Haita Documentation

Make Web documentation w/o external tools in pure Typst

Jeremy Gao

2026-08-07

Contents

1. Introduction 2
   - 1.1. An Unfinished Project 3
   - 1.2. Licensing 3
2. Installation 4
   - 2.1. Installing Typst 4
3. Tutorial 5
   - 3.1. Writing Your First Document 5
   - 3.2. Architecture 7
4. Authoring 8
   - 4.1. Cross-references 8
   - 4.2. Contextual Output And Embedding HTML 8
   - 4.3. Copyable Math Blocks 9
5. Using Other Typst Packages 10
   - 5.1. Package Support 10
   - 5.2. Integration with Tidy 10
6. Integrating Tools and Assets 11
   - 6.1. Reading and Processing Files 11
   - 6.2. Integrating Pagefind 11
   - 6.3. Embedding JavaScript 12
7. Custom Renderer 13
8. Deploying Your Site 14
   - 8.1. GitHub Actions and Pages 14
9. References 16
   - haita 16
   - new-hamber 20
10. Demo 24
    - 10.1. Typography 24
    - 10.2. Math 25
    - 10.3. Lists 26
    - 10.4. Prime Number Table 27
    - 10.5. CJK 27
    - 10.6. CJK Vertical Layout 27
11. Code Demo 28
    - 11.1. Rust 28
    - 11.2. C++ 28
    - 11.3. Python 28
    - 11.4. Typst Math 28
    - 11.5. Typst Markup 29
12. Changelog 31
    - 0.4.0-rc1 (Aug. 3, 2026) 31
    - 0.3.0 (Jul. 21, 2026) 31
    - 0.3.0-rc1 (Jul. 19. 2026) 31
    - 0.2.1 (Jul. 10, 2026) 32
    - 0.2.0 (Jul. 6, 2026) 32
    - 0.1.0 (Jul. 3, 2026) 32

Chapter 1

Introduction

Welcome to the documentation for Haita 0.3.0 (hǎi tǎ, Mandarin Pinyin, lit. Sea Otter).

Web Ready! **Info 1**

The documentation is also available as HTML.

It's not ready yet! **Warning 2**

Haita is a decent choice for organizing long, comprehensive documentation. But just like Typst, Haita is an unfinished project, and is (currently) not a serious tool.

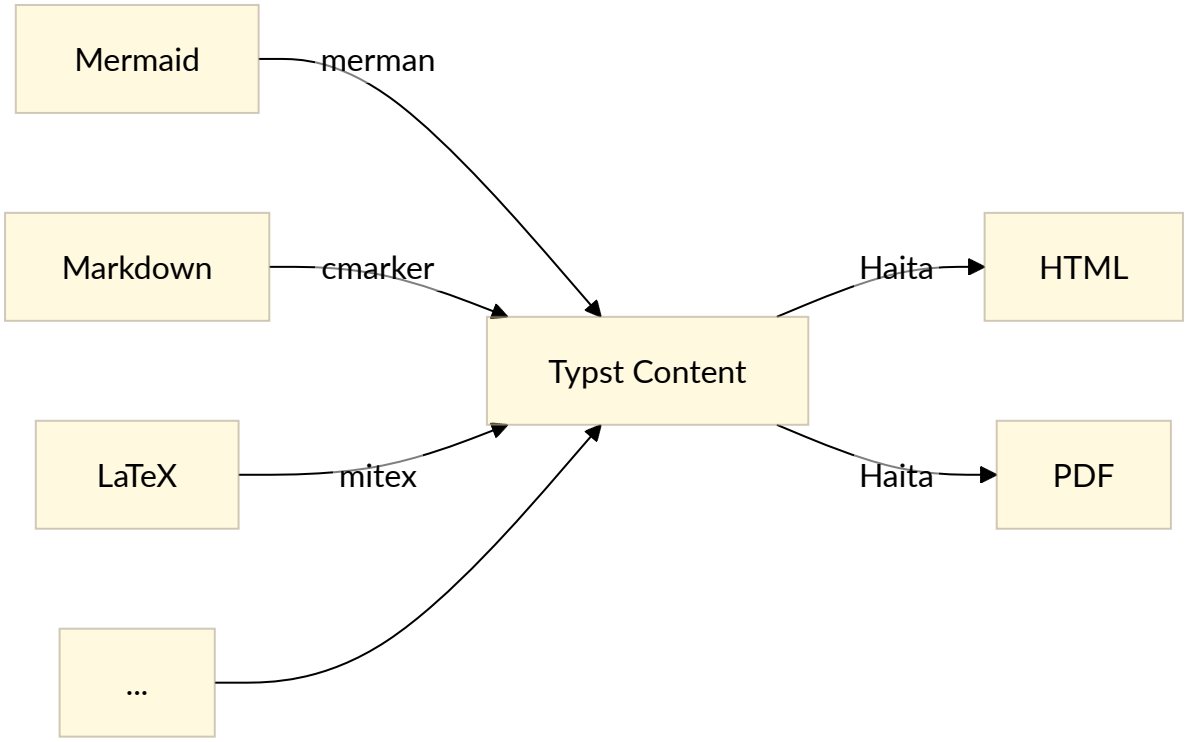

Figure 1: How Haita Works

Writing documentation is a lame task. It is even more boring and frustrating when you have to setup toolchains and environments and debug for hours to make sure that they build correctly, only to find that the current tools cannot plot your diagrams, or the PDF generation is missing fonts and takes hours to build. So here's Haita. **A simple tool that has a single requirement: Typst**. Here are some features:

