html_of_wiki

*`html_of_wiki` is a versatile, minimalist yet powerful **static website generator**. It is designed with simplicity in mind: no special directories, almost no configuration and everything working out of the box. No web server required, just the compilation of your content to HTML and you're free to choose which static files server service fits you: GitHub Pages, GitLab Pages, etc.*

*It allows writing website content in a feature-rich and extensible language: the **wikicréole**. This language, thanks to its clever design, allows to write rich content as much expressively as with plain HTML.*

`html_of_wiki` is part of the Ocsigen project and is used to generate its documentation—including this page! It has no problem dealing with large websites, it can handle several versions of several projects.

`html_of_wiki` is also composable, it can be integrated into a CI/CD process so you can automate the deployment of your website. Rebuild and deploy on each commit!

Installation

Clone this repository:

```
$ git clone https://github.com/ocsigen/html_of_wiki.git
```

then, use `opam pin` to install it:

```
$ opam pin add -y html_of_wiki html_of_wiki
```

What you will learn here

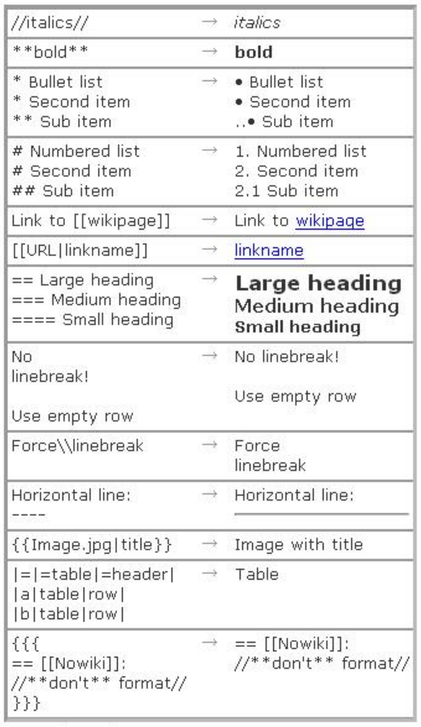
The following sections describe:

1. The wikicréole format
2. How to use `ohow`, a wikicréole compiler to HTML
3. How to use
    1. `dop` for generating the documentation of a whole project's version, and
    2. `quickdop` for generating the documentation of a whole project's.
4. The available extensions

The wikicréole

The standard

The following picture summarises the wikicréole 1.0 format.



//italics//	→	*italics*
bold	→	**bold**
* Bullet list	→	• Bullet list
* Second item		• Second item
** Sub item		..• Sub item
# Numbered list	→	1. Numbered list
# Second item		2. Second item
## Sub item		2.1 Sub item
Link to [[wikipage]]	→	Link to wikipage
[[URL|linkname]]	→	linkname
== Large heading	→	**Large heading**
=== Medium heading		**Medium heading**
==== Small heading		**Small heading**
No	→	No linebreak!
linebreak!		
		Use empty row
Use empty row		
Force\\\\linebreak	→	Force
		linebreak
Horizontal line:	→	Horizontal line:

{{Image.jpg|title}}	→	Image with title
|=|=table|=header|	→	Table
|a|table|row|		
|b|table|row|		
{{{	→	== [[Nowiki]]:
== [[Nowiki]]:		//**don't** format//
//**don't** format//		
}}}		

www.wikicreole.org

Notes:

- The wikicréole supported by `html_of_wiki` uses only one `=` sign for toplevel headings.
- There is also the tilde character `~` for not interpreting the character it prefixes.

Syntax additions

Supported links syntaxes

The following table lists additional link syntaxes supported by the tool.

Syntax	Description
`[[wiki("name"):page]]`	Link to the `page` of the project `name`
`[[wiki:page]]`	Link to the `page` of the current project
`[[site:page]]`	Website root-relative link
`[[href:path]]`	Raw `href` value

The following table lists available links abbreviations.

Abbreviation	Equivalent syntax	Description
`[[]]`	`[[href:.]]`	Current page
`[[#anchor]]`	`[[href:#anchor]]`	Current page with anchor
`[[/path]]`	`[[site:path]]`	Website root-relative link
`[[path]]`	`[[href:path]]`	Relative link

Decorations

The following table lists the additional available decorations.

Syntax	Description	Example
`--`	*en-dash*	–
`---`	*em-dash*	—
`##text##`	Mono-spaced font	`text`
`^^super^^`	Superscript	text^{super}
`,,sub,,`	Subscript	text$_{sub}$
`__text__`	Underline	text
`/-text-/`	Strike-through	~~text~~

Definition lists

The following code produces a definition list.

```
;title1
:definition1
;title2
:definition2
```

title1
: definition1

title2
: definition2

Inline HTML classes

It is possible to inline some HTML classes to the following element using the syntax `@@class="title"@@`.

