NOTE: TWML is in heavy development and subject to change!

1.0 GETTING STARTED

1.1 About TWML

TWML is an alternative solution to existing Markup Languages such as LaTeX, Markdown, Asciidoctor, or Troff. In essence, TWML is supposed to be the alternative that seeks to discover the sweet spot between being too simple and being too complex.

The driving force behind TWML is TailwindCSS, a "stylizing" framework for developing websites. Although TailwindCSS is mainly for website development, it is perfectly capable of creating stunning documents.

To simplify HTML, the Markup Language used by browsers, TWML, specifies a syntax that abstracts HTML for better readability and an extended feature set, such as importing templates.

Using TWML requires learning TailwindCSS and the basic structure of HTML.

1.2 Features and Shortcomings

TWML might be of interest to someone that:

- Already knows TailwindCSS
- Wishes to extend their documents through the use of Javascript or CSS software
- Searches for a more flexible alternative to existing Markup Languages

TWML might not be of interest to someone that:

- Is happy with existing Markup Languages
- Is not interested in learning TailwindCSS or HTML
- Does not wish to adapt to breaking changes (TWML is in early development)

1.3 Hello World!

Setting up TWML requires installing Rust. After setting up Rust , installing a development version of TWML should be as simple as running the following command in your shell of preference:

```
cargo install --git https://github.com/DISTREAT/twml
```

The following software should now be available:

twml-html: Convert a TWML document to html
twml-pdf: Convert a TWML document to pdf
twml-live: Start a webserver that serves a TWML document (used for live preview)

Let's create a simple document and convert it to PDF:

```
\p.font-bold Hello World!
```

Command:

```
twml-pdf document.twml document.pdf
```

Hello World!

This example should be rather simple to interpret. We create a paragraph with a bold font, containing the text "Hello World!".

In HTML terms this would translate to: *<p class="font-bold">Hello World!</p>*

1.3 Resources

The perfect resource for understanding how to style your documents is the official TailwindCSS documentation .

For a good HTML element reference see Mozilla's documentation .

1.4 Breakline-Sensitivity

TWML is breakline-sensitive, this means that empty lines matter:

```
This text is
\span.font-bold bold
!
```

This text is **bold** !

```
This text is

\span.font-bold bold
!
```

This text is
bold !

```
This text is
\p unformated
!
```

This text is
unformated
!

Note: Sometimes and due to the nature of HTML the way newlines are treated can become a bit confusing, but it'll become intuitive with time.

2.0 PAGES

2.1 Adding pages

TWML comes shipped with builtin CSS classes. One of them is the `page` class, used for creating new pages.

```
\div.page
    \p This is page number 1.

\div.page
    \p This is page number 2.
```

The above example will created a document with two pages.

2.2 Obtaining the page number

The class `page-number` replaces an element's content with the current page number.

```
\div.page
    This is page number:
    \span.page-number
```

This is page number: 1

2.3 Changing the paper size

Changing the paper size is done using top-level declarations in the root of the document:

```
@page-width 100
@page-height 200

\div.page
    This page is 100mm in width and 200mm in height.
```

2.4 PDF Outline

The following example adds a table of contents to the PDF (also called bookmarks):

```
\div.page
    \p.toc Page 1

\div.page
    \p.toc Page 2
```

Note: Defining pages is required in this specific case. The ToC class only work if an element has inline content and not content in form of children.

3.0 INCLUDING FILES

3.1 Adding a custom font

A custom font may be included by it's postscript name:

```
@font Cantarell-Regular

\p.font-cantarell-regular This text is in Cantarell-Regular
```

This text is in Cantarell-Regular

3.2 Including Files

Including files is similar to including fonts:

```
@include assets/bird.png

\img{src="bird.png" height="50"}
```

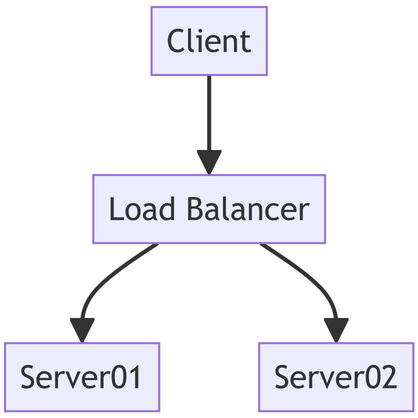
3.3 Including JS

JavaScript can extend the functionality and make it possible to add for example graphs.

```
@js https://cdn.jsdelivr.net/npm/mermaid@10.6.1/dist/mermaid.min.js

\pre.mermaid
    graph TD
    A[Client] --> B[Load Balancer]
    B --> C[Server01]
    B --> D[Server02]

\script
    mermaid.initialize({ startOnLoad: true });
```

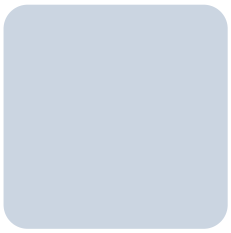


3.4 Including CSS

Similarly, CSS may be included:

```
@include skelly.css
@css skelly.css

\div.w-28.h-28.skelly
```



4.0 TEMPLATING

4.1 Concept

TWML supports including other documents. This is what in TWML terminology is refered to as *templating* or *templates* .

Documents may be imported from one of the following paths:

- The current working directory.
- *$HOME/.config/twml/templates/*
- */usr/share/twml/templates/*

4.2 Simple Example

./document.twml:

```
\!directory-hello
```

./directory/hello.twml:

```
\p Hello World!
```

Hello World!

The document *document.twml* will import *directory/hello.twml* when calling `\!hello` .

4.3 Attributes

Attributes are a way to pass variables to a template.

./document.twml:

```
\!hello{name="John"}
```

./hello.twml

```
\p Hello {name}!
```

Hello John!

Placeholders like these can be escaped using `{{name}}` .

4.4 Children

It is also possible to add children to an imported element:
./document.twml:

```
\!hello
    \p Hello there!
```

./hello.twml

```
\p.font-bold
    {...}
```

Hello there!

Placeholders like these can be escaped using *{{name}}* .

Note: \p.font-bold {...} would be illegal. {...} must always be a child of an element.

4.5 Classes

Templates may also hold placeholders for additional classes:

./document.twml:

```
\!hello.font-bold
```

./hello.twml

```
\p.$ Hello there!
```

Hello there!