- Pure Typst workflow
- Features inherited from Typst:
  - Simple yet expressive Typst syntax helping you focussing on your content

  - Native syntax highlighting
  - Native MathML output
  - Fast compliation
  - Native support for `watch` and `serve`
  - PDF and HTML generation from the same source[1]
  - HTML minification.
- Minimal client side JS by default (for copying code). No JS required for math blocks. Site fully usable and navigatable without JS.
- Good SEO, including generating preview images for links.
- Semantic output, and
- Minimal setup

$$W = \frac{\Psi'^2 ab}{2\mu_0}\left[\mu^2 + \sum_{m,n}\left(\frac{a_{mn}^2}{4}\left(\frac{m^2n^2}{a^2} + \frac{n^2\pi^2}{b^2} + \mu^2\right) + \frac{8a_{mn}\mu^2}{\pi^2 mn}\right)\right]$$

Listing 1: A math formula example (source). Hover on the equation to reveal the copy button!

$$f(x) = \int_{\infty}^{\infty} \hat{f}(\xi) e^{2\pi i \xi x} \,\mathrm{d}\xi$$

Listing 2: See how it renders with Typst + MathML (source)

The default theme, *New Hamber*, also supports dark mode.

You can make a new project in Typst using Haita, set it to bundle export, and Haita would generate a site for you. You don't need to worry about setting up the toolchain – Typst is the only tool required.

1.1. An Unfinished Project

Haita is still missing some features, specifically:

- Internationalization support
- Built-in search functions

> **Pagefind Integration** **Tip 3**
>
> You can integrate Pagefind manually. See this page [6.2.] for details.

However, if you want pure Typst documentation, ease of use, and/or MathML formulae, you might want to give it a try. If you want stability and extremely easy syntax, then maybe you should consider mdBook. If you have any issues, please feel free to open a ticket on GitHub. If you would like to contribute, please open a pull request.

1.2. Licensing

The source and the documentation are available under Apache License v2.0.

> **Tracking** **Info 4**
>
> This site uses Umami to track visitor status. It stores no cookies and does not collect personal data. All data collected are anonymous, and you can disable tracking for this site following this guide.

[1] PDF generation only works when using `--foramt pdf` and does not work with `--format` bundle. See https://github.com/typst/typst/issues/8309 for details.

Chapter 2

Installation

Installing Haita's dependencies is incredibly simple! You only need the Typst compiler. Typst will automatically fetch the required packages when compiling the documents.

2.1. Installing Typst

You can install Typst at https://typst.app/open-source/#download.

After you installed Typst, move to the next section [2.1.] to get started using Haita.

Chapter 3

Tutorial

Haita uses Typst. If you are not familiar with Typst, you can first take a look at Typst's official tutorial for info on how to write Typst.

3.1. Writing Your First Document

If you've used a tool like Shiroa or mdBook, you might be familiar with `book.typ` or `summary.md`. Both files contain metadata and instructions on how to organize the book. In Haita, all of that is concentrated to a single entrypoint – the `book` function. To get started, create a file named `dist.typ` that contains the following:

```
#!/usr/bin/env -S typst compile --features bundle,html --format bundle
// The line above compiles the documentation to an HTML bundle.
// Additionally, you can watch the file using this command:
// ```
// typst watch --features bundle,html --format bundle main.typ
// ```
// You can also build and watch the PDF using the follow commands:
// ```
// typst compile --features bundle,html --format pdf main.typ
// typst watch --features bundle,html --format pdf main.typ
// ```
#import "@preview/haita:0.3.0": * // Always import the package!
#book(
  // Where the site will be deployed. Optional: it is only used for the
  // SEO metadata, everything inside the site is linked relatively.
  // base-url: "https://username.github.io/haita",

  // This sets your html renderer. You can customize the HTML renderer
  // using `html-renderer.with(...)`, or write your own!
  html-renderer: new-hamber.html-renderer,
  // Your document's contents
  tree: (
    // You can add arbitrary content. The content will be displayed
    // in the summary, but will not generate html pages.
    [= Welcome!],
    // This will create index.html. The content of the
    // chapter will be from `index.typ`
    chapter("index", content: include "index.typ"),
    // This will create doc/tutorial.html. In this case,
    // the content of the chapter is not explicitly stated, so it
    // looks into ./doc/tutorial.typ in the current workspace.
    chapter(
      "doc/tutorial",
      content: include "tutorial.typ",
      // you can generate chapters procedurally
      children: range(1, 6).map(num => chapter("doc/" + str(num), content: [
        #title[Chapter #num]
        This page is generated procedurally!
```

```
39        ])),
40      ),
41      // You can add dividers, which will separate content in the summary.
42      divider(),
43      // Alternatively, if you would like to directly include the content
44      // without creating a new file, you can write it like this:
45      chapter("my-page", content: [
46        #title[My Page]
47        = Heading 1
48        = Heading 2
49        foo bar baz
50      ]),
51      // you can also add arbitrary content
52      [Made with Haita.],
53      // you can add more chapters afterwards.
54    ),
55  )
56
```

