Package: bslibreact (via r-universe)

November 15, 2026

Title Bootstrap Layout for 'shinyreact' Apps

Version 0.1.0

Description R-side helpers for building Shiny apps in the shinyreact 'ui.tsx' pattern that look like 'bslib' apps. Provides page dependency discovery so a client bundle can request the same CSS and JavaScript a 'bslib' page would attach on its own, and a hosted-input helper for widgets rendered server-side and displayed by the client. The matching React components live in the 'bslibreact' npm package.

License MIT + file LICENSE

URL https://www.samuelbharti.com/bslibreact/r, https://github.com/samuelbharti/bslibreact

BugReports https://github.com/samuelbharti/bslibreact/issues

Imports bslib (>= 0.9.0), cli, htmltools (>= 0.5.8), rlang (>= 1.1.0), shiny (>= 1.10.0), stats

Suggests jsonlite, shinyreact, spelling, testthat (>= 3.0.0), withr

Remotes github::posit-dev/shinyreact/pkg-r@r/v0.1.0

Config/roxygen2/version 8.1.1

Config/testthat/edition 3

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

Config/pak/sysreqs cmake make libuv1-dev zlib1g-dev

Repository https://samuelbharti.r-universe.dev

Date/Publication 2026-11-15 13:19:46 UTC

RemoteUrl https://github.com/samuelbharti/bslibreact

RemoteRef HEAD

RemoteSha f0f921672c02f6b06411665f432e91b5b0337df2

RemoteSubdir pkg-r

Contents

bslibreact_dep 2
host_input 3
page_deps 4

Index **6**

`bslibreact_dep` *The vendored bslibreact.js dependency*

Description

Returns the `htmltools::htmlDependency()` for the vendored `bslibreact.js` bundle (and its stylesheet, when one is built), which sets `window.bslibreact`. Attach it only for a client with no build step that reads the kit from that global, for example `page_react(page_deps(app_theme), bslibreact_dep(), theme = app_theme)`. A client built with Vite bundles the kit itself and does not need it.

Usage

```
bslibreact_dep()
```

Value

An `htmltools::htmlDependency()`.

Load order

The bundle reads `window.shinyreact` as it evaluates, so it must run after shinyreact's own bundle. `shinyreact::page_react()` attaches that first, and both scripts use `defer`, which keeps document order. The app's own `ui.js` also runs before this bundle, so a client script reads `window.bslibreact` inside a `window` `load` listener.

Before `pkg-js` has built anything

This function never errors for a missing bundle file: an `htmltools::htmlDependency()` does not check that its files exist when it is constructed, only once something actually tries to serve them, so this returns a structurally valid dependency even when `inst/lib/bslibreact/bslibreact.js` has not been vendored in yet. This matters during this repository's own development, where `pkg-js`'s build and `pkg-r`'s own checks can run before `make update-dist` has ever populated `inst/lib/bslibreact/`.

Examples

```
dep <- bslibreact_dep()
dep$name
dep$script
```

host_input *Host a server-rendered widget for a shinyreact client*

Description

Registers `widget` as a `shiny::renderUI()` output named `paste0(id, suffix)`, the way `shinyreact::page_react()` apps host a bslib or shiny widget the client displays with `<ShinyOutput id="<id><suffix>" />` (or the matching kit component, `HostedInput`, from the `bslibreact` npm package) instead of rebuilding the widget's markup in React. See `docs/decisions/0003-hosted-widgets.md` in the repository for the reasoning behind hosting widgets this way instead of driving them from React state.

Usage

```
host_input(output, id, widget, suffix = "_ui")
```

Arguments

`output`	The `output` object from a Shiny server function.
`id`	A short, stable identifier for this hosted widget. This is not itself a DOM id: it is combined with `suffix` to make the output name, so `id` only has to be distinct from any other `host_input()` call's `id` in the same app, and independent of whatever input id the widget itself declares.
`widget`	An expression that builds the widget, evaluated lazily inside `shiny::renderUI()` each time it is invalidated (so it can read reactive values, the same as any `renderUI()` block). Typically a single unevaluated widget call, for example `shiny::sliderInput("bins", "Bins", 1, 50, 30)`.
`suffix`	Appended to `id` to make the output name. With the default, `"_ui"`, `host_input(output, "bins", ...)` registers `output$bins_ui`, and the client shows it with `<SliderInput id="bins_ui" />` or any other `HostedInput`: the client always gets the full output name, suffix included.

Value

The output name (`paste0(id, suffix)`), invisibly.