Extensions

The wikicréole supported by `html_of_wiki` support **extensions**, a powerful mechanism that allows executing arbitrary OCaml code registered to the parser. More detail about this mechanism can be found in the last section of this document.

Extensions syntax: `<<extension attr1="val1" attr2="val2" ... argN="valN"|content>>`

Here are some common, widely used by Ocsigen's documentation extensions:

`<<a_manual>>`
: Link to a page of the manual of a project.

`<<span>>`
: Inserts an HTML `<span/>` element allowing a fine control over the DOM of the page.

`<<doctree>>`
: Displays a menu like the one on the left of this text.

How to use ohow (one_html_of_wiki)

one_html_of_wiki is a CLI tool for generating a *single* HTML document from a *single* wikicréole file.

Basic usage

```
$ ohow file.wiki # generates file.html
$ ohow -o somename.html file.wiki # generates somename.html
$ ohow --help # shows help
$ ohow --version # shows version
$ ohow --print file.wiki # prints the HTML to stdout instead of writing a file
$ ohow --headless file.wiki # do not include HTML head nor wrapping body tag
$ ohow --local file.wiki # generate local-navigation compatible links (REMOVE THAT FOR DEPLOYMENT)
```

Advanced usage

Templates

one_html_of_wiki supports the use of *templates*. A template is a classic wikicréole file which contains a (unique occurrence of) `<<content>>` somewhere. It is where the content will be inserted. Use the `--template` option to provide a path to the template.

Consider the following example for a deeper understanding on how to use templates:

```
$ cat template.wiki
<<head-css|
    .red {color: red}
    .blue {color: blue}
>>
@@class="red"@@Before.
<<div class="around"|
    <<content>>
>>
@@class="red"@@After.
$
$ cat page.wiki
I'm a wiki page, providing some <<span class="blue"|**useful**>> content!
$
$ ohow --template template.wiki --print page.wiki
<html><head><title></title><meta charset="utf8"/><style>
    .red {color: red}
    .blue {color: blue}
</style></head><body><p class="red">Before.
</p><div class="around"><p>I'm a wiki page, providing some
<span class="blue"><strong>useful</strong></span> content!</p></div>
<p class="red">After.</p></body></html>
$
```

There is also an extension, `<<include>>`, that allows a programmatic content insertion. This extension accepts two, *mutually exclusive*, attributes:

`template="PATH"`
: inserts the content of the file `$(dirname T)/PATH`, where `T` is the value of the *required* option `--template`.

`wiki="PATH"`

inserts the content of the file `$(dirname CF)/PATH`, where `CF` is the path to the currently compiled wiki file.

Notes:

- The option `--template` is only required if the `template` is used somewhere.
- This extension ignores its content.

wiki_in_template

If the template has to contain more than one `<<content>>` tag or if several templates have to be used, the integration of `wit` to your workflow has to be considered. *wiki_in_template* (`wit`) is a simple CLI tool for inserting content inside templates without converting them into HTML: it only deals with wikicréole.

Here's an example of how to use it to insert one page inside several templates:

```
$ ls
bottom.wiki hello.wiki top.wiki
$ cat hello.wiki
=Hello world!
This is the content wiki.
$ cat top.wiki
Template above.
<<content>>
$ cat bottom.wiki
<<content>>
Template below.
$
$ cat hello.wiki | wit top.wiki | wit bottom.wiki
Template above.
=Hello world!
This is the content wiki.
Template below.
$
```

And an example of how to insert several pages inside one template:

```
$ ls
title.wiki text.wiki template.wiki
$ cat title.wiki
=Hello world!
$ cat text.wiki
Some text.
$ cat template.wiki
Before first.
<<content>>
In between.
<<content>>
After.
$
$ wit <(wit template.wiki < title.wiki) < text.wiki
Before first.
=Hello world!
In between.
Some text.
After.
$
```

Finally, as far as `wit` is concerned, a page is a wiki file without `<<content>>` and a template is a wiki file with at least one occurrence of `<<content>>`—so there's nothing wrong in saying that `wit` can output a template.

A few notes:

- You cannot escape the string `<<content>>` or `<<content|>>` using the three curly braces syntax.
- You cannot write a comment with the string `<<content>>` or `<<content|>>` inside.
- The `<<include>>` extension is currently not supported by `wit`. (PR welcomed!)

