Package 'shiny.element'

November 4, 2026

Title 'Element Plus' Components for 'shiny'

Version 0.3.1

Description 'Element Plus' ('Vue' 3) components for 'shiny': forms, tables, trees, menus, dialog boxes and the rest, each reporting to the server and updated from it like any other 'shiny' input. 'Element Plus' and 'Vue' are bundled, so applications work offline.

License MIT + file LICENSE

Encoding UTF-8

Language en-GB

Depends R (>= 4.1)

Imports htmltools, shiny, jsonlite, uuid, stats, bslib, grDevices, utils, jquerylib

Suggests testthat (>= 3.0.0), shinyjs, shinyvalidate, shinytest2, fontawesome, callr, chromote, httpuv, pkgload, promises, withr, knitr, rmarkdown, DT, later

VignetteBuilder knitr

Config/testthat/edition 3

URL https://kaipingyang.github.io/shiny.element/, https://github.com/kaipingyang/shiny.element

BugReports https://github.com/kaipingyang/shiny.element/issues

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Kaiping Yang [aut, cre],
Element Plus team [ctb, cph] (Element Plus and its icons, bundled in inst/element-plus, MIT licence),
Evan You [ctb, cph] (Vue library bundled in inst/vue3, MIT licence)

Maintainer Kaiping Yang <1260146556@qq.com>

Repository CRAN

Date/Publication 2026-11-04 09:20:02 UTC

Contents

call_el . . . 5
call_vue . . . 7
df_to_cascader_options . . . 8
df_to_tree_data . . . 9
el . . . 9
element_plus_dependency . . . 11
el_affix . . . 11
el_alert . . . 13
el_anchor . . . 16
el_anchor_link . . . 18
el_aside . . . 19
el_autocomplete . . . 20
el_avatar . . . 24
el_avatar_group . . . 27
el_backtop . . . 29
el_badge . . . 31
el_breadcrumb . . . 33
el_breadcrumb_item . . . 35
el_button . . . 36
el_button_group . . . 39
el_calendar . . . 41
el_calendar_events . . . 46
el_calendar_output . . . 47
el_card . . . 48
el_carousel . . . 49
el_carousel_item . . . 52
el_cascader . . . 53
el_cascader_panel . . . 59
el_checkbox . . . 63
el_checkbox_group . . . 66
el_check_tag . . . 69
el_col . . . 71
el_collapse . . . 72
el_collapse_item . . . 74
el_color_picker . . . 75
el_color_picker_panel . . . 79
el_config_provider . . . 82
el_container . . . 84
el_date_picker . . . 85
el_date_picker_panel . . . 92
el_descriptions . . . 95
el_descriptions_item . . . 98
el_dialog . . . 99
el_divider . . . 103
el_drawer . . . 104
el_dropdown . . . 107

el_dropdown_item . . . 111
el_empty . . . 112
el_events . . . 114
el_feedback_close . . . 115
el_feedback_dependency . . . 116
el_footer . . . 116
el_form . . . 117
el_form_clear_validate . . . 122
el_form_field . . . 122
el_form_item . . . 124
el_form_reset . . . 125
el_form_validate . . . 126
el_header . . . 126
el_icon . . . 127
el_image . . . 128
el_image_viewer . . . 132
el_infinite_scroll . . . 134
el_input . . . 137
el_input_number . . . 142
el_input_otp . . . 146
el_input_tag . . . 149
el_layout_css_dependency . . . 154
el_link . . . 154
el_loading . . . 156
el_loading_close . . . 157
el_load_children . . . 158
el_locales . . . 159
el_locale_dependency . . . 160
el_main . . . 161
el_mention . . . 162
el_menu . . . 165
el_menu_item . . . 169
el_message . . . 171
el_message_box . . . 173
el_notification . . . 176
el_option . . . 178
el_page . . . 179
el_page_header . . . 181
el_pagination . . . 183
el_popconfirm . . . 186
el_popover . . . 189
el_progress . . . 193
el_radio_group . . . 196
el_rate . . . 199
el_result . . . 203
el_row . . . 205
el_rule . . . 206
el_scrollbar . . . 207

