hydra

v0.3.0 2024-01-06

https://github.com/tingerrr/hydra

tinger <me@tinger.dev>

Abstract

A package for querying and displaying heading-like elements.

Contents

I Introduction

`hydra` is a package primarily for displaying the active section or chapter in the header of your document.

1. Terminology & Semantics

The following terms are often used in the following sections to explain the behavior and reasoning of `hydra`:

primary The element which is primarily looked for and meant to be displayed.

ancestor An element which is the immediate or transitive ancestor to the primary element. A level 3 heading is ancestor to both level 2 (directly) and level 1 headings (transitively).

scope The scope of a primary element refers to the section of a document which is between the closest ancestors.

active The active element refers to whatever element is considered for display. While this is usually the previous primary element, it may sometimes be the next primary element.

The search for a primary element is always bounded to it's scope, such that, for the following simplified document, the output of `hydra` does not revert to Section 1.1.

```
= Chapter 1
== Section 1.1

= Chapter 2
=== Subsection 2.1.1
#hydra(2)
```

For this the ancestors of an element must be known. For headings this is simple:

`none` ⟶ `level: 1` ⟶ `level: 2` ⟶ `level: 3` ⟶ `...`

If `hydra` is used to query for level 2 headings it will only do so within the bounds of the closest level 1-3 headings. In principle, elements other than headings can be used (see Section 2.), as long as their semantic relationships are established.

II Features

1. Sane Defaults

If `hydra` is called with no arguments it will not assume anything about your document other than the typst defaults. I.e. that the paper size is `a4`, that the margins and page sizes are not set and that you intend to simply show the last active heading without showing it where it's obvious.

2. Custom Elements

Because some documents may use custom elements of some kind to display chapters or section like elements, `hydra` allows defining its own selectors for tight control over how elements are semantically related.

Given a custom element like so:

```
#let chapter = figure.with(kind: "chapter", supplement: [Chapter])
// ... show rules and additional setup

#chapter[Introduction]
#chapter[Main]
= Section 1.1
== Subsection 1.1.1
= Section 1.2
#chapter[Annex]
```

A user my want to query for the current chapter and section respectively:

```
#import "@preview/hydra:0.3.0": hydra, selectors
#import selectors: custom

#let chap = figure.where(kind: "chapter")
#let sect = custom(heading.where(level: 1), ancestor: chap)

#set page(header: locate(loc => if calc.odd(loc.page()) {
  algin(left, hydra(chap))
} else {
  algin(right, hydra(sect))
}))
```

