Package: kitDesign (via r-universe)

November 1, 2026

Title Shared 'Shiny' User Interface Components for KIT Research Tools

Version 0.7.0

Date 2026-11-01

Description The shared user interface layer of the KIT research tools built with 'shiny': a 'bslib' theme based on Bootswatch Brite, the KIT style sheets and logos, a page wrapper with the KIT header, and documented components (page guide, buttons, download buttons, callouts, cloud-version banner and footer, status boxes, details rows, code panels, step cards, checklists, step tabs, diagrams, tables and a 'plotly' chart style), a skeleton for tabs with one run button (page order and status flow), plus a few server-side helpers (run gate, scroll to result, tab states, notifications, number formatting, user-facing errors), value objects for status boxes, checklists and code, 'shiny' modules for the stateful pieces (step cards, step tabs, run gate, code panel) and a scaffold for new apps. Load it with one call to kit_page() and build the page from kit_*() components. The logo and the colour palette come from a brand object (kit_brand()), so projects other than KIT can use their own.

License MIT + file LICENSE

URL https://github.com/jiewenTsai/kitDesign, https://jiewentsai.github.io/kitDesign/

BugReports https://github.com/jiewenTsai/kitDesign/issues

Encoding UTF-8

Depends R (>= 4.1.0)

Imports bslib (>= 0.11.0), grDevices, htmltools, shiny, utils

Suggests DT, htmlwidgets, knitr, pkgload, plotly, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Roxygen list(markdown = TRUE)
Config/roxygen2/version 8.1.0
Config/pak/sysreqs cmake make libuv1-dev zlib1g-dev
Repository https://jiewentsai.r-universe.dev
Date/Publication 2026-11-01 05:29:10 UTC
RemoteUrl https://github.com/jiewenTsai/kitDesign
RemoteRef HEAD
RemoteSha ab535824a89fda3a68a7f016577d53aa71237166

Contents

kit_abort . . . 4
kit_advanced . . . 4
kit_app_description . . . 5
kit_app_link . . . 6
kit_app_meta . . . 7
kit_brand . . . 7
kit_btn_class . . . 9
kit_button . . . 10
kit_callout . . . 11
kit_chart_colors . . . 11
kit_chart_neutral . . . 12
kit_chart_sequential . . . 12
kit_check . . . 13
kit_checklist . . . 14
kit_cloud_banner . . . 15
kit_cloud_footer . . . 16
kit_code . . . 17
kit_code_blocks . . . 18
kit_code_cols . . . 18
kit_code_panel . . . 19
kit_code_panel_ui . . . 20
kit_code_wrap . . . 21
kit_colors . . . 22
kit_current_brand . . . 22
kit_detail . . . 23
kit_details_row . . . 23
kit_dg_table . . . 24
kit_diagram . . . 25
kit_dims . . . 25
kit_done_status . . . 26
kit_download_button . . . 27
kit_dt . . . 28
kit_error_status . . . 30
kit_files . . . 31
kit_flow_tabs . . . 32

kit_gallery . . . 33
kit_header . . . 33
kit_is_cloud . . . 35
kit_is_user_error . . . 35
kit_logo_dependency . . . 36
kit_logo_file . . . 36
kit_logo_uri . . . 37
kit_navgroup . . . 37
kit_navgroups . . . 38
kit_need_files . . . 39
kit_new_app . . . 39
kit_next_step . . . 40
kit_notify . . . 41
kit_num . . . 42
kit_open_details . . . 42
kit_page . . . 43
kit_plotly_fonts . . . 44
kit_plotly_style . . . 45
kit_preview_table . . . 48
kit_problems . . . 49
kit_ready_line . . . 49
kit_render_code_panel . . . 50
kit_rows_note . . . 51
kit_run_gate . . . 52
kit_run_gate_ui . . . 53
kit_run_tab_ui . . . 54
kit_scope_bar . . . 57
kit_scroll_to . . . 58
kit_send_tab_states . . . 59
kit_settings_line . . . 59
kit_stale_status . . . 60
kit_static_table . . . 61
kit_status . . . 61
kit_status_ui . . . 63
kit_step_body . . . 64
kit_step_card . . . 64
kit_step_head . . . 65
kit_step_tabs_ui . . . 66
kit_stepcards_ui . . . 68
kit_steps . . . 69
kit_tab_guide . . . 70
kit_tab_states . . . 71
kit_table_scroll . . . 72
kit_test_download . . . 73
kit_theme . . . 73
kit_unload_guard . . . 74
kit_versions . . . 75
kit_watch_tab_states . . . 75

use_kit 76

Index **77**

`kit_abort` *An error meant for the user*

Description

The message should be plain-language Chinese that says what went wrong and how to fix it; `kit_error_status()` shows it as written in the status box, with no technical details. Any other, unexpected error is wrapped as "… " ("an unexpected problem occurred while …"), with the original message tucked away.

Usage

```
kit_abort(..., class = NULL, data = list())
```

Arguments

`...`	The message (joined with `paste0()`).
`class`	An extra condition class (the caller can handle it separately, e.g. to show a confirmation dialog).
`data`	Other fields to attach to the condition (a named list).

Value

Does not return: throws a `kit_user_error` error.

Examples

```
e <- tryCatch(kit_abort("   ", 2, "     "), error = identity)  # "Need files from at least 2 waves."
class(e)
conditionMessage(e)
```

`kit_advanced` *Advanced settings: collapsed by default*

Description

Options in a settings area that are rarely changed (text column matching, ID type conversion, reverse-scored item rules, separators…). A one-line small-print title; when opened, a faint line on the left sets it apart from the other settings (no box).

Usage

```
kit_advanced(..., title = "  ", open = FALSE, id = NULL)
```

Arguments

...	The settings inside.
title	The collapsible title (can be a tag, e.g. followed by the current settings).
open	Whether it starts open.
id	The id of the <details> (used by kit_open_details() to open it).

Value

<details class="kit-advanced">.

Examples

```
kit_advanced(shiny::checkboxInput("trim", "     ", TRUE))  # "Trim spaces before matching"
```

---

kit_app_description *Read the app's version and date (Version and Date in DESCRIPTION)*

Description

The cloud version, the local run.bat, and the package's run_xxx() all take Version and Date from DESCRIPTION, so all three show the same thing (file modification times are not used: unzipping, installing, and deploying all change the mtime).

Usage

```
kit_app_description(package = NULL, path = NULL)
```

Arguments

package	The app's package name, e.g. "kitJoin"; when NULL, only DESCRIPTION files are searched.
path	Path to a DESCRIPTION file; when NULL, searched in the order above.

Details

Search order:

1. path is given: read that DESCRIPTION file.
2. package is already loaded (library(), run_xxx(), load_all()): read the loaded copy.
3. DESCRIPTION in the working directory, then one and two levels up (in tests the working directory is tests/testthat); when package is given, only a file whose Package: matches counts. The cloud version and run.bat use runApp(folder), the package is not loaded, and this is the copy that gets read.
4. package is installed (but not loaded): read the installed copy. This comes last because it may be an old version.

Value

`list(version = , date = )`; anything not found is `NA`.

Examples

```
kit_app_description("kitDesign")
```

---

kit_app_link *A link to another website or app (opens in a new tab)*

Description

For example, a link from kitJoin to the cloud version of kitsUI. The link opens in a new tab (`target="_blank"`) and can be followed by a line of small print saying where it goes. Whether to show the link (local version, cloud version) is up to the app.

Usage

```
kit_app_link(label, href, kind = "link", class = NULL, note = NULL)
```

Arguments

`label`	The link text, e.g. ” kitsUI” (”Open kitsUI”).
`href`	The URL.
`kind`	`"link"` (a text link) or a button kind (`"secondary"`, `"primary"`).
`class`	An extra class for the link (e.g. `"kit-next-link"`).
`note`	A line of small print after it, e.g. ” kitsUI ” (”(opens the online cloud version of kitsUI)”); `NULL` shows nothing.

Value

`<span class="kit-app-link">`.

Examples

```
kit_app_link("  kitsUI", "https://example.org/kitsui/", kind = "secondary",
             note = "      kitsUI ")
```

kit_app_meta *The version line in the header, e.g. "kitJoin v0.6.0 · 2026-10-31"*

Description

The version line always sits in the header; it is needed when reporting problems.

Usage

```
kit_app_meta(
  version = NULL,
  date = NULL,
  extra = NULL,
  show_kitdesign = FALSE,
  package = NULL
)
```

Arguments

`version`	The version ("v" is added in front); `NULL` or `NA` shows nothing.
`date`	The date; `NULL` or `NA` shows nothing.
`extra`	Text to append (e.g. `paste("kits", utils::packageVersion("kits"))`).
`show_kitdesign`	Whether to append "kitDesign 0.1.0" (`kit_versions()`).
`package`	The app's package name, written before the version ("kitJoin v0.6.0"); `NULL` writes no name.

Value

`<div class="kit-app-meta">`; `NULL` when all three are missing.

Examples

```
kit_app_meta("0.5.0", "2026-10-31")
kit_app_meta("0.6.0", "2026-10-31", package = "kitJoin")
```

kit_brand *Brand: logo and colors*

Description

kitDesign looks like KIT by default. When another project or lab uses kitDesign, `kit_brand()` swaps the logo and colors; typeface, borders, shadows, and how components look follow the design language and stay the same. With no arguments it is KIT (`kit_brand()`), and pages look exactly as before.

Usage

```
kit_brand(
  name = "KIT",
  logo = NULL,
  mark = NULL,
  logo_alt = NULL,
  colors = list()
)
```

Arguments

`name`	The brand name, e.g. `"KIT"`, `"  "` (”Demo Lab”).
`logo`	Path to the header logo file (SVG or PNG); for `NULL` see above; `FALSE` shows no logo.
`mark`	Path to the browser tab icon file (SVG or PNG, square); when `NULL`, `logo` is used.
`logo_alt`	Alt text for the logo; when `NULL`, ”KIT ” (”KIT logo”) for KIT, or ” ” (” logo”).
`colors`	Colors to replace: a named list or named character vector; names are in the table above, values are `"#RRGGBB"` or color names R knows. An unknown name stops with an error listing the names you can use.

Details

Pass the brand to `kit_page()` (`brand =`), or set it once when the app starts: `options(kitDesign.brand = kit_brand(...))`. After that `kit_page()`, `kit_header()`, `kit_theme()`, `kit_colors()`, `kit_chart_sequential()`, and `kit_plotly_style()` all use it (`kit_current_brand()`).

Colors (`colors`) use the design-language names; anything not given follows KIT:

Name	CSS variable	KIT	Used for
`ink`	`--nb-ink`	`#141414`	text, borders, shadows
`page`	`--nb-page`	`#faf7f0`	page background
`card`	`--nb-card`	`#ffffff`	cards, inputs, tables
`accent`	`--nb-lime`	`#c6d033`	main color block: header, primary button, current tab
`accent_light`	`--nb-bg-new`	`#e8f0a0`	light main color: new columns in diagrams, light end of the colo
`primary`	`--nb-green`	`#006835`	links, checkboxes, progress bars, dark end of the color scale
`primary_hover`	`--nb-green-hover`	`#004d27`	link hover
`success`	`--nb-mint`	`#b9f0a8`	(done)
`warning`	`--nb-yellow`	`#ffde59`	(done, but take note)
`danger`	`--nb-salmon`	`#ff9b8f`	(stopped), danger button
`info`	`--nb-sky`	`#a8dcff`	info callout

If only `accent` is given, `accent_light` is derived from it (mixed with white); if only `primary` is given, `primary_hover` is derived from it (darkened). Text on color blocks is always `ink`: when choosing `accent` and the four status colors, check that `ink` on them has at least 4.5:1 contrast; `primary` is a text color and also needs 4.5:1 on `page`.

Logo: `logo` is the full header logo, `mark` is the small browser tab icon (square; when not given, `logo` is used). Both are paths to SVG or PNG files; the files are read when the brand is created, so changing the working directory later does not matter. In Shiny they are served by an htmlDependency containing only these two files (the folder they live in is not exposed); with `kit_header(logo = "inline")` they are written as a data URI. With `name = "KIT"` and no `logo`, the KIT logo is used; with any other name and no `logo`, the header has no logo. `logo = FALSE` never shows a logo.

Value

A `kit_brand` object (a list: `name`, `logo_alt`, `colors` (11 `#rrggbb` values), and the logo files).

See Also

`kit_current_brand()`, `kit_page()`, `kit_theme()`, `kit_chart_sequential()`

Examples

```
kit_brand()

# Another lab: blue colors, its own logo
demo <- kit_brand(
  "   ",
  logo = system.file("examples", "brand", "demo-logo.svg", package = "kitDesign"),
  mark = system.file("examples", "brand", "demo-mark.svg", package = "kitDesign"),
  colors = list(accent = "#9cc0ff", primary = "#1f4fa3", page = "#f5f7fa")
)
demo
if (interactive()) {
  options(kitDesign.brand = demo)   # use this brand for the whole app
  kit_gallery()
}
```

---

kit_btn_class *Class for a button kind*

Description

When you build a button yourself (e.g. `downloadButton()`, `<a class="btn">`), use this class so it looks the same as `kit_button()`.

Usage

```
kit_btn_class(kind = c("secondary", "primary", "danger"))
```

Arguments

`kind`	`"primary"` (primary button), `"secondary"` (secondary button), or `"danger"` (danger button).

Value

A class string, e.g. `"btn-sm btn-outline-secondary"`.