el_segmented . . . 210
el_select . . . 213
el_select_v2 . . . 221
el_skeleton . . . 228
el_skeleton_item . . . 230
el_slider . . . 231
el_space . . . 235
el_splitter . . . 237
el_splitter_panel . . . 239
el_statistic . . . 240
el_step . . . 244
el_steps . . . 245
el_switch . . . 247
el_table . . . 251
el_table_column . . . 261
el_table_data . . . 264
el_table_output . . . 265
el_table_row . . . 266
el_table_v2 . . . 267
el_table_v2_column . . . 272
el_tabs . . . 273
el_tab_pane . . . 276
el_tag . . . 277
el_text . . . 279
el_theme . . . 281
el_timeline . . . 282
el_timeline_item . . . 285
el_time_picker . . . 286
el_tooltip . . . 292
el_tour . . . 296
el_tour_step . . . 299
el_transfer . . . 300
el_tree . . . 304
el_tree_select . . . 310
el_tree_v2 . . . 314
el_upload . . . 318
el_upload_clear . . . 323
el_watermark . . . 324
el_widget . . . 326
flush_vue . . . 329
insert_el_tab . . . 330
JS . . . 331
render_vue_data . . . 332
template . . . 333
update_vue . . . 334
use_element . . . 336
vue_answer . . . 337
vue_app . . . 338

vue_component 341
vue_output 342
vue_store 344

Index **345**

`call_el` *Call a method of an Element component*

Description

`update_el_table()` and the other `update_el_*()` functions assign into the Vue instance's data, which reaches a component's props. Element also documents *methods* – `clearSelection()`, `setCheckedKeys()`, `validate()` – which are functions on the component and cannot be reached that way. `call_el()` invokes one: it is `call_vue()` with Element's references to table rows, upload files and tree nodes.

Usage

```
call_el(
  session = shiny::getDefaultReactiveDomain(),
  id,
  method,
  args = list(),
  result = TRUE,
  component = NULL
)
```

Arguments

`session`	Shiny session; the current one by default, as for `shiny::updateTextInput()`.
`id`	Component ID (un-namespaced).
`method`	Name of the Element method to call.
`args`	A list of arguments, passed positionally. A row of a table or a file of an upload is given by `el_table_row()` or `el_upload_file()`: the method needs the object itself.
`result`	Whether to report the return value as an input. Default `TRUE`. The input is `<id>_<method>`; an input of your own with that name – an `actionButton("car_next")` beside `call_el(session, "car", "next")` – would hear it too. Give `result = FALSE`, or another name, then.
`component`	Optional Element component name (`"ElTable"`) to look for under the component's id. Only needed when a component nests another of its own.

Details

A method that returns something reports it as `input$<id>_<method>`, with the method name in snake_case, matching how events are reported: `getCheckedKeys` arrives as `input$<id>_get_checked_keys`. The input is set with event priority, so asking twice and getting the same answer still fires an `observeEvent()` the second time.

A method that returns nothing reports `TRUE`, so an `observeEvent()` can still tell that it ran. Pass `result = FALSE` to send nothing back.

Each component's methods are listed in its own help page, under "Element methods".

Value

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

Examples

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

  ui <- el_page(
    el_tree(
      "tree",
      show_checkbox = TRUE,
      node_key = "id",
      checked = "apple",
      default_expand_all = TRUE,
      data = list(list(
        id = "fruit",
        label = "Fruit",
        children = list(
          list(id = "apple", label = "Apple"),
          list(id = "pear", label = "Pear")
        )
      ))
    ),
    el_button("clear", "Clear the ticks"),
    el_button("ask", "Which are ticked?"),
    verbatimTextOutput("answer")
  )

  server <- function(input, output, session) {
    # A command: setCheckedKeys() with an empty set
    observeEvent(input$clear, {
      call_el(session, "tree", "setCheckedKeys", list(list()))
    })

    # A method with a return value answers asynchronously
    observeEvent(input$ask, {
      call_el(session, "tree", "getCheckedKeys")
    })
    output$answer <- renderPrint(input$tree_get_checked_keys)
```

```
  }

  shinyApp(ui, server)
}
```

`call_vue` *Call a method of a Vue component from the server*

Description

`update_vue()` assigns data; a method is a function, reached with this. The method is looked up on the component the template renders – a library table's `clearSelection()`, a child component's own – or, with `component`, on the first component of that name under the id.

Usage

