Package 'glasstabs'

May 23, 2026

Title Animated Glass-Style Tabs and Multi-Select Filter for 'Shiny'

Version 0.3.2

Description Tools for creating animated glassmorphism-style tab navigation and select filter widgets in 'Shiny' applications. Provides a tab navigation component with a sliding glass halo animation, a searchable multi-select dropdown, and a single-select dropdown — all with multiple colour themes and server-side update helpers. Tabs support icons, numeric badges, disable/enable toggling, runtime append/remove, reactive rendering via 'renderGlassTabs()', URL bookmarking, and compact mode for dashboard card layouts. 'glassTabCondition()' generates 'conditionalPanel()' condition strings without needing to recall the internal input key pattern. 'glasstabs_news()' displays the release notes from the R console. Built-in example apps can be launched with 'runGlassExample()'. All widgets are compatible with standard 'Shiny' layouts and 'bs4Dash' dashboards. For full documentation and examples see Arthur (2026) <https://prigasg.github.io/glasstabs/>.

License MIT + file LICENSE

URL https://github.com/PrigasG/glasstabs, https://prigasg.github.io/glasstabs/

BugReports https://github.com/PrigasG/glasstabs/issues

Imports htmltools (>= 0.5.0), shiny (>= 1.7.0)

Suggests bs4Dash, knitr, rmarkdown, spelling, testthat (>= 3.0.0), mockery

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.3

Language en-US

NeedsCompilation no

Author George Arthur [aut, cre]

Maintainer George Arthur <prigasgenthian48@gmail.com>

Repository CRAN

Date/Publication 2026-05-23 21:50:03 UTC

Contents

`appendGlassTab` *Append or remove a glass tab at runtime*

Description

`appendGlassTab()` adds a new `glassTabPanel()` to an existing `glassTabsUI()` at runtime. `removeGlassTab()` removes a tab by value. If the removed tab was active, the first remaining tab is activated.

Usage

```
appendGlassTab(session, id, tab, select = FALSE)

removeGlassTab(session, id, value)
```

Arguments

session	Shiny session object.
id	Module id matching the id passed to glassTabsUI().
tab	A glassTabPanel() object (for appendGlassTab() only).
select	Logical. If TRUE, the new tab is immediately activated. Defaults to FALSE.
value	Value of the tab to remove (for removeGlassTab() only).

Value

Called for its side effect; returns NULL invisibly.

Examples

```
if (interactive()) {
  library(shiny)
  ui <- fluidPage(
    useGlassTabs(),
    glassTabsUI(
      "tabs",
      glassTabPanel("home", "Home", p("Home content"), selected = TRUE)
    ),
    actionButton("add",    "Add tab"),
    actionButton("remove", "Remove tab")
  )
  server <- function(input, output, session) {
    observeEvent(input$add, {
      appendGlassTab(session, "tabs",
        glassTabPanel("new", "New Tab", p("Dynamic content")),
        select = TRUE
      )
    })
    observeEvent(input$remove, {
      removeGlassTab(session, "tabs", "new")
    })
  }
  shinyApp(ui, server)
}
```

---

disableGlassTab *Disable or enable a glass tab*

Description

disableGlassTab() grays out a tab and prevents the user from clicking it without removing it from the navigation bar. enableGlassTab() reverses this. Unlike hideGlassTab(), a disabled tab remains visible.

Usage

```
disableGlassTab(session, id, value)

enableGlassTab(session, id, value)
```

Arguments

session	Shiny session object.
id	Module id matching the id passed to glassTabsUI().
value	Value of the tab to disable or enable.

Value

Called for its side effect; returns NULL invisibly.

