What's on this Page

CONTENT MANAGEMENT FUNDAMENTALS

Content Organization

Hugo assumes that the same structure that works to organize your source content is used to organize the rendered site.

Page Bundles

Hugo 0.32 announced page-relative images and other resources packaged into Page Bundles.

These terms are connected, and you also need to read about Page Resources and Image Processing to get the full picture.

The illustration shows three bundles. Note that the home page bundle cannot contain other content pages, although other files (images etc.) are allowed.

> The bundle documentation is a **work in progress**. We will publish more comprehensive docs about this soon.

Organization of Content Source

In Hugo, your content should be organized in a manner that reflects the rendered website.

While Hugo supports content nested at any level, the top levels (i.e. content/<DIRECTORIES>) are special in Hugo and are considered the content type used to determine layouts etc. To read more about sections, including how to nest them, see sections.

Without any additional configuration, the following will automatically work:

```
.
└── content
    └── about
    |   └── index.md  // <- https://example.com/about/
    ├── posts
    |   ├── firstpost.md   // <- https://example.com/posts/firstpost/
    |   ├── happy
    |   |   └── ness.md  // <- https://example.com/posts/happy/ness/
    |   └── secondpost.md  // <- https://example.com/posts/secondpost/
    └── quote
        ├── first.md       // <- https://example.com/quote/first/
        └── second.md      // <- https://example.com/quote/second/
```

Path Breakdown in Hugo

The following demonstrates the relationships between your content organization and the output URL structure for your Hugo website when it renders. These examples assume you are using pretty URLs, which is the default behavior for Hugo. The examples also assume a key-value of `baseURL = "https://example.com"` in your site's configuration file.

Index Pages: _index.md

`_index.md` has a special role in Hugo. It allows you to add front matter and content to your list templates. These templates include those for section templates, taxonomy templates, taxonomy terms templates, and your homepage template.

> **Tip:** You can get a reference to the content and metadata in `_index.md` using the `.Site.GetPage` function.

You can create one `_index.md` for your homepage and one in each of your content sections, taxonomies, and taxonomy terms. The following shows typical placement of an `_index.md` that would contain content and front matter for a `posts` section list page on a Hugo website:

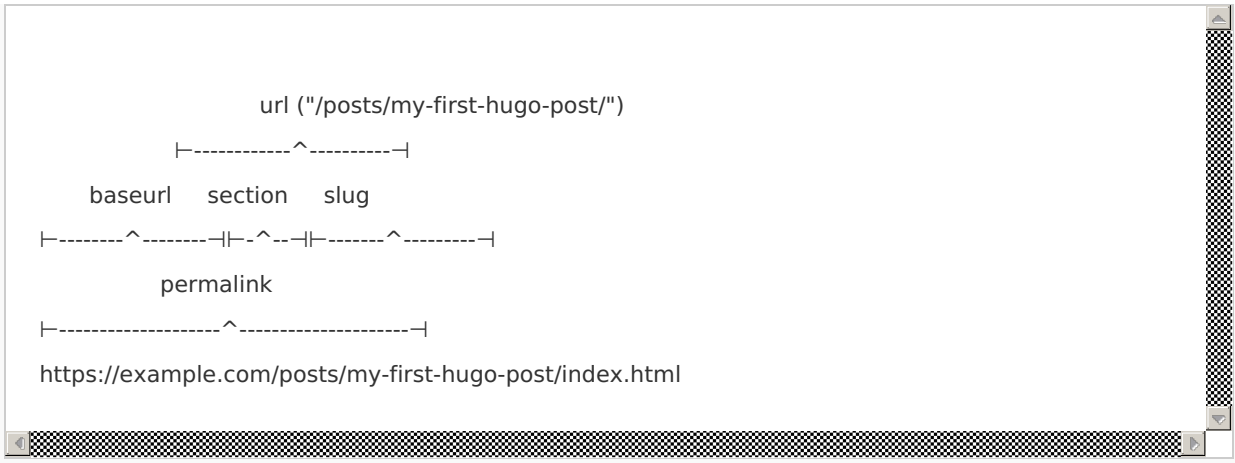
At build, this will output to the following destination with the associated values:

The sections can be nested as deeply as you want. The important thing to understand is that to make the section tree fully navigational, at least the lower-most section must include a content file. (i.e. _index.md).