```
call_vue(
  session = shiny::getDefaultReactiveDomain(),
  id,
  method,
  args = list(),
  result = TRUE,
  component = NULL
)
```

Arguments

`session`	Shiny session; the current one by default, as for `shiny::updateTextInput()`.
`id`	The component's id (un-namespaced).
`method`	The method's name.
`args`	A list of arguments, passed in order.
`result`	Whether to report the return value. Default `TRUE`.
`component`	Name of the component to look for under `id`, when the template renders more than one.

Details

A method that returns something reports it as `input$<id>_<method>`, the method name in snake_case; one that returns nothing reports `TRUE`. The input has event priority, so the same answer twice still fires an `observeEvent()`. A promise reports what it resolves to.

Value

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

See Also

update_vue().

Examples

```
if (interactive()) {
  # inside a server function
  observeEvent(input$reset, call_vue(id = "form", method = "resetFields"))
}
```

`df_to_cascader_options` *Convert a data.frame with custom value/label columns to Element-UI Cascader options list*

Description

Convert a data.frame with custom value/label columns to Element-UI Cascader options list

Usage

```
df_to_cascader_options(df, value_cols, label_cols = NULL)
```

Arguments

df	Data frame with hierarchical columns
value_cols	Character vector of value column names (e.g. c("value1", "value2", ...))
label_cols	Character vector of label column names (e.g. c("label1", "label2", ...)), can be NULL or contain NA for levels without label

Value

Nested list for cascader options

Examples

```
df <- data.frame(
  province = c("Zhejiang", "Zhejiang", "Jiangsu"),
  city = c("Hangzhou", "Ningbo", "Nanjing"),
  stringsAsFactors = FALSE
)
df_to_cascader_options(df, c("province", "city"))

# Separate value and label columns
df$province_label <- paste(df$province, "Province")
df_to_cascader_options(df, c("province", "city"), c("province_label", NA))
```

df_to_tree_data *Build tree data from a data frame*

Description

Turns hierarchical columns into the nested node lists `el_tree()` expects, one level per column. Keys are built by joining a row's values down to that level, so a label repeated under different parents still gets a unique key.

Usage

```
df_to_tree_data(df, cols, sep = "/")
```

Arguments

`df`	A data frame.
`cols`	Column names, outermost level first.
`sep`	Separator used when joining values into a key.

Value

A list of nodes.

Examples

```
df <- data.frame(
  region = c("North", "North", "South"),
  city = c("Leeds", "York", "Bath"),
  stringsAsFactors = FALSE
)
df_to_tree_data(df, c("region", "city"))
```

el *Element Plus tags, for markup inside a component*

Description

A tag generator for every Element tag, as `tags$p` is for HTML: `el$button(type = "primary", "Go")` writes `<el-button type="primary">`. Nothing more – no Vue instance, no Shiny input, no id.

Usage

```
el
```

Details

An `<el-*>` tag becomes an Element component only when a Vue instance compiles it, so these belong where one already does:

- the `markup` of `el_widget()`, when building a component of your own;
- a `template()` or a component's `slots`;
- a table column's `cell` in `el_table()`;
- the trigger of a wrapper – `el_tooltip()`, `el_popover()`, `el_popconfirm()`.

Anywhere else – the top level of a page, or inside the markup-only containers (`el_tabs()`, `el_collapse()`, `el_dialog()`, `el_drawer()`, `el_row()`, `el_container()`) – nothing compiles them and they show as bare text; the browser console says so. There, use the component functions: `el_button()`, `el_tag()` and the rest.

A logical attribute is a Vue boolean: `el$button(link = TRUE)` writes `<el-button link>` and `plain = FALSE` writes `:plain="false"` – as text, `link="TRUE"`, Element Plus would refuse it.

Mounting a Vue instance over such markup automatically is deliberately not done: it would rebuild every component and Shiny input inside it, leaving them on screen but disconnected from the server.

Value

A named list of tag-generating functions, one per Element Plus tag.

Examples

```
names(el)

# As a wrapper's trigger, compiled by the tooltip's own instance
el_tooltip("hint", el$button(type = "primary", "Hover me"), content = "Help")

# In a table cell, once per row
el_table(
  "tasks",
  data = data.frame(task = c("Draft", "Review"), done = c(TRUE, FALSE)),
  columns = list(
    list(prop = "task", label = "Task"),
    list(
      label = "State",
      cell = el$tag(
        ":type" = "scope.row.done ? 'success' : 'info'",
        "{{ scope.row.done ? 'done' : 'open' }}"
      )
    )
  )
)

# As the markup of a component of your own
el_widget(
  "me",
  markup = el$avatar(":size" = "size", "{{ initials }}"),
  data = list(size = 48, initials = "KY")
```