Examples

```
kit_btn_class("secondary")
```

`kit_button` *Button*

Description

A `shiny::actionButton()` with a class set by the button kind:

Usage

```
kit_button(id, label, kind = "secondary", ..., class = NULL)
```

Arguments

`id`	The button's input id.
`label`	The text on the button.
`kind`	`"primary"`, `"secondary"`, or `"danger"`.
`...`	Other arguments passed to `shiny::actionButton()` (e.g. `disabled = TRUE`, `icon`).
`class`	An extra class.

Details

- `"primary"`: primary button, lime background, large text, card shadow. One per page, plus the " " ("Next step") at the bottom.
- `"secondary"`: secondary button, white background, thin border, no shadow. For download, select all, example files, move up/down, copy.
- `"danger"`: danger button, salmon background. For delete, remove, skip this round.

For a regular button (white background, black border, small shadow, e.g. " " ("Yes, next step")) use `shiny::actionButton()` directly. Start the text with a verb and say clearly what pressing it does; if it leaves the page, add "→" after the text.

Value

`<button class="btn ...">`.

Examples

```
kit_button("run", "  ", "primary")       # "Run join"
kit_button("all", " ")                     # "Select all"
kit_button("drop", "   ", "danger")  # "Skip this round"
```

kit_callout *Callout*

Description

A reminder the user should know before pressing a button. It is not a result, so it has no shadow. At most one per section; if it can go in the status box, don't add a separate callout; if there is a button, put the button outside the box.

Usage

```
kit_callout(..., warn = FALSE, class = NULL)
```

Arguments

`...`	The content of the box.
`warn`	`FALSE`: sky-blue info; `TRUE`: yellow caution.
`class`	An extra class.

Value

`<div class="kit-callout">` (plus `kit-callout-warn` for caution).

Examples

```
kit_callout("             ")
# "External linking does not apply: this round has only one cohort. This is not an error."
kit_callout("             ", warn = TRUE)
# "The data seem to contain missing-value codes; go to 'Missing values' first."
```

kit_chart_colors *Categorical chart colors*

Description

Colorblind-friendly Okabe–Ito. Cohorts, scales, and scores take colors in order: blue, orange, green, reddish purple, sky blue, vermillion; past 6 categories it starts over. For reference lines, bands, and points with no judgment, use `kit_chart_neutral()`.

Usage

```
kit_chart_colors(n = NULL)
```

Arguments

`n`	When `NULL`, returns all 6 (named); when a number, takes the first `n` in order (unnamed).

Value

A character vector of colors.

Examples

```
kit_chart_colors()
kit_chart_colors(3)
```

---

kit_chart_neutral *Neutral color for non-categorical chart elements*

Description

Reference lines, bands, and points with no judgment.

Usage

```
kit_chart_neutral()
```

Value

`"#7F7F7F"`.

Examples

```
kit_chart_neutral()
```

---

kit_chart_sequential *Color scale for continuous values (chart)*

Description

Very light lime → light lime (`bg-new`) → KIT green (`nb-green`): larger values are darker. For heatmaps and color bars (e.g. pass rates in a response design plot). The minimum is not white, so 0 looks different from an empty cell with no data (white background). `kit_plotly_style()` switches heatmaps to this scale by default.

Usage

```
kit_chart_sequential(brand = kit_current_brand())
```

Arguments

`brand`	The brand (`kit_brand()`); defaults to `kit_current_brand()` (KIT when none is set).

Details

The scale comes from the brand (`kit_brand()`): 0 is a very light color made by mixing `accent_light` with white, 0.2 is `accent_light`, and 0.2 to 1 interpolates in Lab space from `accent_light` to `primary`. For the KIT brand these are the values above. Categorical colors (`kit_chart_colors()`, Okabe–Ito) do not change with the brand: they are colorblind-friendly.

Value

A plotly colorscale: `list(list(0, "#F3F7D3"), ..., list(1, "#006835"))`.

See Also

`kit_plotly_style()`, `kit_chart_colors()`

Examples

```
kit_chart_sequential()
kit_chart_sequential(kit_brand("Demo", colors = list(accent = "#9cc0ff", primary = "#1f4fa3")))
```

---

kit_check *One checklist row*

Description

An S3 value object (class `kit_check`): its content is checked when it is built; it has an `htmltools::as.tags()` method, so putting it straight into a page renders `<li class="kit-check">`. Usually placed inside `kit_checklist()`.

Usage

```
kit_check(state = c("todo", "ok", "note"), title, why = "", actions = NULL)

## S3 method for class 'kit_check'
as.tags(x, ...)

## S3 method for class 'kit_check'
format(x, ...)

## S3 method for class 'kit_check'
print(x, ...)
```

Arguments

`state`	`"todo"`, `"ok"`, or `"note"`.
`title`	One sentence (a string).
`why`	The small print below (why); `""` or length 0 shows nothing. (kitJoin's `kitsdata_check()` calls it `text`.)

`actions`	Buttons for the decision (a tag or list of tags); placed below the explanation.
`x`	A `kit_check` object.
`...`	Not used.

Details

A mark on the left, preceded by hidden text (read by screen readers):

- `"todo"`: **!** needs a decision (" ", "To decide:"). One bold sentence, a line of small print below saying why, and the decision buttons below that.
- `"ok"`: fine, or already decided (" ", "OK:"). Font weight 500.
- `"note"`: **i** a reminder, no decision needed (" ", "Note:").

Put the recommended option first and write " " ("(recommended)") on its button.

Value

A `kit_check` object.

Examples

```
kit_check("todo", "  3     1  ", "kits     0 1 2… ")
# "3 items have scores starting at 1" / "kits needs scores 0, 1, 2…"
htmltools::as.tags(kit_check("ok", "45      "))  # "All 45 items have integer scores"
```

`kit_checklist` *Checklist*

Description

Lists "things to decide" together with "things already fine", placed before a conversion or run. An S3 value object (class `kit_checklist`, a list of `kit_check()`): put it straight into a page and it renders as `<ul class="kit-checklist">`, one item per row, with the faintest line between rows.

Usage

```
kit_checklist(...)

## S3 method for class 'kit_checklist'
as.tags(x, ...)

## S3 method for class 'kit_checklist'
format(x, ...)

## S3 method for class 'kit_checklist'
print(x, ...)
```

Arguments

`...`	`kit_check()` items, or a list of them (`NULL` is skipped).
`x`	A `kit_checklist` object.

Value

A `kit_checklist` object.

Examples

```
cl <- kit_checklist(
  kit_check("todo", "  3     1  ", "kits     0 1 2… ",
            actions = kit_button("shift", "   1  ")),
  kit_check("ok", "     "),
  kit_check("note", "  2         ")
)
# todo: "3 items have scores starting at 1" / "kits needs scores 0, 1, 2…" / button "Subtract 1 from all
# ok:   "No suspected missing-value codes"
# note: "2 items have no responses and will not go into the response file"
cl
htmltools::as.tags(cl)
```

`kit_cloud_banner` *Cloud version reminder banner (one line below the header)*

Description

A yellow line reminding users that their data go to the cloud. The app writes the text itself (each app's server location and usage notes differ).

Usage

```
kit_cloud_banner(...)
```

Arguments

`...`	The banner content.

Value

`<div class="alert alert-warning kit-cloud-banner" role="alert">`.

Examples

```
kit_cloud_banner("                  ")
# "This is the cloud version: uploaded data are sent to a cloud server. For data with personal informatio
```

`kit_cloud_footer` *Cloud version usage notes (bottom of the page, small gray text)*

Description

A title, a few sections (bold section title followed by a period, then the body), and a final line of reference links (opening in new tabs). Section titles run into the body and body pieces are joined into one paragraph, with no spaces inserted by HTML. The header banner can link here with `<a href="#kit-cloud-terms">`.

Usage

```
kit_cloud_footer(
  sections,
  links = NULL,
  title = "    ",
  id = "kit-cloud-terms"
)
```

Arguments

`sections`	The sections: a named list or character vector; names are section titles (” ”, ”1. Data transfer and storage”), values are the body text (a character vector, joined directly).
`links`	Reference links: a named character vector, names are the link text and values are URLs; `NULL` omits this line.
`title`	The title at the top.
`id`	The footer's id (used by the banner's link).

Value

`<footer id="kit-cloud-terms" class="kit-cloud-footer">`.

See Also

`kit_cloud_banner()`, `kit_is_cloud()`

Examples

```
# Sections: "1. Data transfer and storage", "2. Disclaimer"
kit_cloud_footer(
  sections = list(
    "     " = c("     Posit Connect Cloud ",
                              "       "),
    "    " = "     "
  ),
  links = c("Posit Trust Center" = "https://trust.posit.co")
)
```

`kit_code` *Code for one version (code panel)*

Description

An S3 value object (class `kit_code`): code text plus a version name. It has an `htmltools::as.tags()` method: put it straight into a page and it renders a line with "version name (bold) + Copy" and the code box below, with the code first wrapped by `kit_code_wrap()` to at most 70 columns per line (Chinese characters count as 2). If a line is still too long after wrapping (e.g. a long string that cannot be broken), it gives a warning (it does not stop), and the box scrolls sideways.

Usage

```
kit_code(code, label = "R", width = KIT_CODE_WIDTH)

## S3 method for class 'kit_code'
as.tags(x, ...)

## S3 method for class 'kit_code'
format(x, ...)

## S3 method for class 'kit_code'
print(x, ...)
```

Arguments

`code`	The code (a character vector; one element per line, or containing newlines).
`label`	The version name, e.g. "tidyverse " ("tidyverse version"), "base R " ("base R version"), "kits " ("kits version").
`width`	Maximum columns per line; `NULL` means no wrapping and no check.
`x`	A `kit_code` object.
`...`	Not used.

Value

A `kit_code` object.

See Also

`kit_code_blocks()`, `kit_code_panel_ui()`

Examples

```
x <- kit_code("joined <- dplyr::left_join(t1, t2, by = \"id\")", "tidyverse ")
x
htmltools::as.tags(x)
```

kit_code_blocks *Content of "How to do it in R": a note plus code in one or more versions (with "Copy")*

Description

Each version is a `kit_code()`: a line with "version name (bold) + Copy" and the code box below it; two versions are stacked vertically. Code lines are at most 70 columns (`kit_code_wrap()`); ligatures are always turned off.

Usage

```
kit_code_blocks(snips, note = KIT_CODE_NOTE, width = KIT_CODE_WIDTH)
```

Arguments

`snips`	The code: a string, a `kit_code()` object, or a named list / character vector of them (names are version names). kitJoin's `list(tidy = , base = )` is labeled "tidyverse " ("tidyverse version") and "base R " ("base R version"); `NULL` versions are skipped (e.g. with `base = NULL` there is only the tidyverse version). Without names the version name is "R".
`note`	A one-line note at the top; `NULL` shows nothing.
`width`	Maximum columns per line; `NULL` means no wrapping and no check.

Value

`htmltools::tagList()`.

Examples

```
kit_code_blocks(list(
  tidy = "joined <- dplyr::left_join(t1, t2, by = \"id\")",
  base = "joined <- merge(t1, t2, by = \"id\", all.x = TRUE)"
))
kit_code_blocks(c("kits  " = "fit <- kits::kits_calibrate(resp)"))
```

kit_code_cols *Display width of each line (CJK and full-width characters count as 2 columns)*

Description

KIT code examples and notes are at most about 70 columns per line: Latin letters, digits, and symbols count as 1 column, Chinese characters (and other full-width characters) count as 2.

Usage

```
kit_code_cols(x)
```

Arguments

x	A character vector (one element per line).

Value

An integer vector: the number of columns in each line.

See Also

kit_code_wrap()

Examples

```
kit_code_cols(c("abc", "  CSV", ""))
# Which lines are wider than 70 columns
lines <- c("x <- 1", strrep(" ", 36))
which(kit_code_cols(lines) > 70)
```

---

kit_code_panel *Placeholder for "How to do it in R" (code panel)*

Description

A kit_detail() inside the details row (kit_details_row()), collapsed by default; the server fills output_id with kit_render_code_panel(). With modules, use kit_code_panel_ui().

Usage

```
kit_code_panel(output_id, title = "  R   ", open = FALSE)
```

Arguments

output_id	The uiOutput() id that holds the content (wrap in ns() inside a module).
title	The collapsible title.
open	Whether it starts open (collapsed by default).

Value

<details class="kit-detail">.

Examples

```
kit_details_row(kit_detail(" ", "…"), kit_code_panel("code"))
```

`kit_code_panel_ui` *"How to do it in R" module (code panel)*

Description

A collapsible block in the details row; the code is computed only when it is opened (Shiny does not render outputs inside a collapsed block), and rendered as one or more `kit_code()` (at most 70 columns per line, with "Copy").

Usage

```
kit_code_panel_ui(id, title = "  R   ", open = FALSE)

kit_code_panel_server(
  id,
  code,
  note = KIT_CODE_NOTE,
  width = KIT_CODE_WIDTH,
  empty = KIT_CODE_EMPTY
)
```

Arguments

`id`	The module id.
`title`	The collapsible title.
`open`	Whether it starts open.
`code`	A function or reactive returning the code: a string, a `kit_code()`, or a named list of them (see `snips` in `kit_code_blocks()`).
`note`, `width`, `empty`	See `kit_render_code_panel()`.

Value

`kit_code_panel_ui()`: `<details class="kit-detail">`, placed last in `kit_details_row()`.

`kit_code_panel_server()`: a reactive with the current code (a list of `kit_code()`; `NULL` when there is none yet).