Single Pages in Sections

Single content files in each of your sections will be rendered as single page templates. Here is an example of a single `post` within `posts`:

When Hugo builds your site, the content will be output to the following destination:



Paths Explained

The following concepts provide more insight into the relationship between your project's organization and the default Hugo behavior when building output for the website.

section

A default content type is determined by the section in which a content item is stored. `section` is determined by the location within the project's `content` directory. `section` *cannot* be specified or overridden in front matter.

slug

A content's `slug` is either `name.extension` or `name/`. The value for `slug` is determined by

- the name of the content file (e.g., `lollapalooza.md`) OR
- front matter overrides

path

A content's `path` is determined by the section's path to the file. The file `path`

- is based on the path to the content's location AND
- does not include the slug

url

The `url` is the relative URL for the piece of content. The `url`

- is based on the content item's location within the directory structure OR
- is defined in front matter, in which case it *overrides all the above*

Override Destination Paths via Front Matter

Hugo assumes that your content is organized with a purpose. The same structure that you use to organize your source content is used to organize the rendered site. As displayed above, the organization of the source content will be mirrored at the destination.

There are times when you may need more fine-grained control over the content organization. In such cases, the front matter field can be used to determine the destination of a specific piece of content.

The following items are defined in a specific order for a reason: items explained lower down in the list override higher items. Note that not all items can be defined in front matter.

filename

`filename` is not a front matter field. It is the actual file name, minus the extension. This will be the name of the file in the destination (e.g., `content/posts/my-post.md` becomes `example.com/posts/my-post/`).

slug

When defined in the front matter, the `slug` can take the place of the filename in the destination.

content/posts/old-post.md

```
---
title: A new post with the filename old-post.md
slug: "new-post"
---
```

This will render to the following destination according to Hugo's default behavior:

```
example.com/posts/new-post/
```

section

`section` is determined by a content item's location on disk and *cannot* be specified in the front matter. See sections for more information.

type

A content item's `type` is also determined by its location on disk but, unlike `section`, it *can* be specified in the front matter. See types. This can come in especially handy when you want a piece of content to render using a different layout. In the following example, you can create a layout at `layouts/new/mylayout.html` that Hugo will use to render this piece of content, even in the midst of many other posts.

content/posts/my-post.md

```
---
title: My Post
type: new
layout: mylayout
---
```

url

A complete URL can be provided. This will override all the above as it pertains to the end destination. This must be the path from the baseURL (starting with a `/`). `url` will be used exactly as it is defined in the front matter, and will ignore the `--uglyURLs` setting in your site configuration:

content/posts/old-url.md

```
---
title: Old URL
url: /blog/new-url/
---
```

Assuming your `baseURL` is configured to `https://example.com`, the addition of `url` to the front matter will make `old-url.md` render to the following destination:

```
https://example.com/blog/new-url/
```

You can see more information on how to control output paths in URL Management.

See Also

- Comments
- Page Resources
- Content Sections
- Content Types
- Related Content

"Content Organization" was last updated: November 8, 2021: Update index.md (#1570) (5fbe741d7)

IMPROVE THIS PAGE

By the Hugo Authors

File an Issue Get Help Discuss Source Code

@GoHugoIO @spf13 @bepsays

Hugo Sponsors

Menu Docs Menu

What's on this Page

CONTENT MANAGEMENT

Page Bundles

Content organization using Page Bundles

Page Bundles are a way to group Page Resources.

A Page Bundle can be one of:

- Leaf Bundle (leaf means it has no children)
- Branch Bundle (home page, section, taxonomy terms, taxonomy list)

	Leaf Bundle	Branch Bundle
Usage	Collection of content and attachments for single pages	Collection of attachments for section pages (home page, section, taxonomy terms, taxonomy list)
Index file name	index.md [1]	_index.md [1]
Allowed Resources	Page and non-page (like images, pdf, etc.) types	Only non-page (like images, pdf, etc.) types
Where can the Resources live?	At any directory level within the leaf bundle directory.	Only in the directory level **of** the branch bundle directory i.e. the directory containing the _index.md (ref).
Layout type	single	list
Nesting	Does not allow nesting of more bundles under it	Allows nesting of leaf or branch bundles under it

	Leaf Bundle	Branch Bundle
Example	content/posts/my-post/index.md	content/posts/_index.md
Content from non-index page files...	Accessed only as page resources	Accessed only as regular pages

Leaf Bundles

A *Leaf Bundle* is a directory at any hierarchy within the `content/` directory, that contains an **index.md** file.

Examples of Leaf Bundle organization

```
content/
├── about
│   ├── index.md
├── posts
│   ├── my-post
│   │   ├── content1.md
│   │   ├── content2.md
│   │   ├── image1.jpg
│   │   ├── image2.png
│   │   └── index.md
│   └── my-other-post
│       └── index.md
│
└── another-section
    ├── ..
    └── not-a-leaf-bundle
        ├── ..
        └── another-leaf-bundle
            └── index.md
```

In the above example `content/` directory, there are four leaf bundles:

about

This leaf bundle is at the root level (directly under `content` directory) and has only the `index.md`.

my-post

This leaf bundle has the `index.md`, two other content Markdown files and two image files.

image1

This image is a page resource of `my-post` and only available in `my-post/index.md` resources.

image2

This image is a page resource of `my-post` and only available in `my-post/index.md` resources.

my-other-post

This leaf bundle has only the `index.md`.

another-leaf-bundle

This leaf bundle is nested under couple of directories. This bundle also has only the `index.md`.