```
)
```

`element_plus_dependency` *Element Plus Dependency*

Description

Element Plus's script and stylesheets – its components, its dark mode variables and its display classes (`hidden-xs-only`, ...) – and its icons, which are components registered on every app by name.

Usage

```
element_plus_dependency(offline = TRUE)
```

Arguments

`offline`	Serve Element Plus from the copy bundled with this package (the default) instead of the unpkg CDN. The bundled files are byte-identical to the CDN's. A runtime CDN dependency leaves the page blank on an intranet, offline, or whenever unpkg is unreachable, so the local copy is the safer default; pass `FALSE` to trade that for a smaller deployment bundle.

Value

A list of htmlDependency objects.

Examples

```
element_plus_dependency()
element_plus_dependency(offline = FALSE)
```

`el_affix` *Element Plus Affix*

Description

Content that stays fixed to the top or bottom of the viewport once scrolled to it.

Usage

```
el_affix(
  ...,
  id = NULL,
  offset = NULL,
  position = NULL,
  target = NULL,
  z_index = NULL,
  teleported = NULL,
  append_to = NULL,
  width = NULL,
  slots = NULL,
  events = NULL,
  on = NULL
)

update_el_affix(
  session = shiny::getDefaultReactiveDomain(),
  id,
  offset = NULL,
  position = NULL,
  z_index = NULL,
  target = NULL,
  teleported = NULL,
  append_to = NULL
)
```

Arguments

...	Its content: any Shiny UI. Components of this package are folded into this one's Vue instance, as `el_button_group()` folds its buttons.
id	Component ID. Auto-generated if `NULL`.
offset	Offset distance. Element Plus's `offset` (number).
position	Position of affix. Element Plus's `position` ('top' | 'bottom').
target	Target container (CSS selector). Element Plus's `target` (string).
z_index	`z-index` of affix. Element Plus's `z-index` (number).
teleported	Whether affix element is teleported, if `true` it will be teleported to where `append-to` sets. Element Plus's `teleported` (boolean).
append_to	Which element the affix element appends to. Element Plus's `append-to` (CSS-Selector / HTMLElement).
width	Component width, as a CSS unit.
slots	Named list of Element slot contents. A scoped slot is written with `template()`.
events	Element's events to report besides those reported unasked, by name: `events = "node_drop"` reports `input$<id>_node_drop`. The component's are listed under "Shiny inputs", and by `el_events()`; a name it does not have is an error.

on	Handlers of your own, for an event not reported or to send something else: a named list of JS() functions, one per event – Element's, or a DOM event with Vue's modifiers ("keyup.enter"). Each is called with report and the event's arguments; report(name, value) sets input$<id>_<name>. See el_widget().
session	Shiny session; the current one by default, as for shiny::updateTextInput().

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
input$<id>_change	unasked	triggers when fixed state changed
input$<id>_scroll	events = "scroll"	list(scrollTop, fixed), at most every 200 ms

The same list as el_events("el_affix"), which says how an event's arguments travel.

Element methods

Callable with call_el(): update(), updateRoot().

Updating from the server

update_el_affix() changes the component from the server: every argument of el_affix() that can change once it is drawn, under the same name. One left NULL stays as it is; NA returns it to Element's default.

update_el_affix() is called for its side effect and returns NULL invisibly.

Examples

```
el_affix(el_button("top", "Stays on top"), offset = 120)
```

---

el_alert *Element Plus Alert*

Description

An inline alert banner with optional close button. Fires a Shiny input when the user closes it.

Usage

```
el_alert(
  id = NULL,
  title = "",
  description = NULL,
  type = "info",
  closable = TRUE,
  close_text = "",
  show_icon = FALSE,
  center = FALSE,
  effect = "light",
  width = NULL,
  slots = NULL,
  on = NULL,
  session = NULL
)

