<div focusgroup> 
{ toggle: ... } 
and tabs

Proposals for affordances for common User Interface design patterns. A TPAC breakout session discussion.

21 October 2021
12:00 AM UTC and 3:00 PM UTC
Agenda

• General overview
• focusgroup concepts
• toggle concepts
• Use case: Tabs
• Discussion
focus and
state
management improvements

• Fundamental concepts of UI and control design
• Both available in the platform, but not exposed as reusable concepts for general developer use
<div focusgroup>
Concepts

Detailed Explainer | Open UI presentation/overview
Concept: `TAB` stop redux

Faster webpage navigation for keyboard users.

Group related components of a control together with one TAB stop, use Arrow Keys to navigate between items in the group.
HTML focusgroup attribute

Authors: Travis Leithhead, David Zearing, Chris Holt

Status of this Document

This document is a starting point for engaging the community and standards bodies in developing collaborative solutions fit for standardization. As the solutions to problems described in this document progress along the standards-track, we will retain this document as an archive and use this section to keep the community up-to-date with the most current standards venue and content location of future work and discussions.

- This document status: Active
- Expected venue: WHATWG HTML Workstream
- Current version: this document

1. Introduction

When writing custom controls, authors need to implement the semantics of various known controls (see ARIA authoring guide) to enable proper (and expected) keyboard support. Control examples include tabs and tabsets, combo boxes, accordion panels, carousels, focussable grid tables, etc. Many of these patterns expect arrow-key navigation, as well as support for page down/up, home/end, even “type ahead” behavior.
HTML focusgroup attribute

Authors: Travis Leithead, David Zearing, Chris Holt

Status of this Document

This document is a starting point for engaging the community and standards bodies in developing collaborative solutions fit for standardization. As the solutions to problems described in this document progress along the standards-track, we will retain this document as an archive and use this section to keep the community up-to-date with the most current standards venue and content location of future work and discussions.

- This document status: Active
- Expected venue: WHATWG HTML Workstream
- Current version: this document

1. Introduction

When writing custom controls, authors need to implement the semantics of various known controls (see ARIA authoring guide) to enable proper (and expected) keyboard support. Control examples include tabs and tabsets, combo boxes, accordion panels, carousels, focusable grid tables, etc. Many of these patterns expect arrow-key navigation, as well as support for page down/up, home/end, even "type ahead" behavior.
Concept: navigate control groups

Logical Arrow Key navigation among related concepts (may or may not reduce TAB stops)
List of Edge URLs

- `edge://about`
Concept: accessibility affordance

Keyboard navigation with Arrow Keys (or equivalent) is required for accessibility conformance in common controls (built-in or otherwise)
Radio button groups

What would you like to drink?
- Coffee
- Tea
- Soda

(cycle forward with right/down; cycle backward with left/up)

Select menus

(customer
  - ayara
  - category
  - customer
  - Employee
  - Module
  - Module Access
  - Role)

(up/down to change focused option; home/end; type-ahead to search)

BUILT IN

Radio button groups

Select menus

Accordian

Data table (role=grid)

(role=menu)

(role=tablist)
Concept: focus vs state

Often combined, but separate concepts
Linked focus and state

Radio button groups

- As focus moves, so does `checked` state
Focus and state separated

• focus allows exploration/navigation; *activation* on focused control changes state
Focusgroup proposal *short summary*

- Enables existing focusable elements within a scope to respond to arrow keys, home/end, etc.
  - Does not change the focusability (or TAB order) of anything.
- Intended for content navigation (not layout navigation)
- Only deals with focus change (does not persist any state)
- Has various options: wrap, horizontal/vertical nesting, table content based navigation
{ toggle: ... }

Concepts

Proposal Draft
Concept: "toggleable" state

How to save your activated state
state=toggle
Built-in `<input type=checkbox>`

- Includes persistent toggle (checked)
- User can "activate" it with keyboard/mouse to change the state.
- Developers can leverage the toggle state (e.g., to adjust UI)
  - :checked pseudo class
  - DOM-accessible state
Toggle

- Add persistent toggle state
- User can "activate" it with keyboard/mouse to change the state.
- Developers can leverage the state (e.g., to adjust UI)
  - :toggle() pseudo class
  - DOM-accessible state (TBD)
Concept: non-boolean toggles