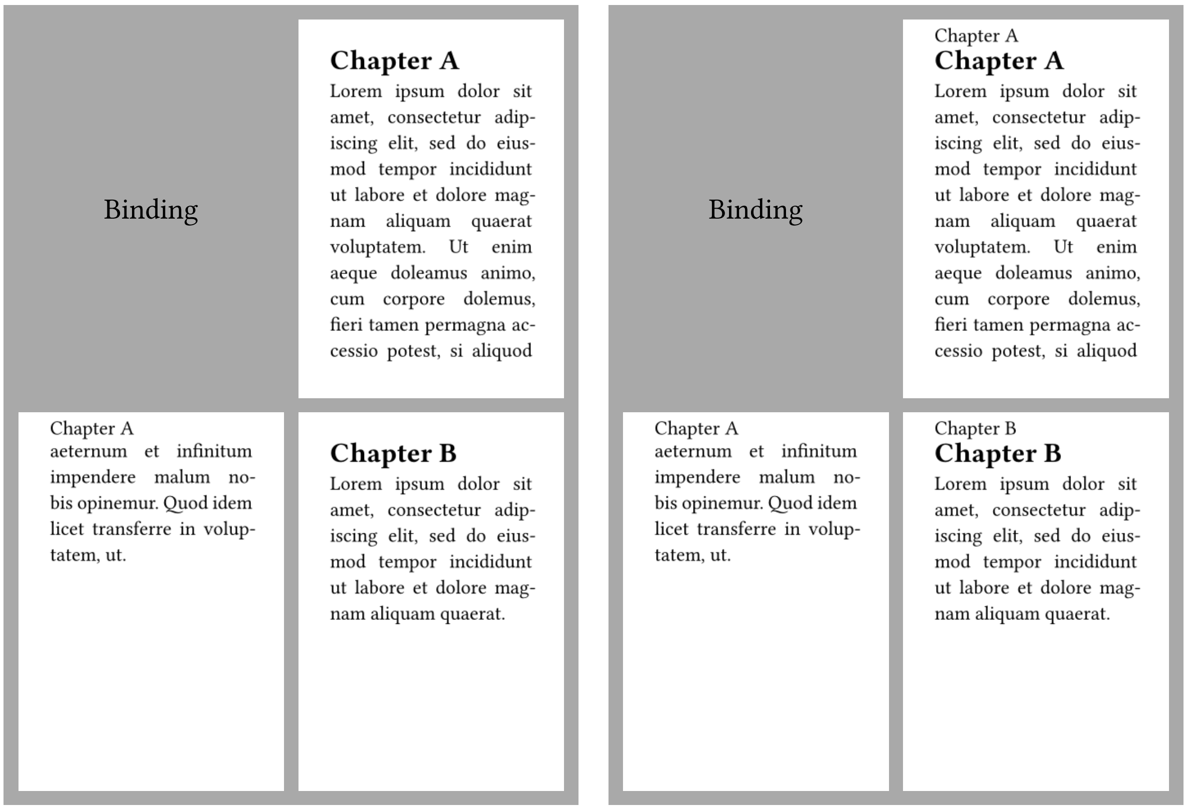
The usage of `custom` allows specifying an element's ancestors, to ensure the scope is corectly defined.

3. Redundancy Checks

Generally `hydra` is used for heading like elements, i.e. elements which semantically describe a section of a document. Whenever `hydra` is used in a place where its output would be redundant, it will not show any output by default. The following sections explain the those checks more closely and will generally assume that hydra is looking for headings.

3.1. Starting Page

Given a page which starts with a primary element, it will not show anything. If `skip-starting` is set to `false`, it will fallback to the next element, in this case the heaiding at the top of the page.



Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut.

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat.

Figure 1: An example document showing `skip-starting: true` (left) and `skip-starting: false` (right).

For more complex selectors this will not correctly work if the first element on this page is an ancestor. See hydra#8.

3.2. Book Mode

Given an odd page, if `book` is set to `true`, then if the previous primary element is still visible on the previous page it is also skipped.