See Also

`kit_code_panel()`, `kit_render_code_panel()` (low-level)

Examples

```
if (interactive()) {
  ui <- kit_page("KIT  ", NULL, kit_details_row(kit_code_panel_ui("code")))
  server <- function(input, output, session) {
    kit_code_panel_server("code", function() list(tidy = "x <- 1", base = "x <- 1"))
```

```
  }
  shiny::shinyApp(ui, server)
}
```

`kit_code_wrap` *Wrap code to at most 70 columns per line*

Description

"How to do it in R" wraps the code only right before showing it (`kit_code()` does this automatically), so functions that generate code need not worry about width:

Usage

```
kit_code_wrap(code, width = KIT_CODE_WIDTH)
```

Arguments

`code`	The code (a string; may have several lines, or be several strings).
`width`	Maximum columns per line.

Details

- Comments: break at spaces or between Chinese characters; continuation lines also start with "# "; punctuation is not put at the start of a line.
- Code: break after a comma inside parentheses, and align continuation lines just after that parenthesis (a line break inside parentheses is always valid R). If the parenthesis was opened on an earlier line (e.g. `mutate(` on its own line with the arguments on the next), continuation lines use the same indent as this line.
- If a trailing comment makes the line too wide, it is moved above the line.

After wrapping, the code is parsed again: if the expression differs from the original (or the original did not parse), the code is left unchanged. Places that cannot be broken (e.g. a very long string) are also left as is and scroll sideways in the code box.

Value

A single string (returned as is when `code` has length 0).

See Also

`kit_code_cols()`, `kit_code()`

Examples

```
code <- paste0(
  "x <- c(\"M03\", \"M03\", \"M03\", \"M36\", \"M36\", \"M36\", ",
  "\"M48\", \"M48\", \"M60\", \"M60\")"
)
cat(kit_code_wrap(code))
```

kit_colors *KIT brand color constants (for custom CSS or charts)*

Description

Same names as `kit_colors()` in kitJoin and kitsUI. Colors for UI components are in the CSS tokens (`var(--nb-lime)` etc.; see the appendix of `brand/en/design-language.md`). `primary`, `accent`, and `subtitle` come from the brand (`primary`, `accent`, and `accent_light` in `kit_brand()`); `primary_light` is a lighter `primary`; the rest are fixed neutrals and older alert colors.

Usage

```
kit_colors(brand = kit_current_brand())
```

Arguments

`brand`	The brand (`kit_brand()`); defaults to `kit_current_brand()` (KIT when none is set).

Value

A named list.

Examples

```
kit_colors()$primary
```

kit_current_brand *The current brand:* `options(kitDesign.brand = )`*, KIT when not set*

Description

The default `brand` for kitDesign functions (`kit_page()`, `kit_header()`, `kit_theme()`, `kit_colors()`, `kit_chart_sequential()`, `kit_plotly_style()`). An app only needs to set `options(kitDesign.brand = kit_brand(...))` once at startup.

Usage

```
kit_current_brand()
```

Value

A `kit_brand` object.

See Also

kit_brand()

Examples

```
kit_current_brand()$name
```

---

kit_detail *One detail: collapsed by default, a one-line small-print title*

Description

Outputs inside a collapsed detail are computed only when it is opened: `kit.js` tells Shiny to check again when it opens.

Usage

```
kit_detail(title, ..., open = FALSE)
```

Arguments

title	The title (a noun: " " ("Item parameters"), " " ("Log for this round")).
...	The content.
open	Whether it starts open.

Value

`<details class="kit-detail">`.

Examples

```
kit_detail(" ", shiny::tags$p("  10  "))  # "Preview", "First 10 rows"
```

---

kit_details_row *Details row*

Description

A row of collapsed small-print titles below the result (R ; "Preview", "Log for this round", "How to do it in R"). The one that is open takes a full line, and its content has no box. " R " ("How to do it in R") always goes last.

Usage

```
kit_details_row(...)
```

Arguments

...	kit_detail() (or kit_code_panel()).

Value

.

Examples

```
kit_details_row(
  kit_detail(" ", "  10  "),
  kit_detail("    ", "   ")
)
# "Preview" / "First 10 rows"; "Log for this round" / "No notes."
```

---

kit_dg_table *Small table for diagrams (2–3 columns, 2–4 rows)*

Description

A supporting element: no box, table as wide as its content, 1px thin lines, a light header (not the black header of large data tables); values in a monospace font. New columns get a light lime background and ” ” (”(new)”) after the name, so it does not rely on color alone; empty cells show a gray ” ” (”(empty)”).

Usage

```
kit_dg_table(df, title, added = NULL, row_header = FALSE)
```

Arguments

df	A data.frame (values are converted to text); NA or "" shows as ” ” (”(empty)”).
title	The table name, placed in <caption> (e.g. ” 1 t1 ” (”Wave 1 (t1)”), ” ” (”Result”)).
added	Names of columns added in this step (e.g. wave).
row_header	Whether column 1 holds row headers (<th scope="row">).

Value

<table class="kit-dg-table">.

Examples

```
kit_dg_table(data.frame(id = 1:2, score = c(3, NA)), "  1   t1 ")  # "Wave 1 (t1)"
```

kit_diagram — *Diagram: small table before → small table after, with a plain-language line below*

Description

Goes in the `diagram` of `kit_tab_guide()`. On narrow screens (576px) "before" is on top, the arrow points down, and "after" is below. Each small table has 2–3 rows and uses generic names (id, score, wave), not column names from a particular database.

Usage

```
kit_diagram(before, after, caption, arrow = "→")
```

Arguments

`before`	The small table(s) before: one `kit_dg_table()` or a list of them (side by side).
`after`	The small table after: one `kit_dg_table()`.
`caption`	One plain-language line (`<figcaption>`).
`arrow`	The arrow in the middle.

Value

`<figure class="kit-diagram">`.

Examples

```
kit_diagram(
  list(kit_dg_table(data.frame(id = 1:2, t1 = c(3, 5)), "  1  "),
       kit_dg_table(data.frame(id = c(1, 3), t2 = c(4, 6)), "  2  ")),
  kit_dg_table(data.frame(id = 1:3, t1 = c(3, 5, NA), t2 = c(4, NA, 6)), " "),
  "  id          "
)
# "Wave 1", "Wave 2", "Result";
# caption: "Joined by id: a wave the child missed is empty."
```

kit_dims — *Dimension text: "5 × 7 " ("5 rows × 7 columns")*

Description

Dimension text: "5 × 7 " ("5 rows × 7 columns")

Usage

```
kit_dims(nrow, ncol, preview_cols = NULL)
```

Arguments

`nrow`, `ncol`	Number of rows and columns.
`preview_cols`	Give this when the screen previews only the first few columns (if fewer than the column count, adds " N " ("previewing only the first N columns below")).

Value

A string.

Examples

```
kit_dims(3418, 25)
kit_dims(3418, 25, preview_cols = 20)
```

`kit_done_status` *Status after a run: (done, but take note) when something needs attention, otherwise (done)*

Description

Anything unusual must be seen: when there are `warnings`, the status cannot be a plain (done).

Usage

```
kit_done_status(
  message,
  warnings = character(0),
  notes = character(0),
  points = character(0)
)
```

Arguments

`message`	The conclusion on the first line (one sentence, with numbers and units): a string or tag.
`warnings`	Things to watch out for (a character vector, or a list of strings and tags). Listed first.
`notes`	Additional notes (small gray text).
`points`	Key results (listed after `warnings`; they do not turn the status into " ", "please note").

Value

A `kit_status` object.

Examples

```
kit_done_status("   3,389  ")  # "Join done: 3,389 children."
kit_done_status("   3,389  ", warnings = "  12   ID      ")
# warning: "12 children have duplicate IDs; only the first record is kept."
```

`kit_download_button` *Download button*

Description

Looks the same as `shiny::downloadButton()`, but Chinese file names save correctly in every browser: the last part of the URL, the `Content-Disposition` header (`filename` and `filename*`), and `<a download>` all carry the same file name. Put it inside `renderUI()`: it is registered again on each redraw, the URL changes with it, and you never get a stale file.

Usage

```
kit_download_button(
  session,
  id,
  label,
  filename,
  content,
  class = NULL,
  app = getOption("kitDesign.app", "kit")
)
```

Arguments

`session`	The module's session (the namespace is added to the id automatically).
`id`	The button id (without namespace).
`label`	The button text, e.g. " CSV" ("Download CSV").
`filename`	The download file name (a string; may contain Chinese and spaces).
`content`	`function(file)`: writes the file to `file`.
`class`	An extra class, e.g. `kit_btn_class("secondary")`, `"btn-primary"`.
`app`	The app name: used in log messages; when the file name is all Chinese, the ASCII fallback name for old browsers is its lowercase form (" .csv" → "kitjoin.csv"). Defaults to `getOption("kitDesign.app", "kit")`: the app sets it once in `.onLoad()`.

Details

The file is written only when the user clicks download (`content(file)`, which can read reactives). If writing fails: the error goes to the log (" …", " download failed (file name): …"), this session shows a red notification (which does not disappear on its own), and the response is 500 with the same sentence. Messages from `kit_abort()` are shown to the user as written; other errors do not expose internal messages.

The button is a `<button>`, not a link: the URL and the file name are in `data-kit-href` and `data-kit-download`, and `kit.js` (loaded by `use_kit()`) fetches the file in the background and saves it, so the page never leaves. Browser extensions that take over links on the page (some "document viewer" add-ons do, and the download then fails with "Not Found") find nothing to take.

`Content-Type` follows the extension: `.csv`, `.zip`, and `.html` get their own MIME types; everything else (`.sav`, `.csv.gz`) is `application/octet-stream`, so browsers that add an extension based on MIME type don't add an extra one.

Value

`<button type="button" class="btn btn-default kit-download" data-kit-href=" " data-kit-download=`
(`data-kit-href` is the download URL, `data-kit-download` the file name).

See Also

`kit_test_download()` (for tests)

Examples

```
if (interactive()) {
  server <- function(input, output, session) {
    output$dl_ui <- shiny::renderUI(kit_download_button(
      session, "dl", "  CSV", "    .csv",   # "Download CSV", "Wave 1 data.csv"
      function(file) utils::write.csv(mtcars, file)
    ))
  }
}
```

---

kit_dt *Interactive table (DT) with the KIT look*

Description

For tables people search, page through, and click: search results, item lists to edit. Same look as `kit_static_table()`: 2px frame, black header with white text, zebra stripes, `small` type, numbers right-aligned; a selected row is light lime (`bg-new`), not blue. Text is in Traditional Chinese (" 37 1–20 ", " ", " "); the page buttons look like small buttons. Width fits the content; on a phone a wide table scrolls sideways inside its own box, never the page.

Usage

```
kit_dt(
  df,
  selection = "none",
  page_length = 20L,
  search = FALSE,
  wrap_cols = NULL,
  min_widths = NULL,
  ordering = TRUE,
  options = list(),
  ...
)
```

Arguments

df	A data.frame. Column names stay as they are (data and function output keep their English names).
selection	"none", "single", or "multiple" (click to select rows). A list is passed to DT::datatable(selection =) as it is (for example with selected =).
page_length	Rows per page. 0 shows all rows (no paging).
search	Show the table's own search box. Usually FALSE: the page has its own conditions.
wrap_cols	Columns whose long text wraps (for example the item text); other columns do not wrap.
min_widths	Named vector of minimum column widths in CSS units, for example c(label = "24rem").
ordering	Click the header to sort.
options	More DataTables options (override the defaults). columnDefs given here come first, so they win over the ones kit_dt() makes.
...	Passed to DT::datatable().

Details

Use it inside DT::renderDT(). 'DT' is in Suggests: the app itself needs Imports: DT. For short result tables that need no paging, use kit_static_table().

Value

A datatables htmlwidget with class kit-dt.

See Also

kit_static_table() for short result tables, kit_table_scroll().

Examples

```
if (requireNamespace("DT", quietly = TRUE)) {
  kit_dt(data.frame(itemID = c("a1", "a2"), label = c("  ", " ")),
         selection = "single", wrap_cols = "label")
}
```

`kit_error_status` *Status for a failed run (, stopped)*

Description

Plain-language errors from `kit_abort()` are shown directly; for unexpected errors it says what to do, and the original message is tucked inside " " ("Original message").

Usage

```
kit_error_status(e, action, todo = NULL)
```

Arguments

`e`	The error (a condition).
`action`	The action name, e.g. " " ("join").
`todo`	A button or link for what to do (a tag); `NULL` shows nothing.

Value

A `kit_status` object.

Examples

```
e <- tryCatch(stop("object 'x' not found"), error = identity)
kit_error_status(e, " ")
u <- tryCatch(kit_abort("  2    "), error = identity)  # "Need files from at least 2 waves."
kit_error_status(u, " ")$message
```

`kit_files` *File helpers: encoding, SPSS variable names, compressed file paths, opening the browser*

Description

- `kit_fix_utf8_chr()`: converts text to UTF-8 (characters that cannot be converted become `?`); returns non-text input as is.
- `kit_fix_utf8_df()`: converts every text column (and factor) of a data frame to UTF-8, keeping SAV variable labels and value labels.
- `kit_is_valid_utf8_df()`: whether the column names and text columns are all valid UTF-8.
- `kit_sav_safe_names()`: converts to valid SPSS variable names (tested against `haven::write_sav()`: only letters, digits, and `_ . @ # $`; starts with a letter or `@`; does not end with `.`; not a reserved word; at most 64 bytes; unique ignoring case, with `_2`, `_3` added from the second duplicate on).
- `kit_truncate_bytes()`: truncates by UTF-8 byte count without cutting a Chinese character in half.
- `kit_gz_path()`: readr decides whether to decompress from the extension; if the path does not end in `.gz`, it copies the file to a temp file ending in `.gz`.
- `kit_open_browser()`: opens the URL in the default browser (on Windows, unaffected by the RStudio viewer).