update_el_alert(
  session = shiny::getDefaultReactiveDomain(),
  id,
  title = NULL,
  type = NULL,
  description = NULL,
  closable = NULL,
  close_text = NULL,
  show_icon = NULL,
  center = NULL,
  effect = NULL
)
```

Arguments

id	Alert ID. Auto-generated UUID if NULL.
title	Main alert title text.
description	Optional secondary description text. When supplied, the icon is rendered at the larger size.
type	Alert type: "info" (default), "success", "warning", "error".
closable	Whether to show a close button. Default TRUE.
close_text	Custom text for the close button. "" for the default cross.
show_icon	Whether to display the type icon. Default FALSE.
center	Whether to centre the content. Default FALSE.
effect	Visual effect: "light" (default) or "dark".
width	Component width, as a CSS unit – "200px", "50%", or a number taken as pixels. Element's own markup carries it, so it behaves like the width argument of a Shiny input.

slots — Named list of Element slot contents, such as `list(title = shiny::tags$b("Bold"))`. A shiny.element component given here is absorbed rather than nested. For a scoped slot, write the template with `template()`.

on — Handlers of your own, for an event not reported or to send something else: a named list of `JS()` functions, one per event – Element's, or a DOM event with Vue's modifiers (`"keyup.enter"`). Each is called with `report` and the event's arguments; `report(name, value)` sets `input$<id>_<name>`. See `el_widget()`.

session — In `el_alert()`, deprecated: inside a module, wrap `id` in `ns()`, as for any Shiny input; a session given here namespaces `id` once more, with a warning. In `update_el_alert()`, the Shiny session, the current one by default, as for `shiny::updateTextInput()`.

Value

An `htmltools` tagList with a Vue-managed alert component.

Shiny inputs

Input	Reported	Value
`input$<id>_close`	unasked	fires when the user closes the alert

The same list as `el_events("el_alert")`, which says how an event's arguments travel.

Updating from the server

Server-side update for `el_alert()`.

Every other argument of `el_alert()` that can change once it is drawn is an argument here too, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_alert()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_alert("al1", "Operation successful", type = "success", show_icon = TRUE)
el_alert("al2", "Warning!", description = "Please review.", type = "warning")
if (interactive()) {
  # inside a server function
  observeEvent(input$go, {
    update_el_alert(session, "hint", title = "Saved", type = "success")
  })
}
```

el_anchor *Element Plus Anchor*

Description

Links to sections of the page, the one in view marked as the page scrolls.

Usage

```
el_anchor(
  id = NULL,
  links = list(),
  container = NULL,
  offset = NULL,
  bound = NULL,
  duration = NULL,
  marker = NULL,
  type = NULL,
  direction = NULL,
  select_scroll_top = NULL,
  width = NULL,
  slots = NULL,
  events = NULL,
  on = NULL
)

update_el_anchor(
  session = shiny::getDefaultReactiveDomain(),
  id,
  offset = NULL,
  bound = NULL,
  duration = NULL,
  marker = NULL,
  type = NULL,
  direction = NULL,
  select_scroll_top = NULL,
  container = NULL
)
```

Arguments

id	Anchor ID. Auto-generated if NULL.
links	The links, each an el_anchor_link() – or a list(title =, href = "#section"), each with optional children, a list of links one level down.
container	A CSS selector for the element that scrolls, when it is not the page.
offset	Offset of the scroll position, in pixels.

bound — Distance from the top at which a section counts as reached, in pixels. Default 15.

duration — Duration of the scroll, in milliseconds. Default 300.

marker — Whether to show the marker beside the current link.

type — "default" or "underline".

direction — "vertical" (the default) or "horizontal".

select_scroll_top — Whether a link counts as current once its section is scrolled to the top.

width — Component width, as a CSS unit.

slots — Named list of Element slot contents.

events — Element's events to report besides those reported unasked, by name: events = "node_drop" reports input$<id>_node_drop. The component's are listed under "Shiny inputs", and by el_events(); a name it does not have is an error.

on — Handlers of your own, for an event not reported or to send something else: a named list of JS() functions, one per event – Element's, or a DOM event with Vue's modifiers ("keyup.enter"). Each is called with report and the event's arguments; report(name, value) sets input$<id>_<name>. See el_widget().

session — Shiny session; the current one by default, as for shiny::updateTextInput().

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
input$<id>	unasked	the href of the current link, as the page scrolls
input$<id>_click	events = "click"	the href of the link clicked

The same list as el_events("el_anchor"), which says how an event's arguments travel.

Element methods

Callable with call_el(): scrollTo(href).

Updating from the server

update_el_anchor() changes the component from the server: every argument of el_anchor() that can change once it is drawn, under the same name. One left NULL stays as it is; NA returns it to Element's default.

update_el_anchor() is called for its side effect and returns NULL invisibly.

Examples

```
el_anchor(
  "toc",
  links = list(
    list(title = "Basic usage", href = "#basic"),
    list(
      title = "API",
      href = "#api",
      children = list(
        list(title = "Attributes", href = "#attributes")
      )
    )
  )
)
```

---

el_anchor_link *A link of* el_anchor()

Description

Element Plus's el-anchor-link, for el_anchor(links =).