Each chapter should start with a `title`. The title of the page will also be displayed in the summary. Do not start your document with a level 1 heading that looks like this: `= Heading`. Instead, write this:

```
1  #title[My amazing document]
2
3  = Heading 1
4  Overseas from coast to coast
5
6  = Heading 2
7  To find a place I love the most
```

The rest of your document is just normal Typst content. You could apply show rules or use functions, just like writing a normal Typst document.

3.1.1. Developing (Typst Web App)

The `typst.app` web app currently does not support the bundle target and MathML exports. Bundle export would not work in the Web App.

https://typst.app is an online Typst editor developed by the Typst team.

3.1.2. Developing (Local machine)

3.1.2.1. Bundle (HTML) Export

Open a terminal on your device. In the terminal, type the following command, then press `return`:

```
1  typst compile --features bundle,html --format bundle dist.typ
```

This will generate a folder `dist/` inside the same folder where you put your `dist.typ` file. The content inside `dist/` is your document. You can see more available options by executing `typst help`.

The `typst compile` command only generates `dist/` once. Any subsequent edits will not appear inside `dist/`. Use the following command to let Typst monitor your changes:

```
typst watch --features bundle,html --format bundle dist.typ
```

This will also generate a folder `dist/` alongside `dist.typ`, but this time there is some extra information:

```
watching ./dist.typ
writing to ./dist
serving at http://127.0.0.1:3000         <-- notice this line!

[HH:MM:SS] compiled with warnings in <some-time>

warning: bundle export is experimental
  = hint: its behaviour may change at any time
  = hint: do not rely on this feature for production use cases
```

This tells you that Typst is actively monitoring your files, and any changes you make to the files will let Typst recompile the project. This line `serving at http://127.0.0.1:3000` (3000 might be a different number) tells you that Typst has opened a local development server that this specific address. You can copy `http://127.0.0.1:3000`, open a browser, and paste that line in the address bar.

You will see your document's index page in your browser.

3.1.2.2. PDF Export

You can export your entire document as a PDF using the following command:

```
typst compile --features bundle,html --format pdf dist.typ
```

This would generate a file named `dist.pdf` which you can open in your PDF viewer. Similarly, you can also watch the PDF output:

```
typst watch --features bundle,html --format pdf dist.typ
```

Instead of opening a local development server, Typst writes into the PDF file every time you modify the `.typ` source.

`typst watch` is very fast for both PDF and HTML output. This is because Typst uses incremental compilation for recompilations.

3.2. Architecture

The documentation system has two parts: the organizer, which is the `book` function [9], and the renderer.

The `book` function will organize the content into a tree, and that tree is then fed into the renderer. The renderer will take a tree, inspect it, then generate pages based on that tree. This means that writing your own renderer is trivial. You can swap to a different renderer at any time.

Chapter 4

Authoring

This section will cover specific authoring details when using Haita. Please refer to https://typst.app/docs/tutorial/ and https://typst.app/docs/reference/syntax/ for Typst reference.

4.1. Cross-references

You can reference different content using the following syntax:

```
// This attaches a `label` to the figure
#figure(..) <my-listing>

See @my-listing for an example.
See #link(<my-listing>)[This listing]
```

Listing 3: Reference syntax

In addition, you can reference different pages using `label("page:/path/to/page")`. See https://typst.app/docs/reference/foundations/label/ for additional information on labels. You can write the following to create a link to a different page:

```
#link(label("page:/path/to/page"))[See this page for help]
```

You can even create a reference to an HTML element:

```
#context if target() == "html" [
  #html.section[Foo bar baz] <foobar>
]
// somewhere else in the document
#context if target() == "html" [
  #link(<foobar>)[This section] explains.
]
```

You may reference a label located in one page from the same page or from a different page.

4.2. Contextual Output And Embedding HTML

You can write different content for different targets, for example, embedding a link to a page via `iframe` in the HTML target. In addition, Typst provides a typed interface for HTML elements. Custom HTML elements may be embedded using `html.elem()`

```
#context if target() == "html" {
  // target IS HTML. Write some HTML specific stuff
  html.elem(..)
  html.iframe(..)
  // or import the entire html interface
  import html: *
  div(..)
} else {
```

```
 9    // target IS NOT HTML. It is instead "paged", which is for PDF, PNG and SVG.
10    // Write some PDF specific stuff
11    curve(..)
12  }
```

Please always keep HTML specific content in the `context if target() == "html" {..}` block. HTML elems will not work when using PDF export.