> The hierarchy depth at which a leaf bundle is created does not matter, as long as it is not inside another **leaf** bundle.

Headless Bundle

A headless bundle is a bundle that is configured to not get published anywhere:

- It will have no `Permalink` and no rendered HTML in `public/`.
- It will not be part of `.Site.RegularPages`, etc.

But you can get it by `.Site.GetPage`. Here is an example:

```
{{ $headless := .Site.GetPage "/some-headless-bundle" }}
{{ $reusablePages := $headless.Resources.Match "author*" }}
<h2>Authors</h2>
{{ range $reusablePages }}
  <h3>{{ .Title }}</h3>
  {{ .Content }}
{{ end }}
```

*In this example, we are assuming the `some-headless-bundle` to be a headless bundle containing one or more **page** resources whose `.Name` matches `"author*"`.*

Explanation of the above example:

1. Get the `some-headless-bundle` Page “object”.
2. Collect a *slice* of resources in this *Page Bundle* that matches `"author*"` using `.Resources.Match`.
3. Loop through that *slice* of nested pages, and output their `.Title` and `.Content`.

A leaf bundle can be made headless by adding below in the Front Matter (in the index.md):

```
headless = true
```

There are many use cases of such headless page bundles:

- Shared media galleries
- Reusable page content “snippets”

Branch Bundles

A *Branch Bundle* is any directory at any hierarchy within the `content/` directory, that contains at least an **`_index.md`** file.

This `_index.md` can also be directly under the `content/` directory.

> Here `md` (markdown) is used just as an example. You can use any file type as a content resource as long as it is a content type recognized by Hugo.

Examples of Branch Bundle organization

```
content/
├── branch-bundle-1
│   ├── branch-content1.md
│   ├── branch-content2.md
│   ├── image1.jpg
│   ├── image2.png
│   └── _index.md
└── branch-bundle-2
    ├── _index.md
    └── a-leaf-bundle
        └── index.md
```

In the above example `content/` directory, there are two branch bundles (and a leaf bundle):

branch-bundle-1

This branch bundle has the `_index.md`, two other content Markdown files and two image files.

branch-bundle-2

This branch bundle has the `_index.md` and a nested leaf bundle.

> The hierarchy depth at which a branch bundle is created does not matter.

---

1. The `.md` extension is just an example. The extension can be `.html`, `.json` or any valid MIME type. ↩

See Also

- Page Resources
- Content Organization
- Single Page Templates

"Page Bundles" was last updated: November 20, 2020: Fix typo in page bundles (#1283) (ae2dc138a)

IMPROVE THIS PAGE

By the Hugo Authors

File an Issue Get Help Discuss Source Code

@GoHugoIO @spf13 @bepsays

DEPLOYS BY netlify

Hugo Sponsors

linode

What's on this Page

CONTENT MANAGEMENT

Content Formats

Both HTML and Markdown are supported content formats.

You can put any file type into your `/content` directories, but Hugo uses the `markup` front matter value if set or the file extension (see `Markup identifiers` in the table below) to determine if the markup needs to be processed, e.g.:

- Markdown converted to HTML
- Shortcodes processed
- Layout applied

List of content formats

The current list of content formats in Hugo:

Name	Markup identifiers	Comment
Goldmark	md, markdown, goldmark	Note that you can set the default handler of `md` and `markdown` to something else, see Configure Markup. New in v0.60.0
Blackfriday	blackfriday	Blackfriday will eventually be deprecated.
MMark	mmark	Mmark is deprecated and will be removed in a future release.

Name	Markup identifiers	Comment
Emacs Org-Mode	org	See go-org.
AsciiDoc	asciidocext, adoc, ad	Needs Asciidoctor installed.
RST	rst	Needs RST installed.
Pandoc	pandoc, pdc	Needs Pandoc installed.
HTML	html, htm	To be treated as a content file, with layout, shortcodes etc., it must have front matter. If not, it will be copied as-is.

The `markup identifier` is fetched from either the `markup` variable in front matter or from the file extension. For markup-related configuration, see Configure Markup.

External Helpers

Some of the formats in the table above need external helpers installed on your PC. For example, for AsciiDoc files, Hugo will try to call the `asciidoctor` command. This means that you will have to install the associated tool on your machine to be able to use these formats.