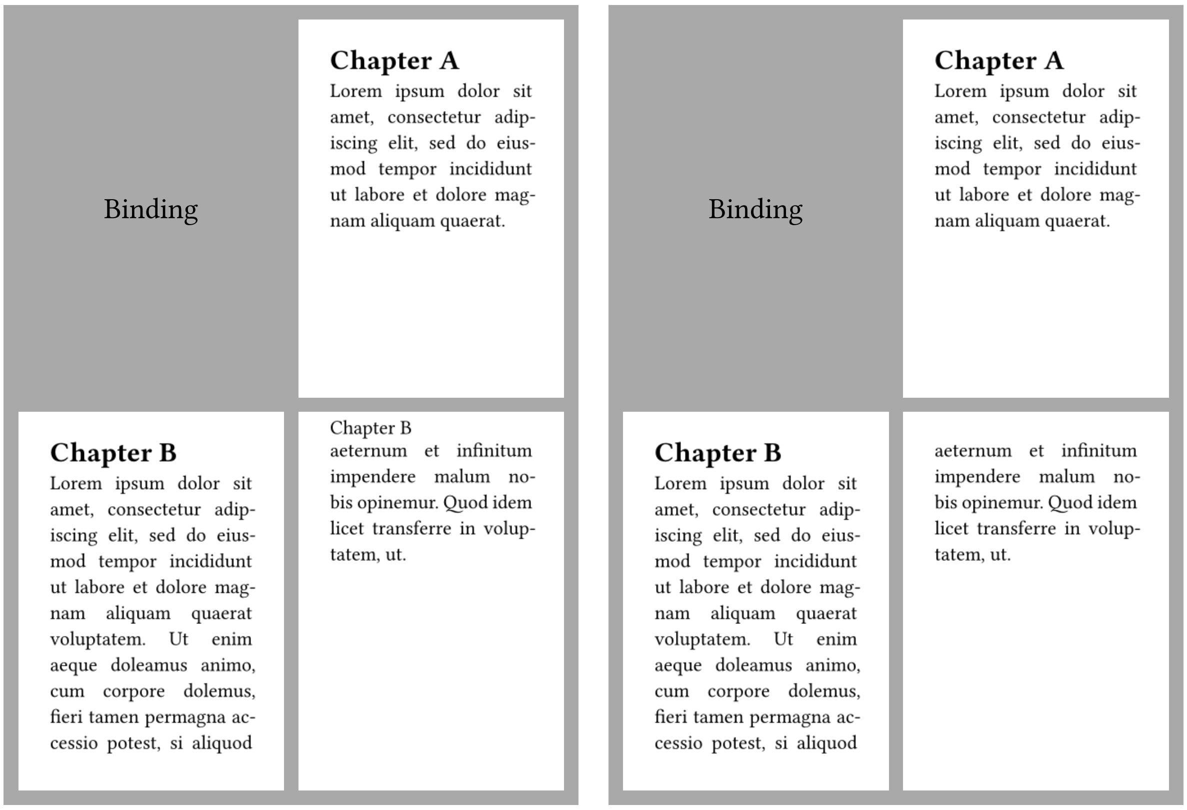

Binding
Chapter A
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat.
Chapter B
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod
Chapter B
aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut.
Binding
Chapter A
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat.
Chapter B
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod
aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut.

Figure 2: An example document showing `book: false` (left) and `book: true` (right).

4. Optional Function Coloring

Hydra requires a context to work, more specifically it needs to know it's own location relative to the elements it queries for. To avoid the need for a user having to use hydra inside `locate` all the time `hydra` will do it by itself. But if it always did this, it would not allow the user to actually check the return value. The following will not work:

```
#import "@preview/hydra:0.3.0": hydra
#set page(header: {
  let chap = hydra(1)
  if chap != none [
    Chapter #chap
  ]
})
```

Because `hydra` needs a location it'll internally call `locate`, making the return value a `locate` element. The fix is quite simple, if a location is provided the return is not wrapped and the callback result is returned as is.

```
#import "@preview/hydra:0.3.0": hydra
#set page(header: locate(loc => {
  let chap = hydra(1, loc: loc)
  if chap != none [
    Chapter #chap
  ]
}))
```

This means that passing a location in contexts where one is already available will generally avoid unecessary function coloring. This allows for more complex queries in casese where both a chapter and section are shown for example.

5. Anchoring

To use `hydra` outside of the header, an `anchor` must be placed to get the correct active elements. `hydra` will always use the last anchor it finds to search, it doesn't have ot be inside the header, but should generally be, otherwise the behavior may be unexpected.

```
#import "@preview/hydra:0.3.0": hydra, anchor
#set page(header: anchor(), footer: hydra())
```

III Reference

1. Stability

The following stability guarantees are made, this package tries to adhere to semantic versioning.

`unstable`	API may change with any version bump.
`stable`	API will not change without a major version bump or a minor version bump pre `1.0.0`, if such a change occures it is a bug and unintended.

2. Custom Types

The following custom types are used to pass around information easily:

2.1. `sanitized-selector` `stable`

Defines a selector for an ancestor or primary element.

```
(
  target: queryable,
  filter: ((context, candidates) => bool) | none,
)
```

2.2. `hydra-selector` `stable`

Defines a pair of primary and ancestor element selectors.

```
(
  primary: sanitized-selector,
  ancestors: sanitized-selector | none,
)
```

2.3. `candidates` `stable`

Defines the candidates that have been found in a specific context.

```
(
  primary: (prev: content | none, next: content | none)
  ancestor: (prev: content | none, next: content | none)
)
```

2.4. `context` `unstable`

Defines the options passed to hydra nad resolved contextual information needed for querying and displaying.

```
(
  prev-filter: (context, candidates) => bool,
  next-filter: (context, candidates) => bool,
  display: (context, content) => content,
  skip-starting: bool,
  book: bool,
  top-margin: length,
  anchor: label | none,
  loc: location,
  primary: sanitized-selector,
  ancestors: sanitized-selector,
)
```

`hydra` `stable`

The package entry point. All functions validate their inputs and panic using error messages directed at the end user.