Examples

```
if (interactive()) {
  library(shiny)
  ui <- fluidPage(
    useGlassTabs(),
    glassTabsUI(
      "tabs",
      glassTabPanel("a", "A", p("Tab A"), selected = TRUE),
      glassTabPanel("b", "B", p("Tab B")),
      glassTabPanel("locked", "Locked", p("Locked content"))
    ),
    checkboxInput("unlocked", "Unlock tab", FALSE)
  )
  server <- function(input, output, session) {
    # Start with "locked" tab disabled
    observe({
      if (input$unlocked) enableGlassTab(session, "tabs", "locked")
      else                disableGlassTab(session, "tabs", "locked")
    })
  }
  shinyApp(ui, server)
}
```

---

glassFilterTags *Shiny tag helper for a filter-tags display area tied to a glassMultiSelect*

Description

Renders a that the JS engine will populate with colored tag pills whenever the corresponding glassMultiSelect() selection changes.

Usage

```
glassFilterTags(inputId, class = NULL)
```

Arguments

inputId	The inputId of the glassMultiSelect() this display should reflect.
class	Additional CSS classes for the container.

Value

An htmltools tag (shiny.tag). A container that renders the current selection of the paired glassMultiSelect() as dismissible pill tags, kept in sync with the widget automatically via JavaScript.

---

glassMultiSelect *Animated glass multi-select dropdown filter*

Description

A stylized multi-select Shiny input with optional search, style switching, select-all behavior, and programmatic updates via updateGlassMultiSelect().

Usage

```
glassMultiSelect(
  inputId,
  choices,
  selected = NULL,
  label = NULL,
  placeholder = "Filter by Category",
  all_label = "All categories",
  check_style = c("checkbox", "check-only", "filled"),
  show_style_switcher = TRUE,
  show_select_all = TRUE,
  show_clear_all = TRUE,
  theme = "dark",
  hues = NULL,
  dark_selector = NULL
)
```

Arguments

inputId	Shiny input id.
choices	Named or unnamed character vector of choices.
selected	Initially selected values. Defaults to all choices when NULL.
label	Optional field label shown above the widget.

placeholder	Trigger label when nothing is selected.
all_label	Label shown when all choices are selected.
check_style	One of "checkbox" (default), "check-only", or "filled".
show_style_switcher	Show the Check / Box / Fill switcher row inside the dropdown? Default TRUE.
show_select_all	Show the "Select all" row? Default TRUE.
show_clear_all	Show the "Clear all" footer link? Default TRUE.
theme	Color theme. One of "dark" (default) or "light", or a glass_select_theme() object.
hues	Optional named integer vector of HSL hue angles (0 to 360) for the "filled" style. Auto-assigned if NULL.
dark_selector	Optional CSS selector that signals dark mode (e.g. "body.dark-mode" for bs4Dash). When provided and theme = "light", emits an extra scoped <style> block that reverts colors to the dark-mode defaults whenever that selector is active.

Details

The widget registers two Shiny inputs:

- input$<inputId> : character vector of selected values
- input$<inputId>_style : active style string ("checkbox", "check-only", or "filled")

By default, when selected = NULL, all choices are initially selected. This preserves the existing package behavior.

Value

An htmltools::tagList containing the trigger button, dropdown panel, and scoped <style> block.

Examples

```
fruits <- c(Apple = "apple", Banana = "banana", Cherry = "cherry")

# Minimal
fruit_filter <- glassMultiSelect("f", fruits)

# Lock style, hide extra controls
locked_filter <- glassMultiSelect(
  "f",
  fruits,
  check_style = "check-only",
  show_style_switcher = FALSE,
  show_select_all = FALSE,
  show_clear_all = FALSE
)
```

```
# Light theme
light_filter <- glassMultiSelect("f", fruits, theme = "light")
```

`glassMultiSelectValue` *Reactive helpers for glassMultiSelect values*

Description

Convenience helper for extracting a multi-select widget's value and style from Shiny's `input` object without using modules.

Usage

```
glassMultiSelectValue(input, inputId)
```

Arguments

`input`	Shiny `input` object.
`inputId`	Input id used in `glassMultiSelect()`.