You are currently looking at the PDF target.

4.3. Copyable Math Blocks

Haita provides an extension to standard Typst to make math blocks copyable as Typst source. This extension only works for math blocks, and does not work for inline math such as $a^2 + b^2 = c^2$. Typically, math blocks are written as follows:

```
1  $
2    f(x) = integral^oo_oo hat(f)(xi)e^(2pi i xi x) dif xi
3  $
```

This will be rendered as follows. Notice how the math block's source cannot be copied:

$$f(x) = \int_{\infty}^{\infty} \hat{f}(\xi) e^{2\pi i \xi x} \,\mathrm{d}\xi$$

Instead, you can remove the `$ $` and wrap your text in the `typm-copy` code block. Haita will extract the text from the code block, evaluate it, and render a math block with a copy button. Additionally, you can also hover on the math block to reveal a tooltip for the Typst source:

````
1  ```typm-copy
2  f(x) = integral^oo_oo hat(f)(xi)e^(2pi i xi x) dif xi
3  ```
````

$$f(x) = \int_{\infty}^{\infty} \hat{f}(\xi) e^{2\pi i \xi x} \,\mathrm{d}\xi$$

Note that writing inside raw blocks won't give you LSP completions. In this case, you can first write the math formula in a standard `$ $`, then remove the dollar sign and wrap it in the code block. You may not reference any outside variables when writing in this way. For example, the following snippet would not work at all since the math block relies on a method that is not defined in Typst's standard library:

````
1  #let my-method(foo) = math.attach(foo, tr: foo)
2  // this works
3  $
4    #my-method[8848.86]
5  $
6  // but this is borked
7  // ```typm-copy
8  // #my-method[8848.86]
9  // ```
````

$$8848.86^{8848.86}$$

Chapter 5

Using Other Typst Packages

You can add any Typst packages into your document using the package import syntax:

```
#import "@preview/my-package:version": foo, bar
```

Many packages are available in the Typst Universe. Go check them out!

5.1. Package Support

Packages have varying levels of HTML support. If you generally use Haita for the PDF export, this won't be a major problem. If you use Haita primarily for HTML export, check if your package supports HTML.

Additionally, if you draw diagrams with Typst, it is a good idea to wrap your diagram in `html.frame()`. In HTML export, it turns the content into an SVG illustration that gets embedded into your HTML page; in PDF export, this emits regular PDF content.

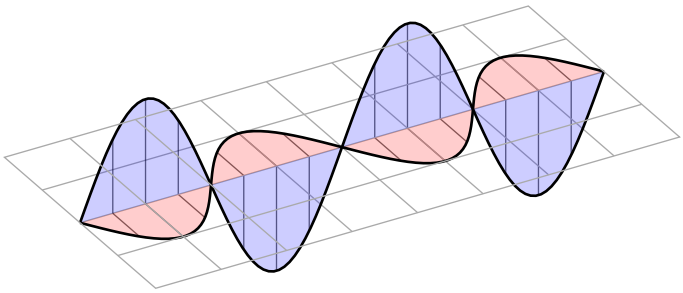

Figure 2: A CeTZ illustration

```
#let cetz-illustration = { /* Draw your content here */ }
#figure(caption: [Your Caption], html.frame(cetz-illustration))
```

5.2. Integration with Tidy

Tidy is a Typst documentation tool. Haita's default theme, *New Hamber*, provides built-in integration for rendering Tidy modules.

Chapter 6

Integrating Tools and Assets

Haita doesn't have macros since Typst doesn't have macros. It is hard to achieve mdBook style preprocessors. However, there are a few ways you can integrate foreign tools and assets in your documentation.

There are a couple of scenarios where you might want to integrate other tools and assets, for example, showing a copy of a file in the docs, displaying an interactive editor, or demonstrating the result of a program. Most of them are doable with Typst's capabilities, and some require specific workarounds.

6.1. Reading and Processing Files

Typst provides native syntax for reading files. Specifically, there are these functions and keywords:

`#read(<path>, encoding: <encoding>)`	Read a file's content using the encoding specified. The `<encoding>` defaults to UTF-8, and the `read` function would return a string when the encoding is specified. Use `encoding: none` for reading and returning raw bytes.
`#include "<filename>"`	Include and evaluate the content of a Typst `.typ` file. Unlike C/C++'s `#include`, this is scoped.
`#raw(block: <boolean>, lang: <code-language>, <content>)`	Display a string as code block. Use `block: true` for block-level code, and specify `<code-language>` in the lang field. (e.g. `"C++"` for C++, and `"rust"` for Rust.)
`#eval(mode: <evaluation-mode>, <content>)`	Evaluate the `<content>` as Typst code using the `<evaluation-mode>` specified.
`#cbor(..)`	Functions for reading and processing data files. Those functions would return structured data. You can use the structured data and spread them into a table, evaluate them, or plot them. See https://typst.app/docs/reference/data-loading/ for details.
`#csv(..)`	
`#json(..)`	
`#toml(..)`	
`#xml(..)`	
`#yaml(..)`	

A few examples include `#table(columns: 3, ..csv("my_file.csv").flatten())` for reading records and displaying them and `#raw(block: true, lang: "rust", read("my_file.rs"))` for reading content from a Rust file in your project's source code.