Usage

```
kit_fix_utf8_chr(x)

kit_fix_utf8_df(df)

kit_is_valid_utf8_df(df)

kit_sav_safe_names(cols)

kit_truncate_bytes(x, max_bytes)

kit_gz_path(path)

kit_open_browser(url)
```

Arguments

`x`	A character vector (`kit_fix_utf8_chr()`) or a single string (`kit_truncate_bytes()`).
`df`	A data frame.
`cols`	Column names.

max_bytes	Maximum number of bytes.
path	A file path.
url	A URL.

Value

An object of the same kind as the input; kit_is_valid_utf8_df() gives TRUE / FALSE; kit_open_browser() gives NULL.

Examples

```
kit_sav_safe_names(c("1st wave", "AND", "id", "ID"))
kit_truncate_bytes(strrep(" ", 30), 64)
```

---

kit_flow_tabs *Ordered workflow tabs (the kitsUI style)*

Description

A regular shiny::tabsetPanel() with circled numbers in the tab names (, …; ”Data”, ”Item calibration”). Tabs that are not a workflow step (e.g. ” ”, ”User manual”) go in aside: dashed border, no shadow, pushed to the far right. Done tabs get a (an extra .kit-tab-mark in the tab); tabs that can’t be used yet are gray (kit_send_tab_states()).

Usage

```
kit_flow_tabs(..., id = "tabs", selected = NULL, aside = NULL)
```

Arguments

...	shiny::tabPanel().
id	The tabsetPanel id (input[[id]] is the current tab).
selected	The tab to start on (value).
aside	Values of tabs that are not a workflow step (a character vector).

Value

 containing a tabsetPanel().

Examples

```
kit_flow_tabs(
  shiny::tabPanel("   ", value = "data"),
  shiny::tabPanel("    ", value = "items"),
  shiny::tabPanel("  ", value = "manual"),
  id = "flow", aside = "manual"
)
```

kit_gallery *Component gallery app*

Description

One app holding every component, filled with KIT examples. The source is in `system.file("examples/gallery", package = "kitDesign")`.

Usage

```
kit_gallery(..., brand = NULL)
```

Arguments

`...`	Arguments passed to `shiny::runApp()` (e.g. `port = 7551`, `launch.browser = FALSE`).
`brand`	Which brand to view: `NULL` (default) is the current brand (`kit_current_brand()`, usually KIT); `"demo"` is a blue demo brand (to see how a different brand looks); or a `kit_brand()`.

Value

The result of `shiny::runApp()` (when the app closes).

Examples

```
if (interactive()) kit_gallery()
if (interactive()) kit_gallery(brand = "demo")
```

kit_header *KIT header*

Description

A lime color block, black border, header shadow; the full logo on the left, and on the right the tool name, one subtitle line, and one version line. One per app, at the very top (`kit_page()` already places it). It also sets the browser tab icon (the bean logo for KIT).

Usage

```
kit_header(
  title,
  subtitle = NULL,
  tagline = NULL,
  version = NULL,
  date = NULL,
```

```
  meta_extra = NULL,
  logo = TRUE,
  show_kitdesign = FALSE,
  brand = kit_current_brand(),
  package = NULL
)
```

Arguments

`title`	The tool name (the header's h1, also the browser tab title), for example "KIT " ("KIT data join tool").
`subtitle`	The subtitle (bold), for example the database name " " (the KIT database).
`tagline`	A phrase after the subtitle (for example " ", "cross-wave data processing"); `NULL` shows nothing.
`version`, `date`	The version and date on the header's version line, usually read from the app's own DESCRIPTION: `kit_app_description("kitJoin")`.
`meta_extra`	Text added after the version line (for example "kits 0.25.0").
`logo`	Whether the header shows the brand logo (KIT's is the bean plus the KIT wordmark): `TRUE` (the logo file comes from an htmlDependency; use in Shiny), `"inline"` (the logo is written into the page, for single HTML files without Shiny, such as user manuals), or `FALSE`.
`show_kitdesign`	Whether to add the kitDesign version after the version line ("kitDesign 0.1.0"), so both versions are visible when someone reports a problem.
`brand`	The brand: logo, tab icon, and colors (`kit_brand()`). Defaults to `kit_current_brand()`: KIT when `options(kitDesign.brand = )` is not set.
`package`	The app's package name, written at the start of the version line: "kitJoin v0.6.0 · 2026-10-31". Anyone reporting a problem can see at once which app and which version. `NULL` (the default) writes no name, as before.

Details

Logo, tab icon, and colors come from the brand (`kit_brand()`). When the brand's colors differ from KIT's, the header adds a short `<style>` that overrides the differing `--nb-*` variables; for the KIT brand nothing is added and the page is the same as before.

Value

`<div class="kit-header mb-4">`.

Examples

```
kit_header("KIT    ", "      ", tagline = "    ",
           version = "0.6.0", date = "2026-10-31", package = "kitJoin")
```

kit_is_cloud *Is this the cloud version?*

Description

Checks the environment variable `KIT_DEPLOY` first (`"cloud"` or `"local"`, set in Connect Cloud); if it is not set, Posit Connect's default `R_CONFIG_ACTIVE = "rsconnect"` counts as the cloud version.

Usage

```
kit_is_cloud()
```

Value

`TRUE` or `FALSE`.

See Also

`kit_cloud_banner()`, `kit_cloud_footer()`

Examples

```
kit_is_cloud()
```

kit_is_user_error *Was the error thrown by* `kit_abort()`*?*

Description

Was the error thrown by `kit_abort()`?

Usage

```
kit_is_user_error(e)
```

Arguments

`e`	The error (a condition).

Value

`TRUE` or `FALSE`.

Examples

```
kit_is_user_error(tryCatch(kit_abort("x"), error = identity))
```

kit_logo_dependency *htmlDependency for the logo files (inst/logo/): the header SVG and tab icon are fetched by URL*

Description

htmlDependency for the logo files (inst/logo/): the header SVG and tab icon are fetched by URL

Usage

```
kit_logo_dependency()
```

Value

An `htmltools::htmlDependency()` object; file URLs are `kitDesign-logo-<version>/<file name>` (`kit_logo_file()`).

Examples

```
kit_logo_dependency()$name
```

kit_logo_file *URL of a logo file (for use on the page)*

Description

Used for both the header's vector logo and the tab icon; the URL is served by `kit_logo_dependency()` (the page must include this dependency: `kit_header()` already attaches it). To embed the logo in the page without a separate file, use `kit_logo_uri()` (a PNG data URI).

Usage

```
kit_logo_file(file = "kit-logo.svg")
```

Arguments

`file`	A file name in `inst/logo/`: `"kit-logo.svg"` (full logo), `"kit-mark.svg"` (the bean), or a PNG (`"kit-mark-64.png"`, `"kit-logo-h512.png"`…).

Value

A relative URL, e.g. `"kitDesign-logo-0.1.0/kit-logo.svg"`.

Examples

```
kit_logo_file("kit-logo.svg")
```

kit_logo_uri *KIT logo (data URI)*

Description

Goes straight into the page (no static file path to set up). The logos are transparent PNGs in `inst/logo/`.

Usage

```
kit_logo_uri(size = 96, kind = c("mark", "logo"))
```

Arguments

`size`	Pixels: the bean (`"mark"`) comes in 64, 96, 256, `"h512"`, `"h1024"`; the full logo (`"logo"`, bean plus KIT wordmark) comes in 128, 192, `"h512"`, `"h1024"` (height). The header uses the vector logo (`kit_logo_file()`); this function gives a PNG to embed in the page.
`kind`	`"mark"` is the bean only (browser tab icon); `"logo"` is the full logo (header).

Value

`"..."`; `""` when that size does not exist.

Examples

```
substr(kit_logo_uri(64), 1, 30)
```

kit_navgroup *One group in the tab bar (for `kit_navgroups()`)*

Description

One group in the tab bar (for `kit_navgroups()`)

Usage

```
kit_navgroup(label, tabs, tools = FALSE)
```

Arguments

`label`	The group label (small gray text), e.g. ” ” (”Prepare”), ” ” (”Make the data (pick one)”), ” ” (”Other tools”).
`tabs`	A named character vector: names are tab values (`tabPanel(value = )`), values are tab names.
`tools`	When `TRUE`, these are standalone tools: dashed border, no shadow.

Value

A list (`class = "kit_navgroup"`).

Examples

```
kit_navgroup(" ", c(upload = "  ", missing = "  "))
# "Prepare": "File upload", "Missing values"
```

---

kit_navgroups *Grouped tab bar (unnumbered; the kitJoin style)*

Description

Tabs are split into groups, each with a line of small gray text above it. On click, `kit.js` switches the hidden `tabsetPanel(id = tabset, type = "hidden")`; after a switch (including `updateTabsetPanel()`) it marks the current tab as active. The for done tabs and the gray for tabs not yet usable are also applied to these links (`kit_send_tab_states()`). At 768px the bar becomes one row that scrolls sideways.

Usage

```
kit_navgroups(groups, tabset = "tabs", selected = NULL)
```

Arguments

`groups`	A list of `kit_navgroup()`.
`tabset`	The id of the matching `tabsetPanel()`.
`selected`	The tab to start on (value); defaults to the first.

Value

`<nav class="kit-navgroups">`.

Examples

```
groups <- list(
  kit_navgroup(" ", c(upload = "  ", missing = "  ")),
  kit_navgroup("    ", c(join = "   ", stack = "   ")),
  kit_navgroup("  ", c(convert = "   "), tools = TRUE)
)
# Groups: "Prepare" (File upload, Missing values), "Make the data (pick one)"
# (Wide join, Long stack of waves), "Other tools" (File format conversion)
kit_navgroups(groups, "tabs")
# Then on the page:
# shiny::tabsetPanel(id = "tabs", type = "hidden",
#   shiny::tabPanel("  ", value = "upload", ...), ...)
```

kit_need_files *" " ("Still needed"): the sentence shown when there are not enough files*

Description

" " ("Still needed"): the sentence shown when there are not enough files

Usage

```
kit_need_files(n_files, need = 2L)
```

Arguments

n_files	How many files have been added.
need	The minimum number of files this page needs.

Value

One sentence; NULL when there are enough files.

Examples

```
kit_need_files(1L, need = 2L)
kit_need_files(0L, need = 1L)
```

kit_new_app *Start a new KIT Shiny app*

Description

Creates a ready-to-run app package skeleton, with one example tab written to the KIT design language's page skeleton:

Usage

```
kit_new_app(
  path,
  title,
  subtitle = "      ",
  package = basename(path)
)
```

Arguments

`path`	The new folder (the last part is the package name, e.g. `"~/kitScore"`).
`title`	The tool name, e.g. "KIT " ("KIT scoring tool").
`subtitle`	The subtitle, e.g. " " (the KIT database).
`package`	The package name (defaults to the folder name): starts with a letter and contains only letters, digits, and periods.

Details

- `DESCRIPTION`: `Imports: kitDesign (>= current version)`, `Remotes: jiewenTsai/kitDesign@v<curren`
- `app.R`: the entry point for Posit Connect Cloud (primary file).
- `R/app.R`: `kit_page()` plus the step tabs module (`kit_step_tabs_ui()`); `run_app()`.
- `R/page_calc.R`: an example tab using a primary button plus the ready line (`kit_run_gate_ui()`), a status box, a result table, "How to do it in R" (`kit_code_panel_ui()`), and next step.
- `tests/testthat/`: one `shiny::testServer()` test.
- `.gitignore` (including `tests/testthat/_problems/` and `testthat-problems.rds`), `.Rbuildignore`, `README.md`, `LICENSE`.

If the folder already exists and is not empty, it does nothing and stops (it never overwrites any file).

Value

`path` (full path), invisibly.

Examples

```
dir <- file.path(tempdir(), "kitDemo")
kit_new_app(dir, "KIT   ", "      ")  # "KIT demo tool", the KIT database
list.files(dir, recursive = TRUE)
unlink(dir, recursive = TRUE)
```

---

kit_next_step *Next step*

Description

At the bottom of the page, aligned right: a line of small text on the left (it can be a list of places to go next), and a button on the right, " ××× →" ("Next step: ××× →"). Shown only after a result is ready (done, or done, but take note); removed when the result is out of date (, needs a rerun).

Usage

```
kit_next_step(
  text = NULL,
  button_id = NULL,
  label = NULL,
  button = NULL,
  kind = "primary"
)
```

Arguments

`text`	Small text on the left; `NULL` shows nothing.
`button_id`	The button id; when `NULL` no button is drawn (unless `button` is given).
`label`	The button text, e.g. ” ” (”Next step: wide join”); ” →” is added automatically.
`button`	A button to use directly instead of an actionButton (e.g. a link to another app).
`kind`	The button kind: primary button by default; use `"secondary"` when the page already has another main action (e.g. a download).

Value

`<div class="kit-next">`.

Examples

```
kit_next_step("      kits   ", "next", "  kits   ")
# "Join done; you can go on to the kits response file." / "Next step: kits response file"
```

---

kit_notify *Show a notification (Shiny's showNotification)*

Description

(stopped) is red and shows for 10 seconds; everything else uses the regular style and shows for 6 seconds.

Usage

```
kit_notify(st, session = shiny::getDefaultReactiveDomain(), duration = NULL)
```

Arguments

`st`	A `kit_status()` object, or a string (a plain notification).
`session`	The Shiny session (defaults to the current session).
`duration`	How many seconds to show it; when `NULL`, the defaults above apply.

Value

The notification id (invisible); `NULL` for `"idle"` or when there is no session.