Value

A named list with two reactives:

`selected` Reactive character vector of selected values

`style` Reactive string for the active style

Examples

```
if (interactive()) {
  library(shiny)

  ui <- fluidPage(
    useGlassTabs(),
    glassMultiSelect("cats", c(A = "a", B = "b", C = "c"))
  )

  server <- function(input, output, session) {
    ms <- glassMultiSelectValue(input, "cats")
    observe({
      message("Selected: ", paste(ms$selected(), collapse = ", "))
      message("Style: ", ms$style())
    })
  }

  shinyApp(ui, server)
}
```

`glassSelect` *Animated glass single-select dropdown*

Description

A stylized single-select Shiny input with optional search, clear control, selection-marker styling, and programmatic updates via `updateGlassSelect()`.

Usage

```
glassSelect(
  inputId,
  choices,
  selected = NULL,
  label = NULL,
  placeholder = "Select an option",
  searchable = TRUE,
  clearable = FALSE,
  include_all = FALSE,
  all_choice_label = "All categories",
  all_choice_value = "__all__",
  check_style = c("checkbox", "check-only", "filled"),
  theme = "dark"
)
```

Arguments

`inputId`	Shiny input id.
`choices`	Named or unnamed character vector of choices.
`selected`	Initially selected value. Defaults to `NULL`.
`label`	Optional field label shown above the widget.
`placeholder`	Trigger label when nothing is selected.
`searchable`	Logical. Show search input inside dropdown? Default `TRUE`.
`clearable`	Logical. Show clear control for removing the current selection? Default `FALSE`.
`include_all`	Logical. Prepend an explicit "All" option. Default `FALSE`.
`all_choice_label`	Label used for the explicit "All" option.
`all_choice_value`	Value used for the explicit "All" option.
`check_style`	One of `"checkbox"` (default), `"check-only"`, or `"filled"`.
`theme`	Color theme. One of `"dark"` (default) or `"light"`, or a `glass_select_theme()` object.

Details

The widget registers one Shiny input:

- `input$<inputId>` : selected value as a length-1 character string, or `NULL` when nothing is selected

Value

An `htmltools::tagList` containing the single-select trigger, dropdown panel, and scoped `<style>` block.

Examples

```
fruits <- c(Apple = "apple", Banana = "banana", Cherry = "cherry")

fruit_select <- glassSelect("fruit", fruits)

selected_fruit <- glassSelect(
  "fruit",
  fruits,
  selected = "banana",
  clearable = TRUE
)

all_fruits <- glassSelect(
  "fruit",
  fruits,
  include_all = TRUE,
  all_choice_label = "All fruits",
  all_choice_value = "__all__"
)

filled_fruit <- glassSelect(
  "fruit",
  fruits,
  check_style = "filled"
)
```

---

glassSelectValue *Reactive helper for glassSelect values*

Description

Convenience helper for extracting a single-select widget's value from Shiny's `input` object without using modules.

Usage

```
glassSelectValue(input, inputId)
```

Arguments

input	Shiny input object.
inputId	Input id used in glassSelect().

Value

A reactive expression returning the current selected value as a character scalar, or NULL when nothing is selected.

Examples

```
if (interactive()) {
  library(shiny)

  ui <- fluidPage(
    useGlassTabs(),
    glassSelect("fruit", c(Apple = "apple", Banana = "banana"))
  )

  server <- function(input, output, session) {
    fruit <- glassSelectValue(input, "fruit")
    observe({
      print(fruit())
    })
  }

  shinyApp(ui, server)
}
```

---

glassTabCondition *Build a conditionalPanel condition for a glasstabs widget*

Description

Returns a JavaScript condition string that evaluates to TRUE when the specified glasstabs widget has a given active tab. Pass the result directly to the condition argument of shiny::conditionalPanel().

Usage

```
glassTabCondition(id, value)
```

Arguments

id	The id passed to glassTabsUI(). **Inside a Shiny module** use ns("tabs") here (same id you passed to glassTabsUI()), NOT the bare id you pass to glassTabsServer().
value	The tab value string (the value argument of the target glassTabPanel()) that should trigger the condition.

Value

A single character string for use in shiny::conditionalPanel().

