# oerforge.make

Hugo-style Markdown to HTML Static Site Generator (Python)

---

## Overview

`oerforge.make` provides functions for building static HTML sites from Markdown using Jinja2 templates, asset management, navigation, and download button generation. It is inspired by Hugo and designed for clarity, maintainability, and extensibility.

---

## Functions

### copy_static_assets_to_build

```
def copy_static_assets_to_build()
```

Copy static assets (CSS, JS, images) from `static/` to `build/`. Overwrites files each time it is called.

---

### get_available_downloads_for_page

```
def get_available_downloads_for_page(rel_path, page_dir=None)
```

Scan the published output directory for a page and return a list of available download formats.

**Parameters** - `rel_path` (str): Relative path to the HTML file (e.g., 'about/index.html'). - `page_dir` (str, optional): Directory containing downloadable files.

**Returns** - `list[dict]`: List of available downloads with label, filename, href, theme, and aria_label.

---

### build_download_buttons_context

```
def build_download_buttons_context(rel_path, page_dir=None)
```

Build the download buttons context for a page.

**Parameters** - `rel_path` (str): Relative path to the HTML file. - `page_dir` (str, optional): Directory containing downloadable files.

**Returns** - `list[dict]`: List of button dictionaries for the template.

---

**slugify**

```
def slugify(title: str) -> str
```

Convert a title to a slug suitable for folder names.

**Parameters** - `title` (str): Page or section title.

**Returns** - `str`: Slugified string.

---

**load_yaml_config**

```
def load_yaml_config(config_path: str) -> dict
```

Load and parse the YAML config file.

**Parameters** - `config_path` (str): Path to the YAML config file.

**Returns** - `dict`: Parsed configuration data.

---

**ensure_output_dir**

```
def ensure_output_dir(md_path)
```

Ensure the output directory for the HTML file exists, mirroring `build/files` structure.

**Parameters** - `md_path` (str): Path to the Markdown file.

---

**setup_template_env**

```
def setup_template_env()
```

Set up the Jinja2 template environment for rendering pages.

**Returns** - `jinja2.Environment`: Configured Jinja2 environment.

---

**render_page**

```
def render_page(context: dict, template_name: str) -> str
```

Render a page using Hugo-style templates.

**Parameters** - `context` (dict): Context dictionary for the template. - `template_name` (str): Name of the template file.

**Returns** - `str`: Rendered HTML string.

---

**generate_nav_menu**

```
def generate_nav_menu(context: dict) -> list
```

Generate top-level navigation menu items from the content table.

**Parameters** - `context` (dict): Context dictionary, typically with `rel_path`.

**Returns** - `list[dict]`: List of menu item dictionaries.

---

**get_header_partial**

```
def get_header_partial(context: dict) -> str
```

Render the header partial using Jinja2.

**Parameters** - `context` (dict): Context dictionary for the template.

**Returns** - `str`: Rendered header HTML.

---

**get_footer_partial**

```
def get_footer_partial(context: dict) -> str
```

Render the footer partial using Jinja2.

**Parameters** - `context` (dict): Context dictionary for the template.

**Returns** - `str`: Rendered footer HTML.

---

**convert_markdown_to_html**

```
def convert_markdown_to_html(md_path: str) -> str
```

Convert Markdown to HTML using markdown-it-py, rewriting local image paths and adding accessibility roles.

**Parameters** - `md_path` (str): Path to the Markdown file.

**Returns** - `str`: Rendered HTML string.

---

**convert_markdown_to_html_text**

```
def convert_markdown_to_html_text(md_text: str) -> str
```

Convert Markdown text to HTML using markdown-it-py, rewriting local image paths and adding accessibility roles.

**Parameters** - `md_text` (str): Markdown text.

**Returns** - `str`: Rendered HTML string.

---

**get_asset_path**

```
def get_asset_path(asset_type, asset_name, rel_path)
```

Compute the relative asset path for CSS, JS, or images based on page depth.

**Parameters** - `asset_type` (str): Asset type ('css', 'js', 'images'). - `asset_name` (str): Asset filename. - `rel_path` (str): Relative path to the page.

**Returns** - `str`: Relative asset path.

---

**add_asset_paths**

```
def add_asset_paths(context, rel_path)
```

Add asset paths (CSS, JS, logo) to the context for template rendering.

**Parameters** - `context` (dict): Context dictionary. - `rel_path` (str): Relative path to the page.

**Returns** - `dict`: Updated context dictionary.

---

**get_top_level_sections**

```
def get_top_level_sections(db_path=None)
```

Get all top-level sections from the database for section index generation.

**Parameters** - `db_path` (str, optional): Path to the SQLite database.

**Returns** - `list[tuple]`: List of (section_title, output_dir) tuples.

---

### build_section_indexes

```
def build_section_indexes()
```

Generate index.html files for all top-level sections using the section template.

---

### build_all_markdown_files

```
def build_all_markdown_files()
```

Build all Markdown files using Hugo-style rendering, using the first # header as the title.

---

### create_section_index_html

```
def create_section_index_html(section_title: str, output_dir: str, context: dict)
```

Generate section `index.html` using the `section.html` template.

**Parameters** - `section_title` (str): Title of the section. - `output_dir` (str): Output directory for the section index. - `context` (dict): Context dictionary for the template.

---

## Usage Example

```
from oerforge import make
make.build_section_indexes()
make.build_all_markdown_files()
```

---

## Requirements

- Python 3.7+
- Jinja2
- markdown-it-py
- mdit-py-plugins
- PyYAML
- SQLite3

---

## See Also

- Jinja2 Documentation
- markdown-it-py Documentation

## License

See `LICENSE` in the project root.