6.2. Integrating Pagefind

Pagefind is a popular, fully static search library that is extremely convenient to use. It works with any static HTML output, hence it works with Haita. It adds a bit of JavaScript overhead.

The default theme *New Hamber* provides built-in support for Pagefind.

- **Download Pagefind** follow the steps in https://pagefind.app/docs/installation/#downloading-a-precompiled-binary or check if Pagefind is available in your package manager.

  After installing, you should be able to run Pagefind in the command line using the command `pagefind`.

- **Enable *New Hamber* integration** Enable the `pagefind-enabled` setting in the html renderer as follows:

```
#import "@preview/haita:...": book, new-hamber
#book(
  html-renderer: new-hamber.html-renderer.with(
    pagefind-enabled: true,
  ),
  .. // your regular content
)
```

- **Running the Pagefind program** After compiling your documentation, you need to run `pagefind` on your generated files. Suppose you have a `dist.typ` file and your output is in `./dist`, you'd need to run this command:

```
pagefind --site ./dist --output-subdir pagefind
```

After you finished all of that, you should be able to see a pagefind searchbox in your main table of contents at the left hand side.

6.3. Embedding JavaScript

JS embedding would be demonstrated in the Web version. PDFs do support JavaScript; however, the support is extremely limited, and many PDF viewers would omit the JavaScript inside PDFs.

Chapter 7

Custom Renderer

Haita is separated into two parts: the collector and the renderer. The collector organizes the data and sends it to the renderer, which is responsible for generating the HTML and PDF pages, PNG images, CSS stylesheets, and other miscellaneous files. In this section, you will learn how to write a custom renderer.

WIP

Chapter 8

Deploying Your Site

If you want to make others see your site, you'll have to make it available somewhere! You'll need to do these things:

1. Setting up the workflow
2. Building the site
3. Deploying the site

8.1. GitHub Actions and Pages

Use the following CI file:

```yaml
# The name of your workflow. Always specify the name!
name: Build Docs

# Events that would trigger your workflow
on:
  # In this case we would like to build the documentation
  # whenever new commits are pushed to the `main` branch
  push:
    branches: [main]
  # Or, if you want to manually trigger the workflow,
  # use `workflow_dispatch`
  workflow_dispatch:

# necessary permissions required for deploying the document
permissions:
  contents: read
  pages: write
  id-token: write

# What jobs would be involved in building the document
jobs:
  # Job 1: build the docs
  build-docs:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v5
      - uses: DeterminateSystems/nix-installer-action@v20
      - uses: DeterminateSystems/magic-nix-cache-action@v13

      - name: Build docs
        run: nix develop .#default --command time just full-docs

      - name: Upload Pages artifact
        uses: actions/upload-pages-artifact@v3
        with:
          path: ./dist
  # Job 2: deploy the docs
  deploy:
```

```
39      needs: build-docs
40      runs-on: ubuntu-latest
41      environment:
42        name: github-pages
43      steps:
44        - uses: actions/deploy-pages@v5
45
```

Chapter 9

References

References for functions and methods in Haita.

haita

- book() [9.]
- chapter() [9.]
- minimal-summary-image-renderer() [9.]

book

The entrypoint of the entire documentation.

Parameters

```
book(
    title [9.] : str,
    description [9.] : str,
    base-url [9.] : none str,
    authors [9.] : array,
    lang [9.] : str,
    html-renderer [9.] : function,
    paged-renderer [9.] : function,
    debug [9.] : bool,
    tree [9.] : array
) -> content
```

title `str`

The title of the documentation.

Default: `""`

description `str`

The description of the documentation.

Default: `""`

base-url `none` or `str`

The URL the documentation is deployed at, including the subfolder when you are not deploying to the root of a site. Examples include `https://<username>.github.io/<project name>` for GitHub Pages and `https://<project name>.pages.dev` for Cloudflare Pages.

This setting is optional: it is only used to build the absolute URLs that SEO metadata requires (the canonical link, Open Graph and Twitter cards).

Default: `none`

authors `array`

The authors of the documentation. It should be an array of strings.

Default: `()`

lang `str`

The language of the documentation

Default: `"en"`

html-renderer `function`

Which HTML renderer to use. By default it uses *New Hamber*'s html renderer.

Default: `new-hamber.html-renderer`

paged-renderer `function`

Which paged (PDF, PNG, SVG) renderer to use. By default it uses *New Hamber*'s paged renderer.

Default: `new-hamber.paged-renderer`

debug `bool`

Whether to enable the debug mode or not. Debug mode outputs a tree of the current document at `/__debug_tree.html`

Default: `false`

tree `array`

The content of your documentation.