Examples

```
if (interactive()) {
  kit_notify(kit_done_status("   "))  # "Join done."
}
```

---

kit_num *Integer with thousands separators*

Description

KIT design language: numbers with four or more digits get thousands separators, as in "3,389 " ("3,389 people"). Decimals are rounded to integers.

Usage

```
kit_num(x)
```

Arguments

`x`	A number (or a string that can be converted to a number).

Value

A string, for example `"3,418"`.

Examples

```
kit_num(3418)
kit_num(c(5, 12345, 1e6))
```

---

kit_open_details *Open a collapsible section (for example, open the advanced settings after applying a preset)*

Description

Open a collapsible section (for example, open the advanced settings after applying a preset)

Usage

```
kit_open_details(id, session = shiny::getDefaultReactiveDomain())
```

Arguments

`id`	The id of the `<details>` (the id inside the module; the namespace is added automatically), for example the `id` of `kit_advanced()`.
`session`	Defaults to the current session.

Value

The full id, invisibly; `NULL` when there is no session.

Examples

```
if (interactive()) kit_open_details("advanced")
```

---

kit_page *A KIT page: theme, styles, and header set up in one call*

Description

Wraps `bslib::page_fluid(lang = "zh-Hant-TW", theme = kit_theme(), use_kit(), kit_header(...), ...)`. Below the header, put the tab bar (`kit_navgroups()` or `kit_flow_tabs()`) and the content of each tab.

Usage

```
kit_page(
  title,
  subtitle = NULL,
  ...,
  tagline = NULL,
  version = NULL,
  date = NULL,
  logo = TRUE,
  meta_extra = NULL,
  window_title = title,
  show_kitdesign = FALSE,
  brand = kit_current_brand(),
  package = NULL
)
```

Arguments

`title`	The tool name (the header's h1, also the browser tab title), for example "KIT " ("KIT data join tool").
`subtitle`	The subtitle (bold), for example the database name " " (the KIT database).
`...`	The page content.

`tagline` A phrase after the subtitle (for example " ", "cross-wave data processing"); `NULL` shows nothing.

`version`, `date` The version and date on the header's version line, usually read from the app's own DESCRIPTION: `kit_app_description("kitJoin")`.

`logo` Whether the header shows the brand logo (KIT's is the bean plus the KIT wordmark): `TRUE` (the logo file comes from an htmlDependency; use in Shiny), `"inline"` (the logo is written into the page, for single HTML files without Shiny, such as user manuals), or `FALSE`.

`meta_extra` Text added after the version line (for example "kits 0.25.0").

`window_title` The browser tab title (defaults to `title`).

`show_kitdesign` Whether to add the kitDesign version after the version line ("kitDesign 0.1.0"), so both versions are visible when someone reports a problem.

`brand` The brand: logo, tab icon, and colors (`kit_brand()`). Defaults to `kit_current_brand()`: KIT when `options(kitDesign.brand = )` is not set.

`package` The app's package name, written at the start of the version line: "kitJoin v0.6.0 · 2026-10-31". Anyone reporting a problem can see at once which app and which version. `NULL` (the default) writes no name, as before.

Value

A complete page (`bslib::page_fluid()`).

See Also

`kit_header()`, `kit_app_description()`

Examples

```
if (interactive()) {
  ui <- kit_page(
    "KIT ××  ", "      ",
    tagline = "    ", version = "0.1.0", date = "2026-10-31",
    kit_tab_guide("   ", kit_steps("  ", "  "), "   ")
  )
  shiny::shinyApp(ui, function(input, output, session) {})
}
```

`kit_plotly_fonts` *Attach the app's typeface to a plotly chart (for downloaded HTML)*

Description

Attaches an htmlDependency to a plotly chart that loads the rounded typeface Chiron GoRound TC from Google Fonts (weights 400 and 700). Shiny pages already have this typeface (`kit_theme()`), so this is mainly for single HTML files saved with `htmlwidgets::saveWidget()`: they also open in the app's typeface; without a network connection they fall back to the system sans-serif.

Usage

```
kit_plotly_fonts(p)
```

Arguments

`p`	A plotly chart (or another htmlwidget).

Details

The `<link>` carries `data-external="1"`: when `saveWidget(selfcontained = TRUE)` bundles the file with pandoc, the font files are not embedded in the HTML (a Chinese typeface is split into hundreds of files, so embedding them would make the file very large). `kit_plotly_style()` already calls it by default (`fonts = TRUE`); attaching it twice to the same chart is the same as attaching it once.

Value

The same chart, with one more htmlDependency (`kit-plotly-fonts`).

See Also

`kit_plotly_style()`

Examples

```
if (requireNamespace("plotly", quietly = TRUE)) {
  p <- kit_plotly_fonts(plotly::plot_ly(x = 1:3, y = c(2, 4, 3), type = "scatter", mode = "lines"))
  vapply(p$dependencies, function(d) d$name, "")
}
```

`kit_plotly_style` *Give a plotly chart the KIT look (chart)*

Description

Makes plotly charts look like part of the same app: typeface, text sizes, and colors all use KIT's design variables. Only the look changes, not the data (points, lines, axis ranges, tick positions, and hover text stay the same).

Usage

```
kit_plotly_style(
  p,
  legend = c("auto", "bottom", "right", "keep"),
  colorscale = TRUE,
  fonts = TRUE,
  filename = "kit-chart",
  font_size = 15,
  brand = kit_current_brand()
)
```

Arguments

p	A plotly chart (`plotly::plot_ly()`, kits' `plot()`; one already passed through `plotly::plotly_build()` also works).
legend	Where the legend goes: • `"auto"` (default): with more than 4 entries, or when the longest entry is wider than 20 columns (a Chinese character counts as 2), the legend goes below the chart, horizontal, left-aligned; otherwise on the right. When the chart is narrower than 600px (phones) it also moves below, and returns to the right when the chart gets wider (decided in the browser). • `"bottom"`: always below; `"right"`: always on the right. • `"keep"`: leave the legend position alone (only change the font).
colorscale	When `TRUE` (default), heatmap-type traces switch to `kit_chart_sequential()`; when `FALSE`, the original color scale is kept.
fonts	When `TRUE` (default), attaches the rounded typeface from Google Fonts (`kit_plotly_fonts()`), so a downloaded single HTML file also uses the app's typeface.
filename	The file name (without extension) for the toolbar's "download PNG" and "download SVG".
font_size	Body text size (px); the other sizes are derived from it.
brand	The brand (`kit_brand()`): the heatmap color scale comes from it (`kit_chart_sequential()`). Defaults to `kit_current_brand()` (KIT when not set). Text, axis lines, and background keep the design language's ink and white.

Details

What changes:

- **Typeface**: everything uses the app's rounded typeface `"Chiron GoRound TC", sans-serif`; text is `nb-ink` (#141414), ticks are `text-hint` (#5a5a5a). Sizes: body `font_size` (default 15px), legend and hover box 1px smaller, ticks 2px smaller, title about `h5` (1.2 times `font_size`).
- **Title**: left-aligned, bold. When the chart is too narrow, it wraps in the browser (breaking between Chinese characters or at spaces).

- **Every axis** (`xaxis`, `xaxis2…`, `yaxis…`, including side-by-side small charts such as a Wright map): axis line `nb-ink` 1.5px, grid lines `line-soft` (#e3ded2), zero line `line` (#d9d4c5), ticks outside; no frame on four sides (`mirror = FALSE`, and the black frame kits draws around each small chart is removed; KIT design language: no frames inside frames). When axis titles and ticks do not fit, the margins widen automatically (`automargin`).
- **Background**: white (the chart sits in a white card); downloaded PNGs and SVGs also have a white background.
- **Colors**: `colorway` is Okabe–Ito (`kit_chart_colors()`). plotly's own default colors are replaced with the Okabe–Ito color in the same position; **colors set by kits or kitsUI stay unchanged** (kitsUI's role colors and category recoloring are kept).
- **Color scale** (`colorscale = TRUE`): heatmap-type traces (heatmap, contour, histogram2d) switch to `kit_chart_sequential()` (very light lime → light lime → KIT green); the color bar's text and outline always get the new look.
- **Hover box**: white background, `nb-ink` border, the app's typeface.
- **Legend**: the app's typeface, no border. For position, see `legend`.
- **Toolbar**: no plotly logo; only "download PNG", "download SVG", zoom in, zoom out, and reset to the original range remain; interface text is Traditional Chinese (`zh-TW`). The PNG is always 2100 pixels wide (7 inches at 300 dpi; the aspect ratio is the one on screen) and the file says 300 dpi, so it is 7 inches wide when inserted in Word. The SVG is a vector image for journals that want 600 dpi or more, EPS or PDF; its text uses the fonts on the computer that opens it (the rounded typeface is a web font). Both are made in the browser, not by R, and also work in a downloaded HTML file.
- **Margins**: equally small on all sides; plotly widens them automatically when axes, legend, or color bar do not fit, so no text is cut off.

What does not change: data, axis ranges and ticks, trace colors (other than plotly's default colors), marker shapes, line widths and dashes, opacity, hover text, legend text, the text of titles and axis titles (only bold is added), chart height and width, and `shapes` (other than kits' small-chart frames; for example, the gray band in fit plots).

Order when used with kits and kitsUI: kits' `plot()` → kitsUI recoloring (`kit_chart_categorical()`, `kit_chart_role()`) → kitsUI Chinese labels (`kit_chart_zh()`) → `kit_plotly_style()`. Recoloring and this function can be swapped (this function does not touch set colors), but **Chinese labels must come first**: this function wraps the title in `<b>`, after which `kit_chart_zh()` no longer recognizes the original English title. Applying it twice to the same chart is the same as applying it once.

Value

A plotly chart (passed through `plotly_build()`).

See Also

`kit_plotly_fonts()`, `kit_chart_colors()`, `kit_chart_sequential()`

Examples

```
if (requireNamespace("plotly", quietly = TRUE)) {
  d <- data.frame(wave = rep(c(3, 6, 12), 2), score = c(8, 11, 15, 7, 10, 13),
                  cohort = rep(c("M03", "M36"), each = 3))
  p <- plotly::plot_ly(d, x = ~wave, y = ~score, split = ~cohort,
                       type = "scatter", mode = "lines+markers")
  p <- plotly::layout(p, title = "     ")
  kit_plotly_style(p)
}
```

---

kit_preview_table *Data for a preview table (pass to* `shiny::renderTable()`*)*

Description

With too many columns, only the first `max_cols` columns are kept (`priority_cols` are kept first); only the first `n` rows are kept. Each column is converted to easy-to-read text (value labels as "`[value] label`", dates as yyyy-mm-dd). Put a `kit_rows_note()` line above it.

Usage

```
kit_preview_table(
  df,
  priority_cols = NULL,
  n = KIT_PREVIEW_ROWS,
  max_cols = KIT_PREVIEW_COLS
)
```

Arguments

`df`	A data.frame.
`priority_cols`	Columns that must be in the preview (for example the ID column); they go first.
`n`	Maximum number of rows.
`max_cols`	Maximum number of columns.

Value

A data.frame (all columns are text).

Examples

```
df <- as.data.frame(matrix(1:60, nrow = 2))
dim(kit_preview_table(df, priority_cols = "V30", n = 1))
```

`kit_problems` *Problems in a table or list (dark orange-red text)*

Description

A list of individual problems inside a step card, such as " …" ("Still needed: …") or " " ("these columns have problems").

Usage

```
kit_problems(..., head = NULL, bullets = TRUE)
```

Arguments

`...`	One string (or tag) per problem.
`head`	Bold text in front, for example " " ("Still needed:"); `NULL` shows nothing.
`bullets`	`TRUE` (default): one bullet per problem; `FALSE`: one per line, without bullets (for example when each line is already a full sentence, or already has its own numbering).

Value

`<div class="kit-problems">`; `NULL` when there are no problems.

Examples

```
kit_problems("  2    ", "  4     ", head = "  ")
kit_problems("bsrs1 w1d5_1   w2d5_1     ", bullets = FALSE)
```

`kit_ready_line` *Ready line ("what's still needed")*

Description

When the primary button cannot be pressed yet, a bold line above it says " …" ("Still needed: …"), followed by a link to where to fix it. Write what is still needed as concrete tasks and numbers: " 2 1 " ("Need files from at least 2 waves; there is only 1 now").

Usage

```
kit_ready_line(missing, link = NULL)
```

Arguments

`missing`	What is still needed (character vector; `NA` and `""` are skipped).
`link`	A link that follows (for example " →", "Go to 'File upload' to add files →"); `NULL` shows nothing.

Value

`<p class="kit-ready todo">`; `NULL` when nothing is missing.

See Also

`kit_run_gate()`, `kit_need_files()`

Examples

```
kit_ready_line(c("  2       1 ", " ID "))
```

---

kit_render_code_panel

Server side: render code into `kit_code_panel()`

Description

`output[["code"]] <- kit_render_code_panel(function() code_text())`. Nothing is computed while the panel is collapsed (Shiny does not render hidden outputs; it computes only when " R " ("How to do it in R") is opened). When `code()` stops (`req()`), fails, or is empty, only the `empty` line is shown.

Usage

```
kit_render_code_panel(
  code,
  note = KIT_CODE_NOTE,
  width = KIT_CODE_WIDTH,
  empty = KIT_CODE_EMPTY
)
```

Arguments

`code`	A function or reactive that returns the code: the `snips` of `kit_code_blocks()`.
`note, width`	See `kit_code_blocks()`.
`empty`	The one line shown while there is no code yet.

Value

`shiny::renderUI()`.