Link extensions

There are several link extensions that ship with `ohow`. They differ from the classic link syntax because they do not explicitly state where is located the resource to link. For example, `<<a_manual project="eliom" chapter="intro" | introduction of Eliom's docs>>` leads to the chapter `intro` of the `eliom` project *without explicitly giving the location of that page*.

Here are the available extensions:

- `<<a_manual>>`
- `<<a_api>>`, `<<a_api_type>>` and `<<a_api_code>>`
- `<<a_img>>`
- `<<a_file>>`

and the attributes they accept:

Attribute	Extensions	Description	Default value
`project`	All except `a_img` and `a_file`	The project of the page	Current project
`chapter`	`a_manual`	The manual chapter to link	None
`subproject`	`a_api`, `a_api_type`, `a_api_code`	The targeted sub-project	None
`text`	`a_api`, `a_api_type`, `a_api_code`	Text of the produced link	What's documented
`version`	All except `a_img` and `a_file`	The version of the project containing the page	`latest`
`fragment`			

	All except `a_img` and `a_file`	HTML fragment	None
`src`	`a_img`, `a_file`	The path to the resource to link	*Required*

However, even if the extensions doesn't require the *writer* to explicitly give the linked resource's location, the compiler still needs to know where to find it. It is why, whenever any of these extensions is used inside a wiki document, the writer must call `ohow` with the correct values for these options: `--manual`, `--api`, `--images`, `--assets` and `--root` (the latter defaults to the current working directory).

The *root* directory is the directory containing all wiki content for a specific project's version. For example, a project `proj/`, with two versions, `1.0/` and `2.0/`, each one containing the `man/`, `api/` and `assets/` directories. `ohow` must then be called with the following options (`pwd = ~/user/proj/`):

- `--root 2.0/`
- `--manual man/` — The path is relative to the *root* directory (not the current directory!). You could still have used `--manual 2.0/man/` (the root prefix is automatically stripped away).
- `--api 2.0/api/` — or equivalently, `--api api`
- `--assets assets`
- (`--images assets` — Only if you used `a_img` and that the images are in the `assets/` directory.)

For inter-project links, `ohow` **supposes that every projects are inside the same directory**. For example, `<<a_manual project="eliom" chapter="intro">>` will produce the following link: `rewinding to the root directory/../../eliom/path to the manual directory given with --manual/intro`. The first `../` rewinds from the root directory into the project directory (containing all versions). The second `../` rewinds inside the directory containing all the projects. It could, in practice, look like `../../../../eliom/latest/man/intro`.

html_of_wiki imposes these constraints for links because these are the constraints GitHub Pages imposes. To conclude, setting up these extensions is not as complicated as it sounds (it's roughly passing some paths to `ohow` through command line arguments) and the benefits are huge: this way you abstract away any hierarchy between wiki pages and projects and let the compiler work out the links by itself. Thus, any structural change of your projects **will not** force you to rewrite your documentation!

The `<<doctree>>` extension

`<<doctree>>` is an extension without parameters that inserts a menu. That menu is built by concatenating the content of all the files named `menu.wiki` inside the *root* directory. The order is:

1. the `menu.wiki` of the manual first, if any
2. the `menu.wiki` files of the API, if any, alphabetically sorted by sub-project's name, if any

For example, for a project with a manual and two API sub-projects, the extension will look for the following files, in that order:

1. `manual/menu.wiki`
2. `api/menu.wiki`
3. `api/sub-project1/menu.wiki`
4. `api/sub-project2/menu.wiki`

The menu on the left of this text is generated using that extension. Here is an example of what one can put inside a `menu.wiki` file:

```
=##html_of_wiki##
==[[#title|Introduction]]
=The wikicréole
==[[#wikistd|Standard]]
==[[#wikiadd|Additions]]
=[[#ohow|##ohow##]]
```

Again, that extension requires some extra parameters—`--root`, `--manual` and `--api`—which are described above. The generated element is a `<nav/>` with `class="how-doctree"`.

The `<<docversion>>` extension

`<<docversion>>` is an extension which inserts a dropdown list that lets the visitor select the version of the documentation to see. An example of such a widget on the left of this text.

To use it, place the versions to display in a file, say `versions.txt`, one per line, in the expected order. Then, pass that file as an argument of `ohow` using the option `--docversion`. If `--root`'s directory name is inside `versions.txt`, that entry will be automatically selected.

The extension places a text—`"Version"`—and a `<select/>` element with `class="how-versions"`. Required options are `--root` and `--docversion`.

The `<<client-server-switch>>` extension

WARNING: This extension is specific to the Ocsigen project.