Usage

```
el_anchor_link(title, href, ...)
```

Arguments

title	The link's text.
href	Where it points: "#section".
...	Links one level down, each an el_anchor_link().

Value

A link, for el_anchor(links =).

See Also

Other items: el_breadcrumb_item(), el_carousel_item(), el_collapse_item(), el_descriptions_item(), el_dropdown_item(), el_menu_item(), el_option(), el_skeleton_item(), el_step(), el_tab_pane(), el_table_column(), el_table_v2_column(), el_timeline_item(), el_tour_step()

Examples

```
el_anchor(
  "an",
  links = list(
    el_anchor_link("Basic Usage", "#basic-usage"),
    el_anchor_link(
      "API",
      "#api",
      el_anchor_link("Attributes", "#attributes"),
      el_anchor_link("Events", "#events")
    )
  )
)
```

---

el_aside *Element Plus Aside*

Description

Element Plus Aside

Usage

```
el_aside(..., width = "300px", style = NULL, class = NULL)
```

Arguments

`...`	Content.
`width`	Aside width. Defaults to `"300px"`, as in Element Plus, which sets it as the CSS variable `--el-<part>-<size>`.
`style`	Extra inline style.
`class`	Extra CSS classes.

Value

A Shiny UI element.

Examples

```
el_aside("Navigation")
el_aside(width = "200px", el_radio_group("nav", choices = c(Home = "h")))
```

el_autocomplete *Element Plus Autocomplete*

Description

A text input that suggests as you type.

Usage

```
el_autocomplete(
  id = NULL,
  value = "",
  suggestions = NULL,
  remote = FALSE,
  fetch_suggestions = NULL,
  placeholder = NULL,
  clearable = NULL,
  disabled = NULL,
  value_key = NULL,
  debounce = NULL,
  placement = NULL,
  trigger_on_focus = NULL,
  select_when_unmatched = NULL,
  highlight_first_item = NULL,
  hide_loading = NULL,
  icon = NULL,
  prefix_icon = NULL,
  suffix_icon = NULL,
  label = NULL,
  name = NULL,
  popper_class = NULL,
  label_position = c("top", "left", "right"),
  label_width = NULL,
  label_suffix = NULL,
  required = FALSE,
  error = NULL,
  show_message = TRUE,
  inline_message = FALSE,
  append_to = NULL,
  aria_label = NULL,
  fit_input_width = NULL,
  loop_navigation = NULL,
  popper_options = NULL,
  popper_style = NULL,
  show_arrow = NULL,
  teleported = NULL,
  width = NULL,
```

```
  slots = NULL,
  events = NULL,
  on = NULL,
  session = NULL
)

update_el_autocomplete(
  session = shiny::getDefaultReactiveDomain(),
  id,
  value = NULL,
  suggestions = NULL,
  placeholder = NULL,
  disabled = NULL,
  label = NULL,
  error = NULL,
  clearable = NULL,
  value_key = NULL,
  debounce = NULL,
  placement = NULL,
  trigger_on_focus = NULL,
  select_when_unmatched = NULL,
  highlight_first_item = NULL,
  hide_loading = NULL,
  icon = NULL,
  prefix_icon = NULL,
  suffix_icon = NULL,
  name = NULL,
  popper_class = NULL,
  append_to = NULL,
  aria_label = NULL,
  fit_input_width = NULL,
  loop_navigation = NULL,
  popper_options = NULL,
  popper_style = NULL,
  show_arrow = NULL,
  teleported = NULL
)
```

Arguments

`id`	Input ID. Auto-generated if `NULL`.
`value`	Initial text.
`suggestions`	Suggestions to offer, as a character vector or a list of `list(value =, ...)`. Filtered in the browser on what has been typed. For suggestions that come from the server, use `remote = TRUE`.
`remote`	Ask the server for suggestions as the user types, as Element's `fetch-suggestions` asks a function: the text arrives as `input$<id>_query`, and `update_el_autocomplete()` with `suggestions` answers it – the list shows what the server sent.

`fetch_suggestions`
: `JS()` function `function(queryString, callback)` that calls `callback(results)`, to fetch in the browser instead.

`placeholder`
: Placeholder text.

`clearable`
: Whether to show a clear button.

`disabled`
: Whether the input is disabled.

`value_key`
: Field of a suggestion object to display. Default `"value"`.

`debounce`
: Debounce while typing, in milliseconds. Default `300`.