Examples

```
# Returns a plain JS condition string — no Shiny session needed:
glassTabCondition("main", "details")

# Inside a module — use ns() for the id:
# UI:   glassTabCondition(ns("tabs"), "details")
# This produces: "input['mymod-tabs-active_tab'] === 'details'"

# Full app example showing conditionalPanel usage:
if (interactive()) {
  library(shiny)
  ui <- fluidPage(
    useGlassTabs(),
    glassTabsUI(
      "main",
      glassTabPanel("overview", "Overview", selected = TRUE,
        p("Always visible.")),
      glassTabPanel("details", "Details",
        p("Detail pane."))
    ),
    conditionalPanel(
      condition = glassTabCondition("main", "details"),
      wellPanel("This panel only shows on the Details tab.")
    )
  )
  server <- function(input, output, session) {}
  shinyApp(ui, server)
}
```

---

glassTabPanel *Define a single glass tab panel*

Description

Used as child arguments inside glassTabsUI(). Each call defines one tab button and its associated content pane.

Usage

```
glassTabPanel(value, label, ..., icon = NULL, selected = FALSE)
```

Arguments

value	A unique string identifier for this tab (e.g. "A").
label	The text shown on the tab button.
...	UI elements for the pane content.
icon	Optional icon shown to the left of the tab label. Accepts any htmltools-compatible tag, e.g. shiny::icon("table") or fontawesome::fa("house"). Pass NULL (default) for no icon.
selected	Logical. Whether this tab starts selected. Only the first selected = TRUE tab takes effect; defaults to FALSE.

Value

A list of class "glassTabPanel" consumed by glassTabsUI().

Examples

```
# Plain text label
overview_tab <- glassTabPanel("overview", "Overview",
  shiny::h3("Welcome"),
  shiny::p("This is the overview tab.")
)

# With a Shiny icon
data_tab <- glassTabPanel("data", "Data",
  icon = shiny::icon("table"),
  shiny::p("Data content here.")
)
```

---

glassTabsOutput *Dynamic glass tab UI output*

Description

A drop-in replacement for shiny::uiOutput() that pairs with renderGlassTabs(). It creates a placeholder that Shiny fills with a fully reactive glassTabsUI() when the server-side render function runs. The JavaScript engine is automatically (re-)initialised after each render.

Usage

```
glassTabsOutput(outputId, ...)
```

Arguments

outputId	The output id used in the paired renderGlassTabs() call.
...	Additional arguments forwarded to shiny::uiOutput().

Value

A shiny.tag suitable for use in a Shiny UI.

Examples

```
# Creates a UI placeholder tag — no Shiny session needed:
tabs_placeholder <- glassTabsOutput("my_tabs")

# Full dynamic-tab app example:
if (interactive()) {
  library(shiny)

  tab_data <- list(
    list(value = "a", label = "Alpha"),
    list(value = "b", label = "Beta"),
    list(value = "c", label = "Gamma")
  )

  ui <- fluidPage(
    useGlassTabs(),
    selectInput("n", "Show tabs", choices = 2:3, selected = 2),
    glassTabsOutput("dynamic_tabs")
  )

  server <- function(input, output, session) {
    output$dynamic_tabs <- renderGlassTabs({
      panels <- lapply(
        head(tab_data, as.integer(input$n)),
        function(t) glassTabPanel(t$value, t$label, p(t$label))
      )
      do.call(glassTabsUI, c(list("dyn"), panels))
    })
  }

  shinyApp(ui, server)
}
```

---

glassTabsServer *Server logic for glass tabs*

Description

Tracks the active tab and exposes it as a reactive value. Optionally integrates with Shiny's bookmarking system so the active tab is preserved in bookmarked URLs.

Usage

```
glassTabsServer(id, bookmark = TRUE)
```

Arguments

id	Module id matching the id passed to glassTabsUI(). Do **not** wrap this in ns() — glassTabsServer() handles namespacing internally via shiny::moduleServer().
bookmark	Logical. When TRUE (default), registers shiny::onBookmark() and shiny::onRestored() hooks so the active tab is saved and restored automatically when Shiny bookmarking is enabled. Set to FALSE to opt out.