Default: `()`

chapter

This function defines a chapter in the output

```
1  #chapter("doc/lib.typ")
```

Parameters

```
chapter(
    path 9. : str,
    content 9. : auto content,
    children 9. : any chapter,
    numbered 9. : bool,
    ..args 9. : any
) -> chapter
```

path `str`

The path to the chapter. When `content` is left blank the function would automatically look into the current directory and fetch the content if there is a file at the path ending in `.typ`

content `auto` or `content`

The content of the chapter.

Default: `auto`

children `any` or `chapter`

Subchapters

Default: `()`

numbered `bool`

Whether if this chapter is numbered

Default: `true`

..args `any`

Extra arguments passed to the renderer

minimal-summary-image-renderer

Minimal page summary renderer. Use it like this. Always use it with the `.with` syntax:

```
1  #book(
2    html-renderer: new-hamber.html-renderer.with(
```

```
3      summary-image-renderer: lib.minimal-summary-image-renderer.with(
4        image-content: it => [...] // your content here
5        // Don't complete any other fields
6      )
7    )
8  )
```

Parameters

```
minimal-summary-image-renderer(
    chapter 9. : chapter,
    base-url 9. : none str,
    width-px 9. : int,
    height-px 9. : int,
    ppi 9. : int,
    image-content 9. : function
)
```

chapter `chapter`

The chapter that would be feeded into the renderer. Usually this is internal to the renderer, and should not be completed.

base-url `none` or `str`

The base URL of your site, taken from `book`'s `base-url`. Usually this is internal to the renderer, and should not be completed. When it is `none` the image is referenced relative to the page instead of by an absolute URL.

Default: `none`

width-px `int`

The width of the image in pixels. For the default PPI it is 1pt -> 1px.

Default: `1200`

height-px `int`

The height of the image in pixels. For the default PPI it is 1pt -> 1px.

Default: `630`

ppi `int`

The PPI of the image in pixels. If you change the PPI via the `--ppi` when compiling you must also change this value.

Default: `144`

image-content `function`

The content of the summary image.

Default: `chapter => none`

new-hamber

- html-renderer() [9.]
- paged-renderer() [9.]

html-renderer

New Hamber's HTML renderer.

Parameters

```
html-renderer(
    tree [9.] ,
    lang [9.] ,
    title [9.] ,
    base-url [9.] ,
    summary-image-renderer [9.] ,
    footer-content [9.] : content,
    extra-css [9.] : none str,
    extra-head-content [9.] : none content,
    sidebar-image [9.] : content,
    favicon [9.] : content none,
    pagefind-enabled [9.] : bool,
   ..args
)
```

tree

The content tree

lang

The language of the book

Default: `"en"`

title

The title of the book

Default: `""`

base-url

The base URL

Default: `none`

summary-image-renderer

The summary image renderer

Default: `none`

footer-content `content`

The footer is displayed on each page. Controls the footer's content.

Default:
```
[
   Powered by #link("https://github.com/wensimehrp/haita")[Haita]. Made in Vancouver with love.
  ]
```

extra-css `none` or `str`

Extra content to put into the global stylesheet

Default: `none`

extra-head-content `none` or `content`

Extra content to put into `head`

Default: `none`

sidebar-image `content`

The image of the navigation sidebar

Default:
```
html.a(
         href:  "https://en.wikipedia.org/wiki/File:Sea_Otter_(Enhydra_lutris)_(25169790524)_crop.jpg",
    html.img(
      class: "w-full h-45 object-cover object-top",
        src:  "https://upload.wikimedia.org/wikipedia/commons/0/02/Sea_Otter_%28Enhydra_lutris%29_%2825169790524%29_crop.jpg",
    ),
  )
```

favicon `content` or `none`

The favicon of the site. Currently only SVG is supported.

Default: `asset("favicon.svg", read("assets/favicon.svg"))`

pagefind-enabled `bool`

Whether or not to enable pagefind integration [6.2.]

Default: `false`

paged-renderer

New Hamber's Paged (PDF, PNG, SVG) renderer.

Parameters

```
paged-renderer(
  tree,
   title [9.] : str,
   description [9.] : content,
   authors [9.] : array,
   lang [9.] : str,
   date-format [9.] ,
  ..args
)
```

title `str`

The title of the documentation

Default: `""`

description `content`

A brief explanation about the document

Default: `[]`

authors `array`

The author(s) of the document

Default: `()`

lang `str`

The language of the document.

Default: `"en"`

date-format

The date format used

Default: `auto`

Chapter 10

Demo

10.1. Typography

Tailwind Typography and Typst helps styling `inline code`, coloured inline code: `fn foo<T: Clone>(f: fn(T) -> ())`, **bold**, *italics*, ***bold italics***, highlighted text, underlined text, overlined text, "quoted text", "quote's 'effect' in quoted text", SmallCaps, ~~strikethrough text~~, subscripts, superscripts, UPPERCASE TEXT, **And** a *"comprehensive"* EXAMPLE

10.1.1. Lorem

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et.