Examples

```
if (interactive()) {
  server <- function(input, output, session) {
    output$code <- kit_render_code_panel(function() "x <- 1")
  }
}
```

`kit_rows_note` *Note for a long table: " 3,418 × 25 10 20 " ("3,418 rows × 25 columns in total; first 10 rows and 20 columns below.")*

Description

One line of small `kit-hint` text placed above the table.

Usage

```
kit_rows_note(nrow, ncol = NULL, shown_rows, shown_cols = NULL)
```

Arguments

`nrow`, `ncol`	Total number of rows and columns (with `ncol = NULL`, columns are not mentioned).
`shown_rows`, `shown_cols`	How many rows and columns are shown on screen (with `shown_cols = NULL`, columns are not mentioned).

Value

A string.

Examples

```
kit_rows_note(157, shown_rows = 20)
kit_rows_note(3418, 25, shown_rows = 10, shown_cols = 20)
kit_rows_note(8, shown_rows = 10)
```

`kit_run_gate` *When the primary button cannot be pressed yet: disable it and show what is still needed above it*

Description

Put `shiny::uiOutput(ns("ready"))` on the page, directly followed by `kit_button(ns("run"), ..., "primary", disabled = TRUE)` (disabled at first; enabled once the server has checked). The primary button is shown from the start, so people know what they will press at the end of this page.

Usage

```
kit_run_gate(
  session,
  output,
  missing,
  link = function() NULL,
  button_id = "run",
  output_id = "ready"
)
```

Arguments

`session`	The module's session.
`output`	The module's output.
`missing`	A reactive: what is still needed (character; length 0 = ready to run).
`link`	function(): the link after " " ("Still needed") (a tag or `NULL`).
`button_id`	The primary button's id (the id inside the module, without `ns()`).
`output_id`	The `uiOutput()` id for the " " line (the id inside the module).

Details

Still check again inside the button's `observeEvent()`: disabling is only on screen and must not be the only safeguard.

Value

`NULL`, invisibly; call it inside the server.

Examples

```
if (interactive()) {
  server <- function(input, output, session) {
    kit_run_gate(session, output, shiny::reactive(kit_need_files(1L, need = 2L)))
  }
}
```

`kit_run_gate_ui` *Primary button + "Still needed: …" module (ready line)*

Description

The primary button is shown (disabled) from the start; it can be pressed only when nothing is still needed (`missing()` is empty). Above it, a bold line says " …" ("Still needed: …"), followed by a link to where to fix it. The reactive returned by the server changes only when the button is pressed **while nothing is still needed** (the server also blocks it, not just the disabled button), so a plain `observeEvent(run(), …)` is enough.

Usage

```
kit_run_gate_ui(id, label, ...)

kit_run_gate_server(id, missing, link = function() NULL)
```

Arguments

`id`	The module id.
`label`	The primary button's text (start with a verb, for example " ", "Run join").
`...`	Other arguments passed to `kit_button()`.
`missing`	A reactive: what is still needed (character vector; length 0 = ready to run).
`link`	A function: the link after " " ("Still needed") (a tag or `NULL`).

Value

`kit_run_gate_ui()`: the slot for the " " line and the primary button.

`kit_run_gate_server()`: a reactive counting presses made while ready to run (0 at first).

See Also

`kit_ready_line()`, `kit_run_gate()` (low-level)

Examples

```
if (interactive()) {
  ui <- kit_page("KIT  ", NULL,
    shiny::selectInput("id_col", "ID  ", c("", "subj_id")),
    kit_run_gate_ui("run", "  ")
  )
  server <- function(input, output, session) {
    run <- kit_run_gate_server("run", shiny::reactive(if (input$id_col == "") "  ID  "))
    shiny::observeEvent(run(), message(" "), ignoreInit = TRUE)
  }
```

```
  shiny::shinyApp(ui, server)
}
```

`kit_run_tab_ui` *Run tab*

Description

Page skeleton for a tab with one primary button. `kit_run_tab_ui()` lays out the fixed parts following the page skeleton; the tab's own settings go in `....` `kit_run_tab_server()` is called inside the tab's `moduleServer()` and handles the status flow:

Usage

```
kit_run_tab_ui(
  id,
  run_label,
  guide = NULL,
  ...,
  gate = TRUE,
  result = FALSE,
  downloads = TRUE,
  details = TRUE,
  code = TRUE,
  next_step = FALSE
)

kit_run_tab_server(
  input,
  output,
  session,
  run_label,
  run,
  done = NULL,
  ready = NULL,
  ready_link = function() NULL,
  settings = NULL,
  stale_on = NULL,
  progress = NULL,
  error = function(e) kit_error_status(e, run_label),
  guard = NULL,
  confirm = list(),
  result_ui = NULL,
  details_ui = NULL,
  code = NULL,
  code_note = KIT_CODE_NOTE
)
```

Arguments

id	The tab (module) id.
run_label	The primary button's text (start with a verb, for example " ", "Run join").
guide	The "what this page does" at the top (kit_tab_guide()).
...	The tab's own settings (inputs, hints), placed above the settings summary.
gate	TRUE: the primary button starts disabled, with the " " ("Still needed") line above it; FALSE: always clickable (no ready).
result, downloads, next_step	The key results below the status box, the download area, and the next step at the very bottom: TRUE is a uiOutput() (ids result, downloads, next_step); you can also pass a tag directly; FALSE leaves it out.
details	Details row: TRUE adds uiOutput("result_details") (preview and checks of the result); when both details and code are FALSE, there is no details row.
code	UI (kit_run_tab_ui()): TRUE adds " R " ("How to do it in R") (kit_code_panel(), output code). Server (kit_run_tab_server()): a function that returns the code (the code of kit_render_code_panel()); with NULL, the tab renders it itself.
input, output, session	The input, output, and session of the tab's moduleServer().
run	The function that runs: returns the result (any object), or a kit_status() (no result; for example, a problem must be handled first). To stop, call kit_abort() (with a plain-language reason).
done	function(result): the status after running (kit_done_status()); the result is already saved before it is called.
ready	A reactive: what is still needed (character vector; length 0 = ready to run); with NULL, there is no " " line (gate = FALSE).
ready_link	A function: the link after " " (a tag or NULL).
settings	A function: the parts of the settings summary (character vector; NULL shows nothing); with NULL, there is no settings summary.
stale_on	A function: things whose change makes the result out of date (for example list(input$method, data())); with NULL, the tab handles this itself (using the returned result and status).
progress	The progress bar message (" ..."", "Joining data..."); with NULL, there is no progress bar.
error	function(e): the status on error; defaults to kit_error_status(e, run_label).
guard	A function: a check before running; when it returns a kit_status(), nothing runs (status box and notification, no scrolling); NULL skips the check.
confirm	Errors the user must confirm: a named list; names are condition classes, values are function(e). It opens a confirmation dialog and returns a status (no notification, no scrolling; the dialog is the response). After confirming, run again with the returned run(...).

`result_ui`	`function(result)`: the key results below the status box (output `result`); with `NULL`, the tab renders it itself.
`details_ui`	`function(result)`: the first cell of the details row (output `result_details`, for example `kit_detail()` " ", "Result preview"); with `NULL`, the tab renders it itself.
`code_note`	The one-line note at the top of " R ".

Details

- Primary button: disabled while something is still needed (`ready()` is not empty), with a " …" ("Still needed: …") line above it (`kit_run_gate()`).
- Settings summary: the parts of `settings()` joined into one line (`kit_settings_line()`).
- Pressing the primary button: clear the old result → `guard()` → `run()` inside a progress bar → save the result, `done(result)` is the status; on error, `error(e)` is the status → status box, notification (`kit_notify()`), scroll to the status box (`kit_scroll_to()`).
- When settings or data change (`stale_on()` changes) and there is a result: the result is cleared and the status is ("Run again").
- Key results, the first cell of the details row, and "How to do it in R": rendered only when there is a result.

Fixed output ids (without namespace): `settings_line`, `ready`, `run` (the button), `status`, `result`, `downloads`, `result_details`, `code`, `next_step`.

Value

`kit_run_tab_ui()`: a `tagList()`.

`kit_run_tab_server()`: a list,

- `result`: a `reactiveVal`, the result of this run (`NULL` when there is none yet or it is out of date);
- `status`: a `reactiveVal`, the status box's `kit_status()`;
- `state`: a reactive, the status box's state (used for the on step tabs, `kit_tab_states()`);
- `run(...)`: runs once, just like pressing the primary button (`...` goes to `run()`; for example " ", "Run anyway", in a confirmation dialog).

See Also

`kit_run_gate()`, `kit_status()`, `kit_code_panel()`

Examples

```
if (interactive()) {
  mod_stack_ui <- function(id) {
    kit_run_tab_ui(
      id, "  ",
      guide = kit_tab_guide("      ", c("    ", "    "), "     "),
      shiny::numericInput(shiny::NS(id, "n"), "  ", 2),
      downloads = FALSE
```

```
    )
  }
  mod_stack_server <- function(id) {
    shiny::moduleServer(id, function(input, output, session) {
      tab <- kit_run_tab_server(
        input, output, session, "  ",
        ready = shiny::reactive(if (input$n < 2) "  2  "),
        settings = function() paste0(input$n, "  "),
        stale_on = function() input$n,
        progress = "  …",
        run = function() data.frame(wave = seq_len(input$n)),
        done = function(res) kit_done_status(paste0("  ", nrow(res), "  ")),
        details_ui = function(res) kit_detail("  ", kit_static_table(res)),
        code = function() "stacked <- dplyr::bind_rows(t1, t2, .id = \"wave\")"
      )
      list(state = tab$state)
    })
  }
  shiny::shinyApp(kit_page("KIT  ", NULL, mod_stack_ui("stack")),
                  function(input, output, session) mod_stack_server("stack"))
}
```

`kit_scope_bar` *Scope bar: an app-wide choice under the tab bar*

Description

For a choice that every tab of the app depends on, such as which item bank is in use. It sits right under the tab bar, without a frame: a radio group (or a select), one line of small hint text (what is in use now, how big it is), small buttons on the right, and an optional area for what must be confirmed after choosing (a checklist and buttons). A thin line separates it from the page below.

Usage

```
kit_scope_bar(input, line = NULL, actions = NULL, panel = NULL)
```

Arguments

`input`	The input: usually `shiny::radioButtons(..., inline = TRUE)` (2–5 choices) or `shiny::selectInput()`.
`line`	Tags or text for the hint line, for example `"  TDCYP  · 412 "`. `NULL` for none. Put dynamic parts in `shiny::uiOutput()` / `shiny::textOutput(inline = TRUE)`.
`actions`	Small buttons on the right of the hint line (`kit_button()` with the default kind, `kit_download_button()`, `shiny::fileInput()`). A list or a single tag; `NULL` for none.
`panel`	Optional area under the line for what must be confirmed after choosing (for example a `kit_checklist()` and buttons, or a `shiny::uiOutput()`). `NULL` for none.

Details

Markup: `div.kit-scopebar` > the input, `div.kit-scopebar-line` (`line`, then `div.kit-scopebar-actions` with `actions`), `div.kit-scopebar-panel` (`panel`). Use `span.kit-scopebar-todo` in `line` for something still to do (" ").

Value

`<div class="kit-scopebar">`.

Examples

```
kit_scope_bar(
  shiny::radioButtons("bank", " ", c("TDCYP" = "tdcyp", "   " = "own"),
                      inline = TRUE),
  line = "  TDCYP   · 412  ",
  actions = list(kit_button("bank_dl", "  "), kit_button("bank_new", "  "))
)
```

---

kit_scroll_to *After running, scroll the status box to the top of the screen*

Description

"Results first, details after": call `kit_scroll_to("status")` at the end of the primary button's `observeEvent()`. It scrolls only after this round's outputs have all reached the browser (`session$onFlushed()`), so the status box is already rendered. When the user has set "reduce motion", it jumps there without smooth scrolling.

Usage

```
kit_scroll_to(id, session = shiny::getDefaultReactiveDomain())
```

Arguments

`id`	The output id inside the module (the module's namespace is added automatically), for example `"status"`.
`session`	Defaults to the current session (called inside a module, that is the module's session).

Value

The full id, invisibly; `NULL` when there is no session.

Examples

```
if (interactive()) {
  shiny::observeEvent(input$run, {
    # …run…
    kit_scroll_to("status")
  })
}
```

`kit_send_tab_states` *Send tab states to the browser*

Description

Works with both kinds of tab bar: grouped tab bars (`kit_navgroups()`, `data-tabset` is `tabset`) and flow tabs (`kit_flow_tabs()`, `id` is `tabset`).

Usage

```
kit_send_tab_states(states, session = shiny::getDefaultReactiveDomain())
```

Arguments

`states`	The result of `kit_tab_states()`.
`session`	Defaults to the current session.

Value

`states`, invisibly.

Examples

```
if (interactive()) kit_send_tab_states(kit_tab_states(list(upload = "success")))
```

`kit_settings_line` *Settings summary: a line of small text above the primary button or the result*

Description

For example ” 5 · ID · ” (”This run: 5 files · ID: per file · complete-attendance sample”).

Usage

```
kit_settings_line(parts, prefix = "  ")
```

Arguments

parts	The parts of the text (empty strings, NA, and NULL are skipped), joined with " · "; can also be a complete sentence written beforehand (kitsUI's ui_settings_line(text); in that case use prefix = NULL).
prefix	The text in front, default " " ("This run:"); NULL adds nothing.

Value

<p class="kit-settings-line">; NULL when there are no parts.