Details

glassTabsServer() follows the same calling convention as all Shiny module server functions: pass the **bare** module id, not a namespaced one. Inside a parent module, pair it with glassTabsUI(ns("tabs"), ...) in the UI, and call glassTabsServer("tabs") (without ns()) in the server.

Value

A reactive expression returning the active tab value.

Examples

```
# --- Standalone app ---
if (interactive()) {
  library(shiny)
  ui <- fluidPage(
    useGlassTabs(),
    glassTabsUI(
      "tabs",
      glassTabPanel("a", "A", p("Tab A"), selected = TRUE),
      glassTabPanel("b", "B", p("Tab B"))
    )
  )
  server <- function(input, output, session) {
    active <- glassTabsServer("tabs")
    observe(print(active()))
  }
  shinyApp(ui, server)
}

# --- Bookmarking ---
if (interactive()) {
  library(shiny)
  ui <- function(request) {
    fluidPage(
      useGlassTabs(),
      bookmarkButton(),
      glassTabsUI(
        "tabs",
        glassTabPanel("a", "A", p("Tab A"), selected = TRUE),
        glassTabPanel("b", "B", p("Tab B"))
      )
    )
  }
```

```
  server <- function(input, output, session) {
    active <- glassTabsServer("tabs", bookmark = TRUE)
  }
  shinyApp(ui, server, enableBookmarking = "url")
}

# --- Inside a Shiny module ---
# UI side: use ns() to namespace the widget id
my_module_ui <- function(id) {
  ns <- shiny::NS(id)
  shiny::tagList(
    useGlassTabs(),
    glassTabsUI(
      ns("tabs"),                          # <-- ns() wraps the id here
      glassTabPanel("a", "A", p("Tab A"), selected = TRUE),
      glassTabPanel("b", "B", p("Tab B"))
    )
  )
}

# Server side: pass the bare id — NOT ns("tabs")
my_module_server <- function(id) {
  shiny::moduleServer(id, function(input, output, session) {
    active <- glassTabsServer("tabs")     # <-- bare id, no ns()
    shiny::observe(print(active()))
  })
}
```

`glassTabsUI` *Animated glass-style tab navigation UI*

Description

Animated glass-style tab navigation UI

Usage

```
glassTabsUI(
  id,
  ...,
  selected = NULL,
  wrap = TRUE,
  compact = FALSE,
  extra_ui = NULL,
  theme = NULL,
  dark_selector = NULL
)
```

Arguments

id	Module namespace id.
...	One or more glassTabPanel() objects.
selected	Value of the initially selected tab.
wrap	Logical. When TRUE wraps everything in a div.gt-container.
compact	Logical. When TRUE applies reduced padding and spacing via the .gt-compact CSS modifier — useful inside dashboard cards or tight layouts (e.g. bs4Dash).
extra_ui	Optional additional UI placed to the right of the tab bar.
theme	One of "dark", "light", or a glass_tab_theme() object.
dark_selector	Optional CSS selector for a parent element that signals dark mode (e.g. "body.dark-mode" for bs4Dash, "[data-bs-theme=dark]" for Bootstrap 5). When provided and theme = "light", a second scoped <style> block overrides the CSS variables back to the dark-mode defaults whenever that selector is active — so the tabs stay readable after a dark-mode toggle without any server-side intervention.

Value

An htmltools::tagList ready to use in a Shiny UI.

---

glasstabs_news *Display the glasstabs changelog*

Description

Prints the package NEWS to the R console. Useful for quickly checking what changed between versions without leaving your R session.

Usage

```
glasstabs_news()
```

Value

Called for its side effect; returns NULL invisibly.

Examples

```
if (interactive()) {
  glasstabs_news()
}
```

glass_select_theme *Create a custom color theme for glass select widgets*

Description

All color arguments accept any valid CSS color string (hex, `rgb()`, `rgba()`, named colors). Unset fields inherit from the `mode` base preset.