Support tri-level state toggles and beyond
0
1
2

- Checked
- Tristate / Indeterminate
- Unchecked
"sticky" cycling

2
1
0

- Checked
- Tristate / Indeterminate
- Unchecked
Concept: toggle scopes

Activation element is separable from the toggle-containing element
parent node

element with toggle

previous siblings

next siblings

Scope

User activation (e.g., click/enter key)

descendant node
parent node

previous siblings

element with toggle

next siblings

descendant node

User activation (e.g., click/enter key)
Concept: linked toggle groups

i.e., like radio button groups
node establishing group scope

related toggle elements are grouped under the scope
Concept: n-toggles per element

Elements not restricted to only one toggle—can have as many toggles as needed
element with five toggles
toggles differentiated by name
Toggle proposal short summary

• Grants toggleability to all elements
  • Toggles are named and have N states
  • Any number of toggles per element

• Toggle "triggers" (toggle-set) make elements activatable

• Named toggle groups link same-named toggles together under scope

• Toggles can be selected with :toggle(<name>)

• TODO
  • Provide toggle state names (currently numbers)
  • Provide DOM access to toggle state
tabs

One use case: pulling it all together
Showing off Spicy Sections

This demo shows different designed affordances at different screen sizes (resize your viewport and view the source too)...

- 0-30em: collapsible sections
- 30-60em: structured content
- 60em+: tab-bar

The tab-bar affordance can use shadow-dom to establish a styleable tab-bar and panels. The required parts can be exposed for styling with consistent pseudo-elements.

In a tab state, we likely want to style:

- The outer box
- The tab bar
- The tabs themselves
- The selected tab
- The tab panel area

https://codepen.io/bkardell/pen/VwpJGGL
The tab-bar affordance can use shadow-dom to establish a styleable tab-bar and panels. The required parts can be exposed for styling with consistent pseudo-elements.

In a tab state, we likely want to style:
The tab-bar affordance can use shadow-dom to establish a styleable tab-bar and panels. The required parts can be exposed for styling with consistent pseudo-elements.

In a tab state, we likely want to style:
Observation: focusgroup and toggle "scope"

The tab-bar affordance can use shadow-dom to establish a styleable tab-bar and panels. The required parts can be exposed for styling with consistent pseudo-elements.
Scenario Gap*: tab stop for toggled item(s)

The tab-bar affordance can use shadow-dom to establish a styleable tab-bar and panels. The required parts can be exposed for styling with consistent pseudo-elements.

In a tab state, we likely want to style:

```html
<!-- abbreviated markup... -->
<spicy-sections affordance="tab-bar">
  <h2 id="cp1" role="tab" aria-controls="cp2" tabindex="0">...</h2>
  <div id="cp2" role="tabpanel"></div>
  <h2 id="cp3" role="tab" aria-controls="cp4" tabindex="-1">...</h2>
  <div id="cp4" role="tabpanel"></div>
  <h2 id="cp5" role="tab" aria-controls="cp6" tabindex="-1">...</h2>
  <div id="cp6" role="tabpanel"></div>
</spicy-sections>
```

*Assumes a motivation to avoid JavaScript
Scenario Gap*: toggle-on-focus

The tab-bar affordance can use shadow-dom to establish a styleable tab-bar and panels. The required parts can be exposed for styling with consistent pseudo-elements.

In a tab state, we likely want to style:

*Assumes a motivation to avoid JavaScript
Solution rough ideas...

• *Tab stop for toggled items*: enable the active toggle (in a group) to participate in sequential keyboard navigation (inactive toggles wouldn't)
  • (replaces roving tabindex method)

• *Toggle-on-focus*: provide a configuration for a toggle to be activated by *focus* or by [explicit action/click/enter]
  • (enables linked focus vs unlinked toggle selection)
Other Toggle Scenarios (non-tab)

- Multi-selection groups? Toggle groups with N>1 maximum active toggles
  - Ex: (Survey) Chose 3 of the following...

- Select All/None linked Toggle
  - Ex:
CSS or HTML?

• Should focusgroup be exposed via CSS?
  • Grouping mechanisms could be aligned?
• toggle one-time creation in HTML?
  • toggle-create has unusual CSS semantics, may be better suited to HTML attribute?
Open Discussion