Examples

```
kit_settings_line(c("5   ", "ID   ", "   "))
kit_settings_line("   R1 M03 M36 ·    M03", prefix = NULL)
```

---

kit_stale_status *Old result no longer valid after files or settings change (, "Run again")*

Description

Old result no longer valid after files or settings change (, "Run again")

Usage

```
kit_stale_status(run_label, message = NULL)
```

Arguments

run_label	The primary button's text, for example " " ("Run join").
message	Your own sentence (the default is " … ", "Files or settings changed and the old result was cleared: press '…' again.").

Value

A kit_status object.

Examples

```
kit_stale_status("  ")$message
```

kit_static_table — *Small table (put directly inside renderUI(); no separate output needed)*

Description

White background, black header with white text, zebra stripes, width fits the content. Integer columns get thousands separators automatically (3389 → 3,389). Put an `h4` title and a `kit_rows_note()` line above it.

Usage

```
kit_static_table(df, class = "table table-sm table-striped")
```

Arguments

`df`	A data.frame.
`class`	The table's class.

Value

`<table class="table table-sm table-striped">`.

Examples

```
kit_static_table(data.frame(  = c("M03", "M36"),   = c(3389, 1250)))
```

kit_status — *Status object (status box)*

Description

The result of an action. This is an S3 value object (class `kit_status`): its content is checked when it is built. It has an `htmltools::as.tags()` method, so `renderUI(st)` or putting it straight into `tagList()` renders it as a status box (the same as `kit_status_ui()`). `print(st)` prints a text version.

Usage

```
kit_status(
  state = "idle",
  message = "",
  warnings = character(0),
  notes = character(0),
  details = NULL,
  action = NULL,
```

```
  points = character(0),
  details_title = "   "
)

## S3 method for class 'kit_status'
as.tags(x, ...)

## S3 method for class 'kit_status'
format(x, ...)

## S3 method for class 'kit_status'
print(x, ...)
```

Arguments

`state`	`"idle"`, `"success"`, `"warning"`, `"error"`, or `"stale"`.
`message`	The conclusion on the first line (one sentence, with numbers and units): a string or tag.
`warnings`	Things to watch out for (a character vector, or a list of strings and tags). Listed first.
`notes`	Additional notes (small gray text).
`details`	The raw message (R's original error text; collapsed): a string or `NULL`.
`action`	A button or link for what to do (a tag); `NULL` shows nothing.
`points`	Key results (listed after `warnings`; they do not turn the status into " ", "please note").
`details_title`	The collapsible title for the raw message (kitsUI uses "kits ", "kits' original text").
`x`	A `kit_status` object.
`...`	Other content for `kit_status_ui()` (as.tags); not used by format and print.

Details

Each of the four states has a fixed label and color:

state	Label	Background
`"success"`	("Done")	mint green
`"warning"`	("Done, but please note")	yellow
`"error"`	("Stopped")	salmon
`"stale"`	("Run again")	yellow

`"idle"` (not run yet) is not shown. kitsUI's names are also accepted: `"done"`, `"warn"`, `"stop"`.

Value

A `kit_status` object.

See Also

`kit_status_ui()`, `kit_done_status()`, `kit_error_status()`, `kit_stale_status()`

Examples

```
st <- kit_status("success", "     3,389   × 2,058   ")
st
htmltools::as.tags(st)
```

---

kit_status_ui *Status box*

Description

Renders a `kit_status()` as a status box: the first line is the label plus a one-sentence conclusion; then up to 3 points (things to watch out for first, key results after), the rest collapsed under ” N ” (”N more points”); a button for what to do; small additional notes; the collapsed raw message. After running, use `kit_scroll_to()` to scroll it to the top of the screen.

Usage

```
kit_status_ui(st, ..., max_points = KIT_STATUS_MAX_POINTS)
```

Arguments

`st`	A `kit_status()` object (a list with the same fields, from older apps, also works); `NULL` or `"idle"` shows nothing.
`...`	Other content placed last inside the box (kitsUI's `ui_status(...)`).
`max_points`	How many points to list directly (default 3).

Details

A `kit_status` object can also go straight into the page (its `htmltools::as.tags()` method is this function); you only need to call this function to add more content or change the number of points.

Value

`<div class="alert alert-success kit-status" role="alert">` or similar; `NULL` when nothing is shown.

Examples

```
kit_status_ui(kit_done_status(
  "     3,389   × 2,058  ",
  warnings = c("  12   ID   ", "  3    sex  "),
  points = c("688      ", "  1   3,389  ")
))
```

`kit_step_body` *Step card content*

Description

Tables inside the card have no outer border and a light header (CSS handles this); no indent on phones.

Usage

```
kit_step_body(...)
```

Arguments

`...`	The content of this step; usually ends with " " ("Yes, next step") (`shiny::actionButton()`, a regular button).

Value

`<div class="kit-stepbody">`.

Examples

```
kit_step_body(shiny::p(" "), shiny::actionButton("ok1", "   "))
```

`kit_step_card` *Step card*

Description

One card: on top is `kit_step_head()` (usually rendered by the server based on the state, in a `uiOutput()`), below is the content of this step (`kit_step_body()`, usually wrapped in `conditionalPanel()` so only the current step is open). The card's border and shadow follow the header's state: the current step has a border and shadow; finished steps and steps not yet available have no shadow and a gray border.

Usage

```
kit_step_card(head, ..., title_id = NULL)
```

Arguments

head	The header: kit_step_head(), or the uiOutput() that holds it.
...	The card content (kit_step_body()).
title_id	The id of the header title (the title_id of kit_step_head()), used by screen readers as the card name.

Details

If you do not want to manage which step is open yourself, use the module kit_stepcards_ui() / kit_stepcards_server().

Value

<section class="kit-stepcard">.

Examples

```
kit_step_card(
  kit_step_head(1, "        ", "current", title_id = "steptitle1"),
  kit_step_body(shiny::p("        "),
                shiny::actionButton("ok1", "   ")),
  title_id = "steptitle1"
)
```

---

kit_step_head *Step card header*

Description

Three states:

Usage

```
kit_step_head(
  k,
  title,
  state = c("current", "done", "locked"),
  summary = NULL,
  edit_id = NULL,
  title_id = NULL
)
```

Arguments

`k`	The step number.
`title`	The title of this step.
`state`	`"current"`, `"done"`, or `"locked"`.
`summary`	A one-sentence summary shown when done (for example ”5 · TDCYP · 2009 2011 ”, ”5 files · cohort TDCYP · 2009, 2011 (year)”). Not shown for the current step; a step not yet available always shows ” N−1 ” (”Finish step N−1 first”).
`edit_id`	The `actionLink()` id for ” ” (”Edit”) when done; `NULL` hides ” ”.
`title_id`	The id of the title (pass it to the `title_id` of `kit_step_card()`).

Details

- `"current"`: the current step, a lime round number, `aria-current="step"`.
- `"done"`: finished, a round (mint green), followed by a one-sentence summary, with ” ” (”Edit”) at the far right (when `edit_id` is given).
- `"locked"`: not yet available, gray, showing ” N−1 ” (”Finish step N−1 first”).

Next to the number there is text for screen readers: ” N ” (”Step N, current step:”).

Value

`<div class="kit-stephead is-current">` or similar.

Examples

```
kit_step_head(1, "        ", "done",
              summary = "5    ·   TDCYP", edit_id = "edit1")
kit_step_head(2, "    ", "locked")
```

---

kit_step_tabs_ui *Step tabs module*

Description

Two looks, one set of states (done ; not yet available gray plus the reason; run again removes the):

Usage

```
kit_step_tabs_ui(id, ..., groups = NULL, selected = NULL, aside = NULL)

kit_step_tabs_server(id, states = NULL, tips = NULL)
```

Arguments

`id`	The module id.
`...`	`shiny::tabPanel()`s (`value` is required).
`groups`	Groups (a list of `kit_navgroup()`); `NULL` means an ordered flow.
`selected`	The tab selected at the start (value).
`aside`	Tabs in a flow that are not a step of the flow (value).
`states`	A reactive (or function): the state of each tab, a named character vector or list. Values are `kit_status()` states (`"success"` and `"warning"` add a ; `"stale"` removes the), or directly `"done"`, `"locked"`, `"ready"`. Tabs not listed are left unchanged. With `NULL`, no states are sent.
`tips`	The hover text for tabs not yet available: a named character vector (or a reactive that returns one), or one sentence for all.

Details

- Ordered flow (kitsUI): do not pass `groups`; the `tabPanel()` names in `...` use circled numbers; `aside` tabs (for example ” ”, ”User manual”) are dashed and placed at the far right.
- Grouped, unnumbered (kitJoin): pass `groups` (a list of `kit_navgroup()`); the tab bar is `kit_navgroups()`, and the real tabs are a hidden `tabsetPanel()`.

The tabs' input is `input[["<id>-tabs"]]`; the `selected()` returned by the server is the same value.

Value

`kit_step_tabs_ui()`: the tab bar and the tabs.

`kit_step_tabs_server()`: a list,

- `selected`: a reactive, the current tab (value);
- `select(value)`: switches to a tab;
- `states`: a reactive, the states sent to the browser (the result of `kit_tab_states()`).

See Also

`kit_flow_tabs()`, `kit_navgroups()`, `kit_tab_states()` (low-level)

Examples

```
if (interactive()) {
  ui <- kit_page("KIT  ", NULL,
    kit_step_tabs_ui("nav",
      shiny::tabPanel("   ", value = "data", "…"),
      shiny::tabPanel("    ", value = "items", "…")
    )
  )
  server <- function(input, output, session) {
    kit_step_tabs_server("nav",
```

```
      states = shiny::reactive(c(data = "success", items = "locked")),
      tips = c(items = "       ")
    )
  }
  shiny::shinyApp(ui, server)
}
```

`kit_stepcards_ui` *Step card module*

Description

Use when a page has several steps that must be done in order: only one step is open at a time; finished steps collapse to one line (, summary, " " ("Edit")), and steps not yet available are gray. Each step ends with " " ("Yes, next step") (a regular button); pressing it marks the step done and opens the next unfinished step. Press " " on a finished step to go back.

Usage

```
kit_stepcards_ui(id, ..., next_label = "   ")

kit_stepcards_server(
  id,
  titles,
  summaries = list(),
  checks = list(),
  start = 1L
)
```

Arguments

`id`	The module id.
`...`	The content of each step (one tag or `tagList()` per step, in order).
`next_label`	The button text at the bottom of each step.
`titles`	The title of each step (character vector, length = number of steps).
`summaries`	The summary shown when each step is done: a list, one per step (a string, function, or reactive; `NULL` shows nothing).
`checks`	Checks before " " ("Yes, next step") is pressed: a list, one function per step, returning what is still needed (character vector; length 0 = can move on). While something is still needed, the step is not done, and " …" ("Still needed: …") is listed above the button (`kit_problems()`).
`start`	Which step is open at the start.

Details

`kit_stepcards_ui()` holds the content of each step (in order); `kit_stepcards_server()` gives each step's title, its summary when done, and the check before " " is pressed.

Value

`kit_stepcards_ui()`: a `tagList()` of `kit_step_card()`s.

`kit_stepcards_server()`: a list,

- `current`: a reactive, which step is open now (0 when all are done);
- `done`: a reactive, whether each step is done (logical vector);
- `open(k)`: opens step k (for example, back to step 1 when the data change);
- `reset()`: marks all steps as not done and opens step 1.

See Also

`kit_step_card()`, `kit_step_head()` (low-level)

Examples

```
if (interactive()) {
  ui <- kit_page("KIT  ", NULL,
    kit_stepcards_ui("steps",
      shiny::p("        "),
      shiny::p("    "),
      shiny::p("   ")
    )
  )
  server <- function(input, output, session) {
    steps <- kit_stepcards_server("steps",
      titles = c("        ", "    ", "   "),
      summaries = list("5   ", function() "45  ", "    ")
    )
  }
  shiny::shinyApp(ui, server)
}
```

---

kit_steps *Numbered list of steps (for the* `steps` *of* `kit_tab_guide()`*)*

Description

Note: this is the `<ol>` inside the guide box, not a step card; step cards are `kit_step_card()`.

Usage

```
kit_steps(...)
```

Arguments

...	One string (or character vector) per step.

Value

`<ol class="kit-steps">`.

Examples

```
kit_steps("  ", "  ID  ", "    ")
```

---

kit_tab_guide *Page guide: what this page does*

Description

A one-line guide at the top of each tab, collapsed by default: only the sentence " " ("What this page does:") is visible, with " " ("See steps ") next to it; opening it shows the steps, what you get, additional notes, and technical notes. No border, small gray text, a light line below.

Usage

```
kit_tab_guide(
  goal,
  steps,
  output,
  technical = NULL,
  note = NULL,
  diagram = NULL
)
```

Arguments

goal	One sentence, no more than 30 characters.
steps	The steps to follow: `kit_steps()` (3–6 points), or a character vector (turned into `kit_steps()` automatically).
output	What you get.
technical	Technical notes (for people who want to know how it works; collapsed one more level); `NULL` shows nothing.
note	Additional notes (small gray text); `NULL` shows nothing.
diagram	A diagram (`kit_diagram()`): placed outside the guide box, right after it, and visible even when the guide is collapsed; `NULL` shows nothing.

Value

Without a diagram, `<details class="kit-guide kit-guide-page">`; with a diagram, it is wrapped in `<div class="kit-guide kit-guide-wrap">`.