`placement`
: Where the list appears: `"bottom-start"` (default), `"bottom-end"`, `"top-start"`, `"top-end"`.

`trigger_on_focus`
: Whether to suggest as soon as the input is focused. Default `TRUE`.

`select_when_unmatched`
: Whether to fire `select` when nothing matched.

`highlight_first_item`
: Whether to preselect the first suggestion.

`hide_loading`
: Whether to hide the loading spinner.

`icon`, `prefix_icon`, `suffix_icon`
: Icon classes.

`label`
: A label shown with the component, as Shiny's inputs have: text or a tag. `NULL`, the default, shows none. It is the component's accessible name too – tied to it with `for` where the component has a native input that takes the id `<id>-input`, else with `aria-labelledby`.

`name`
: Native `name` attribute.

`popper_class`
: Extra class name for the suggestion list.

`label_position`
: Where the label sits, as `el_form()`'s `label_position`: `"top"` (the default, as Shiny's labels sit), or beside the component, its text aligned `"left"` or `"right"` – which shows once `label_width` gives the labels a common width.

`label_width`
: Width of a label beside the component, as a CSS unit, so that several line up. Element's `label-width`.

`label_suffix`
: Text after the label, such as `":"`. Element's `label-suffix`.

`required`
: Draw Element's red asterisk before the label. It marks the field; it does not check it – shinyvalidate or `el_form()` does that.

`error`
: An error message shown under the component in Element's style, the field framed in red. Element's `error`.

`show_message`, `inline_message`
: Whether `error`'s message is shown, and whether beside the component rather than under it. Element's `show-message` and `inline-message`.

`append_to`
: Which select dropdown appends to. Element Plus's `append-to` (CSSSelector / HTMLElement).

`aria_label`
: Native `aria-label` attribute. Element Plus's `aria-label` (string).

`fit_input_width`
: Whether the width of the dropdown is the same as the input. Element Plus's `fit-input-width` (boolean).

`loop_navigation`
: Whether keyboard navigation loops from end to start. Element Plus's `loop-navigation` (boolean).

`popper_options`
: Popper.js parameters. Element Plus's `popper-options` (object).

`popper_style`
: Custom style for autocomplete's dropdown. Element Plus's `popper-style` (string / object).

`show_arrow`
: Whether the dropdown has an arrow. Element Plus's `show-arrow` (boolean).

`teleported`
: Whether select dropdown is teleported to the body. Element Plus's `teleported` (boolean).

`width`
: Component width, as a CSS unit.

`slots`
: Named list of Element slot contents, such as `list(title = shiny::tags$b("Bold"))`. A shiny.element component given here is absorbed rather than nested. For a scoped slot, write the template with `template()`.

`events`
: Element's events to report besides those reported unasked, by name: `events = "node_drop"` reports `input$<id>_node_drop`. The component's are listed under "Shiny inputs", and by `el_events()`; a name it does not have is an error.

`on`
: Handlers of your own, for an event not reported or to send something else: a named list of `JS()` functions, one per event – Element's, or a DOM event with Vue's modifiers (`"keyup.enter"`). Each is called with `report` and the event's arguments; `report(name, value)` sets `input$<id>_<name>`. See `el_widget()`.

`session`
: In `el_autocomplete()`, deprecated: inside a module, wrap `id` in `ns()`, as for any Shiny input; a session given here namespaces `id` once more, with a warning. In `update_el_autocomplete()`, the Shiny session, the current one by default, as for `shiny::updateTextInput()`.

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
`input$<id>`	unasked	the current text
`input$<id>_query`	unasked	with `remote = TRUE`, the text to suggest for; answer with `update_el_autocomp`
`input$<id>_select`	unasked	triggers when a suggestion is clicked
`input$<id>_change`	`events = "change"`	triggers when the icon inside Input value change
`input$<id>_blur`	`events = "blur"`	triggers when Input blurs
`input$<id>_clear`	`events = "clear"`	triggers when the Input is cleared by clicking the clear button
`input$<id>_focus`	`events = "focus"`	triggers when Input focuses
`input$<id>_input`	`events = "input"`	triggers when the Input value change

The same list as `el_events("el_autocomplete")`, which says how an event's arguments travel.

Element methods

Callable with `call_el()`:

- `focus()` – focus the input

Updating from the server

Server-side update for `el_autocomplete()`.