Hugo passes reasonable default arguments to these external helpers by default:

- asciidoctor: --no-header-footer -
- rst2html: --leave-comments --initial-header-level=2
- pandoc: --mathjax

> Because additional formats are external commands, generation performance will rely heavily on the performance of the external tool you are using. As this feature is still in its infancy, feedback is welcome.

External Helper AsciiDoc

AsciiDoc implementation EOLs in Jan 2020 and is no longer supported. AsciiDoc development is being continued under Asciidoctor. The format AsciiDoc remains of course. Please continue with the implementation Asciidoctor.

External Helper Asciidoctor

The Asciidoctor community offers a wide set of tools for the AsciiDoc format that can be installed additionally to Hugo. See the Asciidoctor docs for installation instructions. Make sure that also all optional extensions like `asciidoctor-diagram` or `asciidoctor-html5s` are installed if required.

> External `asciidoctor` command requires Hugo rendering to *disk* to a specific destination directory. It is required to run Hugo with the command option `--destination`.

Some Asciidoctor parameters can be customized in Hugo:

Parameter	Comment
backend	Don't change this unless you know what you are doing.
doctype	Currently, the only document type supported in Hugo is `article`.
extensions	Possible extensions are `asciidoctor-html5s`, `asciidoctor-bibtex`, `asciidoctor-diagram`, `asciidoctor-interdoc-reftext`, `asciidoctor-katex`, `asciidoctor-latex`, `asciidoctor-mathematical`, `asciidoctor-question`, `asciidoctor-rouge`.
attributes	Variables to be referenced in your AsciiDoc file. This is a list of variable name/value maps. See Asciidoctor's attributes.
noHeaderOrFooter	Output an embeddable document, which excludes the header, the footer, and everything outside the body of the document. Don't change this unless you know what you are doing.
safeMode	Safe mode level `unsafe`, `safe`, `server` or `secure`. Don't change this unless you know what you are doing.
sectionNumbers	Auto-number section titles.
verbose	Verbosely print processing information and configuration file checks to stderr.
trace	Include backtrace information on errors.
failureLevel	The minimum logging level that triggers a non-zero exit code (failure).

Hugo provides additional settings that don't map directly to Asciidoctor's CLI options:

workingFolderCurrent

Sets the working directory to be the same as that of the AsciiDoc file being processed, so that `include` will work with relative paths. This setting uses the `asciidoctor` cli parameter `--base-dir` and attribute `outdir=`. For rendering diagrams with `asciidoctor-diagram`, `workingFolderCurrent` must be set to `true`.

preserveTOC

By default, Hugo removes the table of contents generated by Asciidoctor and provides it through the

built-in variable `.TableOfContents` to enable further customization and better integration with the various Hugo themes. This option can be set to `true` to preserve Asciidoctor's TOC in the generated page.

Below are all the AsciiDoc related settings in Hugo with their default values:

config. yaml toml json

```yaml
markup:
  asciidocExt:
    attributes: {}
    backend: html5
    extensions: []
    failureLevel: fatal
    noHeaderOrFooter: true
    preserveTOC: false
    safeMode: unsafe
    sectionNumbers: false
    trace: false
    verbose: false
    workingFolderCurrent: false
```

Notice that for security concerns only extensions that do not have path separators (either `\`, `/` or `.`) are allowed. That means that extensions can only be invoked if they are in one's ruby's `$LOAD_PATH` (ie. most likely, the extension has been installed by the user). Any extension declared relative to the website's path will not be accepted.

Example of how to set extensions and attributes:

```
[markup.asciidocExt]
    extensions = ["asciidoctor-html5s", "asciidoctor-diagram"]
    workingFolderCurrent = true
    [markup.asciidocExt.attributes]
        my-base-url = "https://example.com/"
        my-attribute-name = "my value"
```

In a complex Asciidoctor environment it is sometimes helpful to debug the exact call to your external helper with all parameters. Run Hugo with `-v`. You will get an output like

```
INFO 2019/12/22 09:08:48 Rendering book-as-pdf.adoc with C:\Ruby26-x64\bin\asciidoctor.bat using as
```

Learn Markdown

Markdown syntax is simple enough to learn in a single sitting. The following are excellent resources to get you up and running:

- Daring Fireball: Markdown, John Gruber (Creator of Markdown)
- Markdown Cheatsheet, Adam Pritchard
- Markdown Tutorial (Interactive), Garen Torikian
- The Markdown Guide, Matt Cone

See Also

- Shortcodes
- markdownify
- .RenderString
- anchorize

"Content Formats" was last updated: August 21, 2021: highlight: Remove some pygments references (8fcf2c55d)

IMPROVE THIS PAGE

Hugo Sponsors

 The Hugo Gopher is based on an original work by Renée French.

Menu | Docs Menu