Usage

```
glass_select_theme(
  mode = c("dark", "light"),
  bg_color = NULL,
  border_color = NULL,
  text_color = NULL,
  accent_color = NULL,
  label_color = NULL
)
```

Arguments

`mode`	Base preset. One of `"dark"` (default) or `"light"`. Custom colors are layered on top.
`bg_color`	Background of the trigger button and dropdown panel.
`border_color`	Border color of the trigger and dropdown.
`text_color`	Main text color for options and the trigger label.
`accent_color`	Highlight color for checkmarks, badges, and selected states. Also used for the focus ring.
`label_color`	Widget label color. Defaults to `text_color` when `NULL`.

Value

A named list of class `"glass_select_theme"` for passing to the `theme` argument of `glassMultiSelect()` or `glassSelect()`.

Examples

```
# Teal accent on a dark base
teal_theme <- glass_select_theme(
  mode         = "dark",
  accent_color = "#2dd4bf",
  bg_color     = "rgba(9, 20, 42, 0.97)"
)

# Light mode with a custom purple accent
purple_light <- glass_select_theme(
  mode         = "light",
```

```
  accent_color = "#7c3aed",
  border_color = "rgba(124, 58, 237, 0.35)"
)

if (interactive()) {
  library(shiny)
  choices <- c(Revenue = "rev", Orders = "ord", Returns = "ret")
  ui <- fluidPage(
    useGlassTabs(),
    glassMultiSelect("metric", choices, theme = teal_theme),
    glassSelect("region", c(All = "all", North = "n", South = "s"),
                theme = purple_light)
  )
  server <- function(input, output, session) {}
  shinyApp(ui, server)
}
```

`glass_tab_theme` *Create a custom color theme for glassTabsUI*

Description

All arguments accept any valid CSS color string (hex, `rgb()`, `rgba()`, named colors). Pass only the fields you want to override — unset fields fall back to the dark-mode defaults.

Usage

```
glass_tab_theme(
  tab_text = NULL,
  tab_active_text = NULL,
  halo_bg = NULL,
  halo_border = NULL,
  content_bg = NULL,
  content_border = NULL,
  card_bg = NULL,
  card_text = NULL
)
```

Arguments

`tab_text`	Inactive tab text color.
`tab_active_text`	Active tab text color (and headings inside cards).
`halo_bg`	Background fill of the animated glass halo.
`halo_border`	Border color of the glass halo.
`content_bg`	Tab content area background.

content_border	Tab content area border.
card_bg	Inner .gt-card background.
card_text	Inner .gt-card text color.

Value

A named list of class "glass_tab_theme" for passing to the theme argument of glassTabsUI().

Note

Light mode color accessibility: When building a light-mode theme, ensure tab_text is dark enough to read on a white background (e.g. at least "#374151") and tab_active_text provides strong contrast (e.g. "#1d4ed8" or darker). Light-grey or near-white values that look fine on dark backgrounds become invisible on light ones.

Examples

```
# Amber / warm accent on a dark base
amber <- glass_tab_theme(
  halo_bg         = "rgba(251, 191, 36, 0.15)",
  halo_border     = "rgba(251, 191, 36, 0.40)",
  tab_active_text = "#fef3c7"
)

if (interactive()) {
  library(shiny)
  ui <- fluidPage(
    useGlassTabs(),
    glassTabsUI(
      "demo",
      glassTabPanel("a", "Alpha", selected = TRUE, p("Alpha content")),
      glassTabPanel("b", "Beta",  p("Beta content")),
      theme = amber
    )
  )
  server <- function(input, output, session) {}
  shinyApp(ui, server)
}
```

---

renderGlassTabs *Render a reactive glass tab UI*

Description

Server-side render function that pairs with glassTabsOutput(). The expression should return a glassTabsUI() call. After each render the glasstabs JavaScript engine is automatically reinitialised so animations and event handlers are correctly attached to the new DOM nodes.