- anchor()
- hydra()

```
anchor()
```

An anchor used to search from. When using `hydra` ouside of the page header, this should be placed inside the pge header to find the correct searching context. `hydra` always searches from the last anchor it finds, if and only if it detects that it is outside of the top-margin.

```
hydra(
    prev-filter: function,
    next-filter: function,
    display: function,
    skip-starting: bool,
    book: bool,
    paper: str,
    page-size: length auto,
    top-margin: length auto,
    anchor: label none,
    loc: location none,
    ..sel: any
  ) -> content
```

Query for an element within the bounds of its ancestors.

The context passed to various callbacks contains the resolved top-margin, the current location, as well as the binding direction, primary and ancestor element selectors and customized functions.

Parameters:

`prev-filter` (`function` = `(ctx, c) => true`) – A function which receives the `context` and `candidates`, and returns if they are eligible for display. This function is called at most once. The primary next candidate may be none.

`next-filter` (`function` = `(ctx, c) => true`) – A function which receives the `context` and `candidates`, and returns if they are eligible for display. This function is called at most once. The primary prev candidate may be none.

`display` (`function` = `core.display`) – A function which receives the `context` and candidate element to display.

`skip-starting` (`bool` = `true`) – Whether `hydra` should show the current candidate even if it's on top of the current page.

`book` (`bool` = `false`) – The binding direction if it should be considered, `none` if not. If the binding direction is set it'll be used to check for redundancy when an element is visible on the last page.

`paper` (`str` = `"a4"`) – The paper size of the current page, used to calculate the top-margin.

`page-size` (`length` or `auto` = `auto`) – The smaller page size of the current page, used to calculated the top-margin.

`top-margin` (`length` or `auto` = `auto`) – The top margin of the current page, used to check if the current page has a primary candidate on top.

`anchor` (`label` or `none` = `<hydra-anchor>`) – The label to use for the anchor if `hydra` is used outside the header. If this is `none`, the anchor is not searched.

`loc` (`location` or `none` = `none`) – The location to use for the callback, if this is not given `hydra` calls locate internally, making the return value opaque.

`..sel` (`any`) – The element to look for, to use other elements than headings, read the documentation on selectors. This can be an element function or selector, an integer declaring a heading level.

`core` `unstable`

The core logic module. Some functions may return results with error messages that can be used to panic or recover from instead of panicking themselves.

- display()
- execute()
- get-anchor-pos()
- get-candidates()
- is-active-redundant()
- is-active-visible()
- is-on-starting-page()

display(ctx: `context`, candidate: `content`) -> `content`

Display a heading's numbering and body.

Parameters:

`ctx` (`context`) – The context in which the element was found.

`candidate` (`content`) – The heading to display, panics if this is not a heading.

execute(ctx: `context`) -> `content`

Execute the core logic to find and display elements for the current context.

Parameters:

`ctx` (`context`) – The context for which to find and display the element.

get-anchor-pos(ctx: `context`) -> `location`

Get the last anchor location. Panics if the last anchor was not on the page of this context.

Parameters:

`ctx` (`context`) – The context from which to start.

get-candidates(ctx: `context`) -> `candidates`

Get the element candidates for the given context.

Parameters:

`ctx` (`context`) – The context for which to get the candidates.

is-active-redundant(ctx: `context`, candidates: `candidates`) -> `bool`

Check if showing the active element would be redudnant in the current context.

Parameters:

`ctx` (`context`) – The context in which the redundancy of the previous primary candidate should be checked.