10.1.2. Typst on Wikipedia

See also: *original page on wikipedia.org*
Try hovering your cursor on the footnotes![2]

Typst (/ˈtaɪpst/)[3] is an open-source[4] typesetting[5] system and corresponding markup language. The Typst compiler[6] is free[7] software and is distributed under the Apache License 2.0 license.[8]

The system is designed for writing and formatting scientific[I] texts and mathematical[J] formulas. Typst supports simple formatting[K] for common formatting applications, customizable functions, an integrated scripting language, and mathematical typesetting. It is designed to be an alternative to LaTeX.[L]

The compiler is developed by Typst GmbH,[xiii] which maintains[xiv] and supports the software's development, and operates a proprietary[xv] collaborative[xvi] cloud-based[xvii] editor, offering both

[2] You can put arbitrary content in the footnote. Like this: $\log_a u = \frac{\log_b u}{\log_b a}$.
[3] The stuff we're using
[4] Meaning that the software's source is available to everyone
[5] Basically meaning putting a bunch of text together
[6] A translator in the computer science sense.
[7] Free as in freedom, not as in free lunch
[8] Very cool license
[I] Science related
[J] Mathematical as in math
[K] See the typography chapter 10.1. for details
[L] Ohh this is the baddie
[xiii] Nice company with nice people
[xiv] Maintain is the act of maintaining
[xv] Have you heard of open source? Maybe yes. Try it out, it's super cool

free and paid services in a manner similar to Overleaf,[xviii] which allows users to preview their work while writing and includes a collaboration feature.

10.1.2.1. History

Typst has been developed since 2019 and was first published in 2022 by Laurenz Mädje and Martin Haug for their masters theses at Technische Universität Berlin. In March 2023, the Typst compiler was released as open-source, and the beta version of its web app was simultaneously announced. As of 2025, the web app is out of beta.

According to GitHub, Typst was the second fastest-growing programming language by percentage in 2025.

10.1.3. Headings

Heading 1

Heading 2

Heading 3

Heading 4

Note that these headings are not outlined, so they would not appear on the sidebar.

Framed text:

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do.

10.2. Math

Inline: the root formula is $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ and the Chudnovsky algorithm is $\frac{1}{\pi} = \frac{\sqrt{10005}}{4270934400} \sum_{k=0}^{\infty} \frac{(-1)^{k(6k)!}(545140134k+13591409)}{(3k)!(k!)^3(640320)^{3k}}$ with Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri. and $A = \int_a^b (f(x) - g(x))\,\mathrm{d}x$ Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do.

$$\frac{1}{\pi} = \frac{\sqrt{10005}}{4270934400} \sum_{k=0}^{\infty} \frac{(-1)^{k(6k)!}(545140134k + 13591409)}{(3k)!(k!)^3(640320)^{3k}}$$

Figure 3: The Chudnovsky algorithm

$$\mu = \frac{1}{n} \sum^{n} i = 1x_i$$

Figure 4: Something to do with math

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

Figure 5: Something to do with math

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Figure 6: Something to do with math

[xvi] Lots of people can edit together simultaneously

[xvii] No need to store files locally

[xviii] Baddie

$$\lim_{n \to \infty} \left(1 + \frac{1}{n}\right)^n = e$$

Figure 7: Something to do with math

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$$

Figure 8: Something to do with math

$$x + y^{\frac{2}{k+1}}$$

Figure 9: Something to do with math

$$x_{y_b^a}^{z_c^d}$$

Figure 10: Something to do with math

$$\det(A) = \sum_{\sigma \in S_n} \varepsilon(\sigma) \prod_{i=1}^{n} a_{i,\sigma_i}$$

Figure 11: Something to do with math

$$\left\{ \underbrace{\overbrace{a, ..., a}^{k a\text{'s}}, \overbrace{b, ..., b}^{l b\text{'s}}}_{k+l \text{ elements}} \right\}$$

Figure 12: Something to do with math

$$\det \begin{vmatrix} c_0 & c_1 & c_2 & \cdots & c_n \\ c_1 & c_2 & c_3 & \cdots & c_{n+1} \\ c_2 & c_3 & c_4 & \cdots & c_{n+2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_n & c_{n+1} & c_{n+2} & \cdots & c_{2n} \end{vmatrix} > 0$$

Figure 13: Something to do with math

10.3. Lists

Unordered list:

- Foo
- Bar
- Baz
  - Fizz
  - Buzz
    - Qux
      - Quux
        - Quux
          - Quuux
            - Quuuux

Numbered list:

1. Foo
2. Bar
3. Baz
   1. Fizz

2. Buzz
    1. Qux
        1. Quux
            1. Quux
                1. Quuux
                    1. Quuuux

10.4. Prime Number Table

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

10.5. CJK

10.5.1. Tiles

10.5.2. I am a Cat

10.6. CJK Vertical Layout

Unfortunately Paged target doesn't support vertical layout yet.

