet[*1] 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 2. https://developer.mozilla.org/en-US/docs/Web/CSS/Cascade#user_stylesheets user stylesheet[*2], 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.

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
<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.

Creating Table of Contents Page

Creating Table of Contents Page

- 1. https://github.com/vivliostyle/vivliostyle-cli/tree/main/examples/table-of-contents
  Example: table-of-contents*1

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).

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 1. https://github.com/vivliostyle/themes/tree/main/packages/%40vivliostyle/theme-epub3j"@vivliostyle/theme-epub3j"[*1] compliant with the 2. https://dpfj.or.jp/counsel/guideEBPAJ EPUB 3 File Creation Guide[*2].

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 3. https://www.w3.org/TR/pub-manifest/Publication Manifest[*3].

Frontend Framework Support

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.*

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()