To include a client/server switch, include `<<client-server-switch>>` without attributes somewhere in the template. The data of this extension is automatically collected by either dop or quickdop. At a lower level, this extension requires a parameter—`--csw`—which expects a file. Inside must figure the name a all wiki files (with extension `.wiki`), one per line, that should have a client/server switch. dop uses the Unix builtin `comm -12` to extract the files the sub-projects `client` and `server` have in common.

Legacy support

- In previous versions of `html_of_wiki`, projects used to have a file—`config.js`—which declaratively described these projects. Amongst that information, a field `default_subproject` declared the `<<a_api* subproject` attribute's default value. Since that file no longer exist, an option `--default-subproject` exists but is **highly deprecated**.
- That previous version also supported a special link syntax—`[[wiki(id):page]]`—in which projects were given an ID (in `config.js`). Considered unclear, that syntax is now **deprecated**, any wiki using it will *not* compile.

Higher level documentation generators

one_html_of_wiki works great for compiling one wiki page, but a project's documentation is often composed of many wiki pages for each version. However, `ohow` is designed with the Unix philosophy in mind: **simplicity** and **composability**. This, it is very easy to integrate it in an existing workflow.

To make things even easier, we provide out of the box two higher level documentation generators:

- *doc_of_project* (`dop`) — for generating a single version of a project
- *quick-doc_of_project* (`quickdop`) — for generating the documentation of a project with many versions

These tools are described in detail in the following sections.

dop

It is a wrapper on `ohow` designed to generate the document of a single version of a project with a one-line command.

Configuration file

Even if all configuration data could be passed using CLI options—as `ohow` does—it is often more convenient to have a configuration file. Two formats are accepted: *JSON* and *plain text*. Here is an example of the same configuration file, written in these two formats:

config.json:

```
{
  "project": "html_of_wiki",
  "manual": "man",
  "api": "api",
  "assets": "files",
  "images": "files/images",
  "csw": false,
  "menu": true,
  "templates": ["template1.wiki", "template2.wiki"]
}
```

config.txt:

```
project   html_of_wiki
manual    man
api       api
assets    files
images    files/images
csw       false
menu      true
templates template1.wiki:template2.wiki
```

Note that the JSON format needs **jq** (https://stedolan.github.io/jq/) installed and in the `PATH`. Use the plain format if `jq` cannot be installed. It also can be useful (thanks to its minimalism) if it needs to be algorithmically generated. It is read by `awk` with default settings: `$0` should contain the key, and the other records its value. Arrays have their value separated by a colon `:` (in a `PATH`-like fashion).

Before reading the configuration file you provide (if any), `dop` will try to **infer it** by analyzing the *root* directory. If the option `-i` is given, it will print that inferred configuration (either in JSON or in plain text). Then, if a configuration file is explicitly provided, its keys replaces the inferred ones and the ones *not* present keep their inferred values. First check what `dop` can infer for your organization's architecture from the doc's version its given (use the `-n` option for not generating the docs and just do the inferring work).

The following table describes the entries recognized by `dop`:

Key name	Type	`ohow` option	Short description
`project`	string	`--project`	Project's name
`manual`	string	`--manual`	Manual directory (root-relative path)
`api`	string	`--api`	API directory (root-relative path)
`client`	string	—	API's client directory (real path)
`server`	string	—	API's server directory (real path)
`assets`	string	`--assets`	Assets directory (root-relative path)
`images`	string	`--images`	Images directory (root-relative path)
`csw`	boolean	`--csw`	Contains a client-server-switch?

`menu`	boolean	`--doctree`	Contains a doctree?
`templates`	string array	`--template`	List of templates
`default_subproject`	string	`--default-subproject`	*Deprecated*

The `templates` key is an array because it would be possible (thanks to `wit`) to have several templates but this feature is not currently implemented in `dop`. PR welcomed! ;-)

Command-line interface

The command `dop` accepts the following short options:

Name	Option	Value	Default value	Description
Config	`-c`	File	—	Configuration file
Config type	`-t`	`json` or `plain`	`plain`	Configuration file type (format)
Clean	`-k`	Flag	—	Do not clean wikis after compilation
Inferred	`-i`	Flag	—	Print inferred configuration
Used	`-u`	Flag	—	Print used (final) configuration
No run	`-n`	Flag	—	Dry run (no compilation)
Root dir	`-r`	Name	`_dop`	Root directory (output directory)
Force	`-f`	Flag	—	Removes root dir if exists
Local	`-l`	Flag	—	Generate local links (`--local`)
Docversion	`-d`	File	—	`docversion` files (see `--docversion` option)
Verbose	`-v`	Flag	—	Verbose
Help	`-h`	Flag	—	Show help and exit