Chapter 11

Code Demo

Typst provides syntax highlighting for code blocks. See https://typst.app/docs/reference/text/raw/ for details including how to change the scheme and how to add highlighting for custom languages.

11.1. Rust

```rust
use typst_wasm_protocol::wasm_export;

/// Tell me something and I'll repeat that
#[wasm_export]
pub fn echo(data: &[u8]) -> Result<Vec<u8>, String> {
    unsafe {
        let a: &str = Ok("I'm doing nothing...").unwrap_unchecked();
    }
    Err("unimplemented, do not call...".to_string())
}
```

11.2. C++

```cpp
#include <iostream>

int foo(int *a, int *b);

int main(int argc, char **argv) {
    int a = 21, b = 21;
    std::cout << "Hi!\nThe answer is " << foo(&a, &b) << "\n";
    return 0;
}

[[nodiscard]] int foo(int *a, int *b) {
    return *a + *b;
}
```

11.3. Python

```python
with open("file.txt", "w") as f:
    print("Hello, world!", file=f)
print("error!", file=f)
```

11.4. Typst Math

```
L = integral^b_a sqrt(1 + ((dif y) / (dif x))^2 dif x)
```

$$L = \int_a^b \sqrt{1 + \left(\frac{\mathrm{d}y}{\mathrm{d}x}\right)^2}\,\mathrm{d}x$$

11.5. Typst Markup

Source code of this file.

````
#title[Code Demo]

Typst provides syntax highlighting for code blocks. See #link("https://typst.app/docs/reference/text/raw/") for details
including how to change the scheme and how to add highlighting for custom languages.

= Rust

```rust
use typst_wasm_protocol::wasm_export;

/// Tell me something and I'll repeat that
#[wasm_export]
pub fn echo(data: &[u8]) -> Result<Vec<u8>, String> {
    unsafe {
        let a: &str = Ok("I'm doing nothing...").unwrap_unchecked();
    }
    Err("unimplemented, do not call...".to_string())
}
```

= C++

```cpp
#include <iostream>

int foo(int *a, int *b);

int main(int argc, char **argv) {
    int a = 21, b = 21;
    std::cout << "Hi!\nThe answer is " << foo(&a, &b) << "\n";
    return 0;
}

[[nodiscard]] int foo(int *a, int *b) {
    return *a + *b;
}
```

= Python

```py
with open("file.txt", "w") as f:
    print("Hello, world!", file=f)
print("error!", file=f)
```

= Typst Math

#let m = ```typm
L = integral^b_a sqrt(1 + ((dif y) / (dif x))^2 dif x)
```

````
````

```
#m

#math.equation(block: true, eval(m.text, mode: "math"))

= Typst Markup

_Source code of this file._

#raw(read("./demo-code.typ"), block: true, lang: "typ")

```

Chapter 12

Changelog

This is the change log for Haita, a pure Typst documentation framework.

0.4.0-rc1 (Aug. 3, 2026)

Added

- Favicon support, with a default favicon provided out of the box (#18)
- Copyable math blocks
- New documentation examples: KaTeX, CeTZ, and initial Tidy integration

Changed

- Renamed `book`'s `language` parameter to `lang` to match the renderer (**breaking change**, #15)
- Unified `canonical-url` and `root` into a single optional `base-url` parameter (**breaking change**, #16)
- Code copy buttons now use icons instead of text
- Sidebar focus ring now fits properly

Removed

- Footnote popup on hover (**breaking change**)

Fixed

- Overscroll behaviour on mobile
- `html.frame` nested inside `html.frame` not rendering properly
- Math block rendering inside frames
- Includes now always resolve relative to the workspace root (**breaking change**)
- Ace theme/mode failing to load in the JS embedding example (#17)
- Documentation typo

0.3.0 (Jul. 21, 2026)

There are no user-facing changes in this release.

0.3.0-rc1 (Jul. 19. 2026)

Added

- Pagefind search integration, including placement, styling, and behaviour improvements
  - No-JS banner with informational messaging when JavaScript is unavailable
- Code block line counts and a copy-to-clipboard button
- Vendored fonts.
  - Used Lato for main content, Lete Sans Math for math, and Roboto Mono for monospaced text.
- Basic accessibility links for TOC and main content
- Extended documentation

Changed

- Figure elements now centered
- Sidebar styles tweaked
- On-page TOC moved to right side of the page.
- Used relative CSS paths

Removed

- Google Fonts API dependency for *New Hamber*
- `separator` element, in favour of `std.divider` (**breaking change**)

Fixed

- Footnote color
- Pagefind nav box shrinking at small window heights
- Main TOC custom content styling

0.2.1 (Jul. 10, 2026)

- Improved the style of the handle for opening the navigation sidebar.

0.2.0 (Jul. 6, 2026)

- Changed name from `otter-docs` to `haita`.

0.1.0 (Jul. 3, 2026)

- *New Hamber* HTML renderer
- Basic documentation features
- Typst template