Usage

```
renderGlassTabs(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

expr	An expression that returns a `glassTabsUI()` tag object.
env	The environment in which to evaluate `expr`.
quoted	Logical. Whether `expr` is already quoted.

Value

A render function suitable for assigning to an `output` slot.

Examples

```
if (interactive()) {
  library(shiny)

  ui <- fluidPage(
    useGlassTabs(),
    radioButtons("theme", "Theme", c("dark", "light"), inline = TRUE),
    glassTabsOutput("tabs_out")
  )

  server <- function(input, output, session) {
    output$tabs_out <- renderGlassTabs({
      glassTabsUI(
        "themed",
        glassTabPanel("x", "X", selected = TRUE, p("X content")),
        glassTabPanel("y", "Y", p("Y content")),
        theme = input$theme
      )
    })
  }

  shinyApp(ui, server)
}
```

---

runGlassExample *Run a built-in glasstabs example app*

Description

Launches one of the example Shiny apps that ship with the package. A list of available examples is printed when called with no arguments. Example apps are launched only in interactive sessions.

Usage

```
runGlassExample(example = NULL, ...)
```

Arguments

example	Name of the example to run. One of "smoke-test", "basic", "bs4dash", "dashboard". When NULL (default), lists all available examples.
...	Additional arguments passed to shiny::runApp().

Value

Called for its side-effect (launches a Shiny app).

Examples

```
# List available examples
runGlassExample()

# Run an example interactively
if (interactive()) {
  runGlassExample("smoke-test")
}
```

---

showGlassTab *Show or hide a glass tab*

Description

showGlassTab() makes a hidden tab visible again. hideGlassTab() hides a tab from the navigation bar. If the hidden tab is currently active, the first remaining visible tab is activated automatically.

Usage

```
showGlassTab(session, id, value)

hideGlassTab(session, id, value)
```

Arguments

session	Shiny session object.
id	Module id matching the id passed to glassTabsUI().
value	Value of the tab to show or hide.

Value

Called for its side effect; returns NULL invisibly.

Examples

```
if (interactive()) {
  library(shiny)
  ui <- fluidPage(
    useGlassTabs(),
    glassTabsUI(
      "tabs",
      glassTabPanel("a", "A", p("Tab A"), selected = TRUE),
      glassTabPanel("b", "B", p("Tab B")),
      glassTabPanel("admin", "Admin", p("Admin only"))
    ),
    checkboxInput("is_admin", "Admin mode", FALSE)
  )
  server <- function(input, output, session) {
    observeEvent(input$is_admin, {
      if (input$is_admin) showGlassTab(session, "tabs", "admin")
      else                hideGlassTab(session, "tabs", "admin")
    }, ignoreInit = FALSE)
  }
  shinyApp(ui, server)
}
```

---

updateGlassMultiSelect

Update a glassMultiSelect widget

Description

Update the available choices and/or current selection of an existing `glassMultiSelect()` input.

Usage

```
updateGlassMultiSelect(
  session,
  inputId,
  choices = NULL,
  selected = NULL,
  check_style = NULL
)
```

Arguments

`session`	Shiny session.
`inputId`	Input id of the widget.
`choices`	New choices, or `NULL` to keep current choices.
`selected`	New selected values, or `NULL` to keep current selection. Use `character(0)` to clear.
`check_style`	Optional new style string. One of `"checkbox"`, `"check-only"`, or `"filled"`. Defaults to `NULL`, which keeps the current style unchanged.

Details

This function now follows Shiny-style update semantics more closely:

- `choices = NULL` leaves choices unchanged
- `selected = NULL` leaves selection unchanged
- `selected = character(0)` clears the selection

When `choices` is supplied and `selected` is not, the browser side keeps the intersection of the current selection and the new set of choices.

Value

No return value. Called for its side effect of updating the client-side widget.

`updateGlassSelect` *Update a glassSelect widget*

Description

Update the available choices, current selection, and/or selection-marker style of an existing `glassSelect()` input.

Usage