Why the suffix matters

A hosted widget contributes two elements to the page, not one: the output wrapper Shiny creates for `output[[paste0(id, suffix)]]` (a `<div id="foo_ui" class="shiny-html-output">`, in the default case), and the widget's own element, with the widget's own input id, nested inside it (a `<div id="foo" class="form-group shiny-input-container">`, for a `shiny::sliderInput("foo", ...)`). These are two different ids on purpose: the wrapper is what the client's `HostedInput` component addresses to swap in new markup, and the widget's own id is what Shiny's input binding addresses to read `input$foo`. If the two ever match (a widget whose own id is `"foo_ui"`, or `suffix = ""`), the page holds two elements under one id; `document.getElementById()`, Shiny's

own `$("#foo")` lookups, and every downstream consumer of that id become ambiguous, silently addressing whichever one the browser happens to return first. `check_no_id_collision()` (an internal helper) catches this at render time with a message naming the id involved, instead of leaving it to surface later as an input or output that mysteriously never updates.

The output keeps rendering while it is hidden (`suspendWhenHidden = FALSE`), so a widget inside a closed panel or an inactive tab exists before it is first shown, as it would in the server-rendered app.

Examples

```
## Not run:
server <- function(input, output, session) {
  host_input(output, "bins", shiny::sliderInput("bins", "Bins", 1, 50, 30))
  # The client displays this with <ShinyOutput id="bins_ui" />, and reads
  # the slider's own value the usual way, from input$bins.
}

## End(Not run)
```

---

page_deps *Collect the page dependencies a client bundle needs*

Description

Renders a representative sample of bslib and shiny markup under `theme` and returns the `htmltools::htmlDependency()` objects that markup attaches, deduplicated, minus the `"bootstrap"` and `"jquery"` dependencies the page function attaches itself (`page_react()` attaches Bootstrap only when it is given a `theme`). Attach the result to a shinyreact page, for example `shinyreact::page_react(page_deps(app_theme), theme = app_theme)`, so the client gets exactly the CSS and JavaScript a matching bslib page would have attached on its own: `sliderInput()`'s ionRangeSlider skin, `selectizeInput()`'s selectize assets, `bslib::accordion()`'s and `bslib::tooltip()`'s Bootstrap bundles, and so on.

Usage

```
page_deps(theme = NULL, ..., components = NULL)
```

Arguments

`theme`	A `bslib::bs_theme()` object, or `NULL` for an app with no bslib theme. Pass the same theme the page gets. When not `NULL`, it is validated with `bslib::is_bs_theme()`.
`...`	Additional tags to include in the dependency sample, unnamed (checked with `rlang::check_dots_unnamed()`). Use this for anything the default `components` sampler does not cover: a hosted third party widget (`DT::DTOutput()`, `reactable::reactableOutput()`, an icon pack, or a page-level element specific to one app.
`components`	Character vector of catalog sample names to render for their dependencies, or `NULL` (the default) to use every `"tag"` and `"page"` sample, which is deliberately generous: an unused dependency costs the page one extra download, while a missing one is a widget that renders unstyled. Pass `character()` to include no catalog samples at all, relying only on `...`.

Details

"bootstrap" itself is dropped from the result because a shinyreact page already attaches its own theme through the page function's own theme= argument (or gets no theme at all); attaching it twice serves the compiled Bootstrap CSS from two dependency entries pointed at two separately compiled copies, which is wasted bytes at best and, if the two copies were compiled from different themes, a page that looks different depending on which <link> happens to load last.

Value

A list of htmltools::htmlDependency() objects, one per dependency name, keeping the highest version of each, and with no dependency named "bootstrap" or "jquery".

The theme

theme, when supplied, is set with bslib::bs_global_set() for the duration of this call and restored on exit. Several bslib widgets, sliderInput()'s ionRangeSlider skin among them, resolve their CSS through bslib::bs_dependency_defer(), which reads the globally active theme at render time. Outside a running Shiny session, which is where this function usually runs, that would be bslib's stock theme.

Set a global theme with bs_global_set(), not bslib::bs_global_theme(): its first argument is a Bootstrap version number, so passing it a theme object builds a different, stock theme and drops custom fonts.

Examples

```
deps <- page_deps(bslib::bs_theme(version = 5))
length(deps)

# A themeless app still gets the widget dependencies it needs (the
# ionRangeSlider skin, selectize assets, and so on), just with no
# bslib/Bootstrap theme compiled in.
length(page_deps(theme = NULL)) > 0

## Not run:
# A real app, with a client built by Vite:
app_theme <- bslib::bs_theme(version = 5)
ui <- shinyreact::page_react(
  page_deps(app_theme),
  title = "Demo",
  theme = app_theme
)

## End(Not run)
```

Index

bslib::bs_dependency_defer(), *5*
bslib::bs_global_set(), *5*
bslib::bs_global_theme(), *5*
bslib::bs_theme(), *4*
bslib::is_bs_theme(), *4*
bslibreact_dep, 2

host_input, 3
htmltools::htmlDependency(), *2*, *4*, *5*

page_deps, 4

rlang::check_dots_unnamed(), *4*

shiny::renderUI(), *3*
shinyreact::page_react(), *3*