See Also

kit_steps(), kit_diagram()

Examples

```
kit_tab_guide(
  goal = "      ID      ",
  steps = kit_steps("    ", "  ID  ", "    "),
  output = "       CSV SAV "
)
```

---

kit_tab_states *State of each tab*

Description

Works out how each tab looks from the state of its status box (the `state` of kit_status()):

Usage

```
kit_tab_states(
  status,
  locked = character(0),
  locked_tips = NULL,
  tabset = "tabs"
)
```

Arguments

`status`	A named list or character vector: names are tab values, values are states. Tabs not listed are left unchanged.
`locked`	Tabs that cannot be used yet: a character vector of values, or a named logical.
`locked_tips`	The hover text for tabs that cannot be used yet: one string (the same for all), or a character vector named by value.
`tabset`	The tabset id (`tabsetPanel(id = )`).

Details

- `"locked"` (gray): tabs listed in `locked`. The hover text is `locked_tips`.
- `"done"` (with): the state is `"success"` or `"warning"` (or directly `"done"`).
- `"stale"` (removed): the state is `"stale"`.
- `"ready"`: anything else (`"idle"`, `"error"`, `"ready"`).

Pass the result to kit_send_tab_states() (or kit_watch_tab_states()) to send it to the browser.

Value

A list with one `list(tabset, value, state, tip)` per tab.

Examples

```
kit_tab_states(
  list(upload = "success", missing = "stale", join = "idle"),
  locked = "join", locked_tips = c(join = "      2   ")
)
```

`kit_table_scroll` *Put wide tables in here: they scroll sideways in their own box and do not break the layout*

Description

Put wide tables in here: they scroll sideways in their own box and do not break the layout

Usage

```
kit_table_scroll(..., tall = FALSE)
```

Arguments

`...`	Tables (for example `kit_static_table()` or `shiny::tableOutput()`).
`tall`	For long tables: fixed height (about 26rem), with the header fixed at the top.

Value

`<div class="kit-table-scroll">`.

Examples

```
kit_table_scroll(kit_static_table(data.frame(id = 1:3)))
```

kit_test_download *Press a download button in tests*

Description

The mock session of `shiny::testServer()` does not really register downloads (`registerDataObj()` does nothing), so `kit_download_button()` stores the download in `session$userData` instead; this function takes it from there and writes the file and responds the same way as when a browser presses download.

Usage

```
kit_test_download(session, id, write = TRUE)
```

Arguments

`session`	The `session` of `testServer()`.
`id`	The button id: inside a module, the id without namespace (`"dl"`); at the app level, the full id (`"stack-download_csv"`).
`write`	When `FALSE`, no file is written and no response is made; only the file name registered is checked (for example, downloads that are slow to draw a chart).

Value

`shiny::httpResponse()` (`status`, `content_type`, `headers`, `content$file`), with one more field `filename`: the file name registered. With `write = FALSE`, only `list(filename = )`.

Examples

```
session <- shiny::MockShinySession$new()
b <- kit_download_button(session, "dl", " ", "a.csv", function(file) writeLines("x", file))
res <- kit_test_download(session, "dl")
res$content_type
readLines(res$content$file)
```

kit_theme *KIT's bslib theme*

Description

Based on Bootswatch Brite (available from bslib 0.11.0), with KIT's colors; all text uses the rounded typeface Chiron GoRound TC, code uses JetBrains Mono. Both typefaces are downloaded by the browser from Google Fonts on demand (`local = FALSE`): CJK typefaces are split into hundreds of unicode-range pieces, so the server does not fetch every font file at startup.

Usage

```
kit_theme(brand = kit_current_brand())
```

Arguments

brand	The brand (kit_brand()); defaults to kit_current_brand() (KIT when not set).

Details

You usually do not need to call it yourself: kit_page() already uses it.

Colors come from the brand (kit_brand()): primary, success, navbar-bg, and link-color are the brand's primary; link-hover-color is primary_hover; body-bg is page; border-color is ink. info, warning, danger, and secondary are Bootstrap's own dark colors and do not change with the brand (the background colors of callouts and status boxes are in the CSS --nb-* variables, which do change with the brand). brand.yml is not read (bslib::bs_theme(brand = FALSE)): when the working directory or a parent has a _brand.yml (for example brand/_brand.yml), the theme does not change.

Value

A bslib::bs_theme() object.

See Also

kit_page(), use_kit(), kit_brand()

Examples

```
theme <- kit_theme()
bslib::theme_bootswatch(theme)
```

---

kit_unload_guard *Reminder before leaving the page*

Description

When there are results from this session (for example, after calibration), the browser asks once before the tab is closed or reloaded. Turn it on and off in the server based on the result: shiny::observe(kit_unload_guard(!is.null(result()))).

Usage

```
kit_unload_guard(on = TRUE, session = shiny::getDefaultReactiveDomain())
```

Arguments

on	TRUE turns it on, FALSE turns it off.
session	Defaults to the current session.

Value

on, invisibly.

Examples

```
if (interactive()) kit_unload_guard(TRUE)
```

---

kit_versions *Versions: kitDesign and the app's own*

Description

When reporting a problem, include both versions. The header's version line can include the kitDesign version: `kit_page(..., show_kitdesign = TRUE)` → "v0.5.0 · 2026-10-31 · kitDesign 0.1.0".

Usage

```
kit_versions(package = NULL)
```

Arguments

package	The app's package name (for example "kitJoin"); with NULL, only kitDesign. Versions are read the same way as kit_app_description().

Value

A named character vector, for example `c(kitDesign = "0.1.0", kitJoin = "0.5.0")`.

Examples

```
kit_versions()
```

---

kit_watch_tab_states *Send tab states to the browser whenever they change*

Description

`kit_watch_tab_states(session, shiny::reactive(kit_tab_states(...)))`.

Usage

```
kit_watch_tab_states(session, states)
```

Arguments

session	The Shiny session.
states	A reactive: the result of kit_tab_states().

Value

The shiny::observe() observer (invisibly).

Examples

```
if (interactive()) {
  server <- function(input, output, session) {
    kit_watch_tab_states(session, shiny::reactive(kit_tab_states(list(upload = "success"))))
  }
}
```

---

use_kit *KIT styles and browser-side scripts (an htmltools dependency)*

Description

Can go anywhere in the page (htmltools moves it into <head>). It contains:

Usage

```
use_kit()
```

Details

- kit-base.css: basic styles for the header, guide boxes, callouts, and so on (formerly kit_css(), kit_css_guide()).
- kit-brite.css: KIT's neo-brutalism styles (formerly kit_css_brite()).
- kit-design.css: the few rules kitDesign adds (header layout, dashed tabs in flow tabs, code box width).
- kit.js: collapsible sections, tab bars, tab states, scrolling to results, the reminder before leaving, and the " " ("Copy") button.

kit_page() already adds it; you only need to add use_kit() when you build the page yourself with bslib::page_*(), and pass theme = kit_theme() to the page.

Value

An htmltools::htmlDependency() object.

Examples

```
use_kit()
```

Index

as.tags.kit_check *(kit_check)*, 13
as.tags.kit_checklist *(kit_checklist)*, 14
as.tags.kit_code *(kit_code)*, 17
as.tags.kit_status *(kit_status)*, 61

bslib::bs_theme(), *74*

format.kit_check *(kit_check)*, 13
format.kit_checklist *(kit_checklist)*, 14
format.kit_code *(kit_code)*, 17
format.kit_status *(kit_status)*, 61

htmltools::as.tags(), *13*, *17*, *61*, *63*
htmltools::htmlDependency(), *36*, *76*
htmltools::tagList(), *18*

kit_abort, 4
kit_abort(), *28*, *30*, *35*, *55*
kit_advanced, 4
kit_advanced(), *43*
kit_app_description, 5
kit_app_description(), *44*, *75*
kit_app_link, 6
kit_app_meta, 7
kit_brand, 7
kit_brand(), *12*, *13*, *22*, *23*, *33*, *34*, *44*, *46*, *74*
kit_btn_class, 9
kit_button, 10
kit_button(), *9*, *53*, *57*
kit_callout, 11
kit_chart_colors, 11
kit_chart_colors(), *13*, *47*
kit_chart_neutral, 12
kit_chart_neutral(), *11*
kit_chart_sequential, 12
kit_chart_sequential(), *8*, *9*, *22*, *46*, *47*
kit_check, 13
kit_check(), *14*, *15*
kit_checklist, 14
kit_checklist(), *13*, *57*
kit_cloud_banner, 15
kit_cloud_banner(), *16*, *35*
kit_cloud_footer, 16
kit_cloud_footer(), *35*
kit_code, 17
kit_code(), *18*, *20*, *21*
kit_code_blocks, 18
kit_code_blocks(), *17*, *20*, *50*
kit_code_cols, 18
kit_code_cols(), *21*
kit_code_panel, 19
kit_code_panel(), *20*, *24*, *50*, *55*, *56*
kit_code_panel_server *(kit_code_panel_ui)*, 20
kit_code_panel_ui, 20
kit_code_panel_ui(), *17*, *19*, *40*
kit_code_wrap, 21
kit_code_wrap(), *17–19*
kit_colors, 22
kit_colors(), *8*, *22*
kit_current_brand, 22
kit_current_brand(), *8*, *9*, *12*, *22*, *33*, *34*, *44*, *46*, *74*
kit_detail, 23
kit_detail(), *19*, *24*, *56*
kit_details_row, 23
kit_details_row(), *19*, *20*
kit_dg_table, 24
kit_dg_table(), *25*
kit_diagram, 25
kit_diagram(), *70*, *71*
kit_dims, 25
kit_done_status, 26
kit_done_status(), *55*, *63*
kit_download_button, 27
kit_download_button(), *57*, *73*

kit_dt, 28
kit_error_status, 30
kit_error_status(), *4*, *63*
kit_files, 31
kit_fix_utf8_chr *(kit_files)*, 31
kit_fix_utf8_df *(kit_files)*, 31
kit_flow_tabs, 32
kit_flow_tabs(), *43*, *59*, *67*
kit_gallery, 33
kit_gz_path *(kit_files)*, 31
kit_header, 33
kit_header(), *8*, *22*, *36*, *44*
kit_is_cloud, 35
kit_is_cloud(), *16*
kit_is_user_error, 35
kit_is_valid_utf8_df *(kit_files)*, 31
kit_logo_dependency, 36
kit_logo_dependency(), *36*
kit_logo_file, 36
kit_logo_file(), *36*, *37*
kit_logo_uri, 37
kit_logo_uri(), *36*
kit_navgroup, 37
kit_navgroup(), *38*, *67*
kit_navgroups, 38
kit_navgroups(), *37*, *43*, *59*, *67*
kit_need_files, 39
kit_need_files(), *50*
kit_new_app, 39
kit_next_step, 40
kit_notify, 41
kit_notify(), *56*
kit_num, 42
kit_open_browser *(kit_files)*, 31
kit_open_details, 42
kit_open_details(), *5*
kit_page, 43
kit_page(), *8*, *9*, *22*, *33*, *40*, *74*, *76*
kit_plotly_fonts, 44
kit_plotly_fonts(), *46*, *47*
kit_plotly_style, 45
kit_plotly_style(), *8*, *12*, *13*, *22*, *45*
kit_preview_table, 48
kit_problems, 49
kit_problems(), *68*
kit_ready_line, 49
kit_ready_line(), *53*
kit_render_code_panel, 50
kit_render_code_panel(), *19*, *20*, *55*
kit_rows_note, 51
kit_rows_note(), *48*, *61*
kit_run_gate, 52
kit_run_gate(), *50*, *53*, *56*
kit_run_gate_server *(kit_run_gate_ui)*, 53
kit_run_gate_ui, 53
kit_run_gate_ui(), *40*
kit_run_tab_server *(kit_run_tab_ui)*, 54
kit_run_tab_ui, 54
kit_sav_safe_names *(kit_files)*, 31
kit_scope_bar, 57
kit_scroll_to, 58
kit_scroll_to(), *56*, *63*
kit_send_tab_states, 59
kit_send_tab_states(), *32*, *38*, *71*
kit_settings_line, 59
kit_settings_line(), *56*
kit_stale_status, 60
kit_stale_status(), *63*
kit_static_table, 61
kit_static_table(), *28*, *29*, *72*
kit_status, 61
kit_status(), *41*, *55*, *56*, *63*, *67*, *71*
kit_status_ui, 63
kit_status_ui(), *61–63*
kit_step_body, 64
kit_step_body(), *64*, *65*
kit_step_card, 64
kit_step_card(), *66*, *69*
kit_step_head, 65
kit_step_head(), *64*, *65*, *69*
kit_step_tabs_server *(kit_step_tabs_ui)*, 66
kit_step_tabs_ui, 66
kit_step_tabs_ui(), *40*
kit_stepcards_server *(kit_stepcards_ui)*, 68
kit_stepcards_server(), *65*
kit_stepcards_ui, 68
kit_stepcards_ui(), *65*
kit_steps, 69
kit_steps(), *70*, *71*
kit_tab_guide, 70
kit_tab_guide(), *25*, *55*, *69*
kit_tab_states, 71

kit_tab_states(), *56*, *59*, *67*, *76*
kit_table_scroll, 72
kit_table_scroll(), *29*
kit_test_download, 73
kit_test_download(), *28*
kit_theme, 73
kit_theme(), *8*, *9*, *22*, *45*
kit_truncate_bytes *(kit_files)*, 31
kit_unload_guard, 74
kit_versions, 75
kit_versions(), *7*
kit_watch_tab_states, 75
kit_watch_tab_states(), *71*

print.kit_check *(kit_check)*, 13
print.kit_checklist *(kit_checklist)*, 14
print.kit_code *(kit_code)*, 17
print.kit_status *(kit_status)*, 61

shiny::actionButton(), *10*
shiny::downloadButton(), *27*
shiny::httpResponse(), *73*
shiny::renderUI(), *50*
shiny::runApp(), *33*
shiny::testServer(), *73*

use_kit, 76
use_kit(), *28*, *74*