```
updateGlassSelect(
  session,
  inputId,
  choices = NULL,
  selected = NULL,
  check_style = NULL
)
```

Arguments

`session`	Shiny session.
`inputId`	Input id of the widget.
`choices`	New choices, or `NULL` to keep current choices.
`selected`	New selected value, or `NULL` to keep the current selection. Use `character(0)` to clear.
`check_style`	Optional new style string. One of `"checkbox"`, `"check-only"`, or `"filled"`. Defaults to `NULL`, which keeps the current style unchanged.

Details

This function follows Shiny-style update semantics:

- `choices = NULL` leaves choices unchanged
- `selected = NULL` leaves selection unchanged
- `selected = character(0)` clears the selection
- `check_style = NULL` leaves the current style unchanged

When `choices` is supplied and `selected` is not, the browser side keeps the current selection if it is still present in the new choices.

Value

No return value. Called for its side effect of updating the client-side widget.

`updateGlassTabBadge` *Update the badge count on a glass tab*

Description

Adds or updates a small numeric badge on a tab button — useful for surfacing counts such as unread items, pending rows, or notification totals. Set `count` to `0` or `NA` to hide the badge.

Usage

```
updateGlassTabBadge(session, id, value, count)
```

Arguments

`session`	Shiny session object.
`id`	Module id matching the `id` passed to `glassTabsUI()`.
`value`	Value of the tab to update.
`count`	Integer count to display. Values above 99 are shown as `"99+"`. `0` or `NA` hides the badge.

Value

Called for its side effect; returns `NULL` invisibly.

Examples

```
if (interactive()) {
  library(shiny)
  ui <- fluidPage(
    useGlassTabs(),
    glassTabsUI(
      "tabs",
      glassTabPanel("inbox",  "Inbox",  p("Messages here"), selected = TRUE),
      glassTabPanel("sent",   "Sent",   p("Sent items")),
      glassTabPanel("drafts", "Drafts", p("Draft items"))
    ),
    actionButton("refresh", "Refresh counts")
  )
  server <- function(input, output, session) {
    observeEvent(input$refresh, {
      updateGlassTabBadge(session, "tabs", "inbox",  count = sample(1:20, 1))
      updateGlassTabBadge(session, "tabs", "drafts", count = sample(0:5, 1))
    })
  }
  shinyApp(ui, server)
}
```

---

updateGlassTabsUI *Programmatically switch the active glass tab*

Description

Server-side equivalent of Shiny's `updateTabsetPanel()`. Sends a message to the browser to animate the tab switch just as if the user had clicked the tab button.

Usage

```
updateGlassTabsUI(session, id, selected)
```

Arguments

`session`	Shiny session object.
`id`	Module id matching the `id` passed to `glassTabsUI()`.
`selected`	Value of the tab to activate.

Value

Called for its side effect; returns `NULL` invisibly.

Examples

```
if (interactive()) {
  library(shiny)
  ui <- fluidPage(
    useGlassTabs(),
    glassTabsUI(
      "tabs",
      glassTabPanel("a", "A", p("Tab A"), selected = TRUE),
      glassTabPanel("b", "B", p("Tab B"))
    ),
    actionButton("go", "Go to B")
  )
  server <- function(input, output, session) {
    observeEvent(input$go, {
      updateGlassTabsUI(session, "tabs", selected = "b")
    })
  }
  shinyApp(ui, server)
}
```

`useGlassTabs` *Attach glasstabs CSS and JS dependencies*

Description

Call this once in your UI — either inside `fluidPage()`, `bs4DashPage()`, or any other Shiny page wrapper. It injects the required CSS and JS as proper `htmltools` dependencies so they are deduplicated automatically.

Usage

```
useGlassTabs()
```

Value

An `htmltools::htmlDependency` object (invisible to the user, consumed by Shiny's renderer).

Examples

```
# Returns an htmlDependency object — no Shiny session needed:
deps <- useGlassTabs()

# Typical usage inside a Shiny UI:
if (interactive()) {
  library(shiny)
  ui <- fluidPage(
    useGlassTabs(),
    glassTabsUI("demo",
      glassTabPanel("A", "Tab A", p("Content A")),
```

```
      glassTabPanel("B", "Tab B", p("Content B"))
    )
  )
  server <- function(input, output, session) {}
  shinyApp(ui, server)
}
```

Index