Examples

Minimal example

```
$ tree
.
├── 1.0
│   └── manual
│       └── intro.wiki
└── 2.0
    ├── api
    └── manual
        ├── intro.wiki
        └── other.wiki
$ dop 2.0
$ tree
.
├── 1.0
│   └── manual
│       └── intro.wiki
├── 2.0
│   ├── api
│   └── manual
│       ├── intro.wiki
│       └── other.wiki
└── _dop
    ├── api
    └── manual
        ├── intro.html
        └── other.html
```

More examples

You can look at the files called `.howdocgen` inside the `wikidoc` branch of each documented Ocsigen project for a real demonstration.

quickdop

To make things easier for projects with multiple versions of the documentation, html_of_wiki provides `quickdop`. It takes care of generating the docs for version found (directory name matching `[0-9]+|dev`) and provides automatically a value for the `--docversion` option.

Usage

```
quickdop [-f] PROJECT OUTDIR [DOP_OPTIONS]
```

`PROJECT`
: The directory containing the versions.

`OUTDIR`
: The build directory. Automatically created and replaced when the `-f` option is provided.

`DOP_OPTIONS`
: The options usually passed to `dop`. Obviously, the options `-r`, `-d` and the target are provided by `quickdop` and must not be explicitly given.

Built-in wikicréole extensions

html_of_wiki is shipped with a number of built-in, general purpose, wikicréole extensions. The documentation of the extensions doctree, docversion, client-server-switch, a_manual, a_api, a_api_type, a_api_code, a_img and a_file can be found in earlier sections of this document.

Each extension documented below accept the attributes class and id for explicitly giving an HTML class and an HTML id to the generated code.

Nameless extension

Acts as a comment, i.e., inserts no HTML. For example: `<<|I'm a comment: I will not appear inside the generated HTML>>`.

code and code-inline

Attributes
: `language="X"`, `translated="translated"` (optional, for `X` = `"ocaml"` only)

Description
: Inserts a code block.

HTML element
: `<pre class="manually-translated"><code class="language-X"></code></pre>` or `<pre><code class="language-ocaml translatable"></code></pre>`

Example

```
<<code language="ocaml"|int_of_string "12" + 3>>
```

produces

```
int_of_string "12" + 3
```

The code-inline extension is similar but inserts a code snippet inside the current paragraph and not in a block of its own.

div, span, nav, pre

Inserts the corresponding HTML elements.

googlesearch

Attributes
: `domain="DOMAIN"`, `icon="ICON"`

Description
: Inserts a search bar with icon `ICON` performing a Google search restricted to the specified domain `DOMAIN`.

Tyxml element

```
Html.[form ~a:[a_id "googlesearch";
                 a_action "https://google.com/search"]
          [input ~a:[a_name "q";
                     a_id "gsearch-box";
                     a_placeholder "Search using Google"]
             ();
           label ~a:[a_label_for "gsearch-box"]
             [img ~src:ICON ~alt:"" ~a:[a_id "gsearch-icon"] ()];
           input ~a:[a_input_type `Submit;
                     a_id "gsearch-submit";
                     a_onclick @@ "document.getElementById('gsearch-box').value +=
' site:" ^ DOMAIN ^ "';"]
             ()]]
```

Example

```
<<googlesearch domain="ocsigen.org" icon="search.svg">>
```

wip and wip-inline

Inserts the content of extension in a `<div/>` element with the class `wip`. You may want to add in your CSS the following rule:

```
.wip {display: none;}
```

`wip-inline` is the inline counterpart of `wip`.

`when-local` and `unless-local`

`when-local` inserts its content when `ohow` (or any higher level documentation generator) is called with the `--local` (or `-l`) option. `unless-local` inserts its content when not. This extension is useful for addressing complex resource linking issues.

`when-project`

Attributes
: `when="P"` or (**exclusive**) `unless="P"`

Description
: Inserts the content when (or unless) the project the wiki file belongs to is `P` (see the configuration key `project`).

`script`

Attributes
: `src="JS_SRC"`

Description
: Inserts a `<script>` tag importing the given `JS_SRC` JavaScript source file *inside the `<body>` of the document*.

`head-script`

Attributes
: `src="JS_SRC"`

Description
: Inserts a `<script>` tag importing the given `JS_SRC` JavaScript source file *inside the `<head>` of the document*.

`head-css`

Attributes
: `href="CSS_SRC"`

Description
: Inserts a `<link rel="stylesheet">` tag importing the given `CSS_SRC` CSS stylesheet *inside the `<head>` of the document*.