Bullseye

Hit the target (HTML or paged/PDF) when styling your Typst document

v0.0.1

https://github.com/SillyFreak/typst-bullseye

Clemens Koza

Contents

I Introduction 2
I.a Experimental feature polyfills 2
II Module reference 3
II.a `bullseye` 3
II.b `bullseye.html` 4

I Introduction

Bullseye supports you in writing packages and documents that target multiple outputs, i.e. currently (Typst 0.13) `"paged"` (PDF, image) and `"html"`.

I.a Experimental feature polyfills

Typst's `html` module and the `target()` function for determining the kind of output are currently unstable, meaning they can't be used without a feature flag. This is an obstacle for packages and documents that want to optionally support HTML output; suppose you need some separate content for the two formats, and so you may write some code like this:

```typ
// for PDF output, Typst handles syntax highlighting
#let snippet = ```py
def x(): pass
```
// for HTML output, syntax highlighting is added externally through a CSS class
#let html-snippet = html.elem("code", attrs: (class: "lang-py"), snippet.text)

// conditionally render either of these
#context if target() == "html" {
  html-snippet
} else {
  snippet
}
```

However, compiling this will result in an error:

```
$ typst compile test.typ
error: unknown variable: html
   | #let html-snippet = html.elem("code", attrs: (class: "lang-py"), snippet.text)
   |                      ^^^^
```

Even though the current export target is pdf, the fact that we have not enabled HTML support makes this code fail! Some workarounds for this problem include

- Restructure the code to only call experimental functions when HTML export is requested. This doesn't always make the code more manageable, and note that the `target()` function itself is among the unstable functions.
- Require the user to always enable HTML support:

  ```
  $ typst compile --features html test.typ
  ```

  This is especially annoying when writing packages for people who may or may not be interested in HTML export. Also, this requires different approaches for plain CLI compilation (demonstrated above), Tinymist users, or web app users (not supported),

For this reason, Bullseye "polyfills" the `target()` function when HTML support is not enabled, and contains a stub `html` module that allows compiling code *calling* but not *rendering* HTML elements:

```typ
#import "@preview/bullseye:0.0.1": target, html
#let snippet = /* ... */
#let html-snippet = html.elem(/* ... */)
#context if target() == "html" { /* ... */ }
```
```

There are two scenarios in which this code can be executed:

- HTML support is not enabled: Bullseye's polyfills are used. The `target()` function always returns `"paged"`, and the `html.elem()` and `html.frame()` functions don't do anything useful. If you tried to unconditionally put `html-snippet` into your document, it would panic.
- HTML support is enabled: Bullseye's exports simply forward to the standard ones. The `target()` function returns the same result as `std.target()`, and `html` is an exact alias to `std.html`. This is is independent from the export *target*, but it usually won't make a difference if not exporting to HTML.

There is one small difference between the polyfilled and original `html` module: when not exporting to HTML, the if an original `std.html.elem()` appears in the document, it will result in a warning; Bullseye's `html.elem()` will panic instead!

II Module reference

II.a `bullseye`

- `target()`
- `html`


```
target() -> str
```

Returns `"paged"` or `"html"` depending on the current output target.

When HTML is supported, this is equivalent to Typst's built-in `std.target()`; otherwise, it always return `"paged"`.

This is a polyfill for an unstable Typst function. It may not properly emulate the built-in function if it is changed before stabilization.

This function is contextual.

```
html: module
```

The `html` module.

When HTML is supported, this is equivalent to Typst's built-in `std.html`; otherwise, it's the Bullseye module documented below. That module doesn't *support* HTML, it just makes sure that calls to the html module that don't end up in a document don't prevent compilation.

This is a polyfill for an unstable Typst module. It may not properly emulate the built-in module (i.e. miss functions; no functionality beyond that is intended) if it is changed before stabilization.

II.b `bullseye.html`

- `elem()`
- `frame()`

`elem(..args)`

A stub function for `std.html.elem()`. This function simply contextually panics, i.e. it can be called but its result must not appear in a document:

```typ
// assume `--features html` is not active
#let x = std.html.elem("div")  // panics, because `std.html` does not exist
#let y = bullseye.html.elem("div")  // ok
#y  // panics, because bullseye doesn't actually implement HTML elements
```

This is useful for writing code where alternative content for HTML is specified, but only rendered when HTML support is activated.

`frame(..args)`

A stub function for `std.html.frame()`. This function simply contextually panics, i.e. it can be called but its result must not appear in a document:

```typ
// assume `--features html` is not active
#let x = std.html.frame[body]  // panics, because `std.html` does not exist
#let y = bullseye.html.frame[body]  // ok
#y  // panics, because bullseye doesn't actually implement HTML elements
```

This is useful for writing code where alternative content for HTML is specified, but only rendered when HTML support is activated.