`candidates` (`candidates`) – The candidates for this context.

is-active-visible(ctx: `context`, candidates: `candidates`) -> `bool`

Checks if the previous primary candidate is still visible.

Parameters:

`ctx` (`context`) – The context in which the visibility of the previous primary candidate should be checked.

`candidates` (`candidates`) – The candidates for this context.

is-on-starting-page(ctx: `context`, candidates: `candidates`) -> `bool`

Checks if the current context is on a starting page, i.e. if the next candidates are on top of this context's page.

Parameters:

`ctx` (`context`) – The context in which the visibility of the next candidates should be checked.

`candidates` (`candidates`) – The candidates for this context.

`selectors` `stable`

Contians functions used for creating custom selectors.

- by-level()
- custom()
- sanitize()

by-level(min: `int` `none`, max: `int` `none`, ..exact: `int` `none`) -> `hydra-selector`

Create a heading selector for a given range of levels.

Parameters:

`min` (`int` or `none` = `none`) – The inclusive minimum level to consider as the primary heading

`max` (`int` or `none` = `none`) – The inclusive maximum level to consider as the primary heading

`..exact` (`int` or `none`) – The exact level to consider as the primary element

custom(element: `function` `selector`, filter: `function`, ancestors: `function` `selector`,
ancestors-filter: `function`) -> `hydra-selector`

Create a custom selector for `hydra`.

Parameters:

`element` (`function` or `selector`) – The primary element to search for.

`filter` (`function` = `none`) – The filter to apply to the element.

`ancestors` (`function` or `selector` = `none`) – The ancestor elements, this should match all of its ancestors.

`ancestors-filter` (`function` = `none`) – The filter applied to the ancestors.

sanitize(name: `str`, sel: `any`, message: `str` `auto`) -> `hydra-selector`

Turn a selector or function into a hydra selector.

This function is considered unstable.

Parameters:

`name` (`str`) – The name to use in the assertion message.

`sel` (`any`) – The selector to sanitize.

`message` (`str` or `auto` = `auto`) – The assertion message to use.

`util` `unstable`

Utlity functions and values.

`util/core` `unstable`

Utlity functions.

- or-default()

```
or-default(value: any, default: function, check: any) -> any
```

Substitute `value` for the return value of `default()` if it is a sentinel value.

Parameters:

`value` (`any`) – The value to check.

`default` (`function`) – The function to produce the default value with.

`check` (`any` = `none`) – The sentinel value to check for.

`util/assert` `unstable`

Assertions used for input and state validation.

- element()
- enum()
- queryable()
- types()

element(name: `str`, element: `any`, ..expected-funcs: `type`, message: `str` `auto`)

Assert that `element` is an element creatd by one of the given `expected-funcs`.

Parameters:

`name` (`str`) – The name use for the value in the assertion message.

`element` (`any`) – The value to check for.

`..expected-funcs` (`type`) – The expected element functions of `element`.

`message` (`str` or `auto` = `auto`) – The assertion message to use.

enum(name: `str`, value: `any`, ..expected-values: `type`, message: `str` `auto`)

Assert that `value` is any of the given `expected-values`.

Parameters:

`name` (`str`) – The name use for the value in the assertion message.

`value` (`any`) – The value to check for.

`..expected-values` (`type`) – The expected variants of `value`.

`message` (`str` or `auto` = `auto`) – The assertion message to use.

queryable(name: `str`, value: `any`, message: `str` `auto`)

Assert that `value` can be used in `query`.

Parameters:

`name` (`str`) – The name use for the value in the assertion message.

`value` (`any`) – The value to check for.

`message` (`str` or `auto` = `auto`) – The assertion message to use.

types(name: `str`, value: `any`, ..expected-types: `type`, message: `str` `auto`)

Assert that `value` is of any of the given `expected-types`.

Parameters:

`name` (`str`) – The name use for the value in the assertion message.

`value` (`any`) – The value to check for.

`..expected-types` (`type`) – The expected types of `value`.

`message` (`str` or `auto` = `auto`) – The assertion message to use.