Every other argument of `el_autocomplete()` that can change once it is drawn is an argument here too, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_autocomplete()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_autocomplete("city", suggestions = c("Beijing", "Shanghai", "Shenzhen"))

el_autocomplete(
  "city",
  suggestions = c("Beijing", "Shanghai"),
  placeholder = "Where to?",
  clearable = TRUE,
  width = 260
)
if (interactive()) {
  # inside a server function
  observeEvent(input$country, {
    update_el_autocomplete(
      session,
      "city",
      suggestions = cities_of(input$country)
    )
  })
}
```

`el_avatar` *Element Plus Avatar*

Description

A user avatar, from an image, an icon, or text.

Usage

```
el_avatar(
  id = NULL,
  content = NULL,
  src = NULL,
  icon = NULL,
```

```
  size = NULL,
  shape = NULL,
  fit = NULL,
  src_set = NULL,
  alt = NULL,
  width = NULL,
  class = NULL,
  style = NULL,
  slots = NULL,
  events = NULL,
  on = NULL,
  session = NULL
)

update_el_avatar(
  session = shiny::getDefaultReactiveDomain(),
  id,
  content = NULL,
  src = NULL,
  icon = NULL,
  size = NULL,
  shape = NULL,
  fit = NULL,
  src_set = NULL,
  alt = NULL
)
```

Arguments

`id`	Avatar ID. Auto-generated if `NULL`.
`content`	Text shown when there is no `src` or `icon`, such as initials.
`src`	Image URL.
`icon`	Element icon class, such as `"el-icon-user-solid"`.
`size`	Size: `"large"`, `"default"` or `"small"`; `NULL` follows the form or the page.
`shape`	`"circle"` (default) or `"square"`.
`fit`	How an image fills the avatar: `"cover"` (default), `"fill"`, `"contain"`, `"none"` or `"scale-down"`.
`src_set`	Candidate image sources, as a `srcset` string.
`alt`	Alternative text for the image.
`width`	Component width, as a CSS unit.
`class, style`	Extra classes and inline style on the avatar, as Element passes them to its root.
`slots`	Named list of Element slot contents, such as `list(title = shiny::tags$b("Bold"))`. A shiny.element component given here is absorbed rather than nested. For a scoped slot, write the template with `template()`.

events	Element's events to report besides those reported unasked, by name: `events = "node_drop"` reports `input$<id>_node_drop`. The component's are listed under "Shiny inputs", and by `el_events()`; a name it does not have is an error.
on	Handlers of your own, for an event not reported or to send something else: a named list of `JS()` functions, one per event – Element's, or a DOM event with Vue's modifiers (`"keyup.enter"`). Each is called with `report` and the event's arguments; `report(name, value)` sets `input$<id>_<name>`. See `el_widget()`.
session	In `el_avatar()`, deprecated: inside a module, wrap `id` in `ns()`, as for any Shiny input; a session given here namespaces `id` once more, with a warning. In `update_el_avatar()`, the Shiny session, the current one by default, as for `shiny::updateTextInput()`.

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
`input$<id>_error`	unasked	trigger when image load error.

The same list as `el_events("el_avatar")`, which says how an event's arguments travel.

Updating from the server

Server-side update for `el_avatar()`.

Every other argument of `el_avatar()` that can change once it is drawn is an argument here too, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_avatar()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_avatar("me", src = "https://example.org/face.png")
el_avatar("initials", content = "KY", shape = "square", size = 40)
el_avatar("anon", icon = "el-icon-user-solid")
if (interactive()) {
  # inside a server function
  observeEvent(input$sign_in, {
    update_el_avatar(session, "me", src = user_photo())
  })
}
```

el_avatar_group *Element Plus Avatar Group*

Description

Avatars overlapping in a row, the overflow collapsed into a count.

Usage

```
el_avatar_group(
  ...,
  id = NULL,
  size = NULL,
  shape = NULL,
  collapse_avatars = NULL,
  collapse_avatars_tooltip = NULL,
  max_collapse_avatars = NULL,
  effect = NULL,
  placement = NULL,
  popper_class = NULL,
  popper_style = NULL,
  collapse_class = NULL,
  collapse_style = NULL,
  width = NULL,
  slots = NULL,
  on = NULL
)

update_el_avatar_group(
  session = shiny::getDefaultReactiveDomain(),
  id,
  size = NULL,
  shape = NULL,
  collapse_avatars = NULL,
  collapse_avatars_tooltip = NULL,
  max_collapse_avatars = NULL,
  effect = NULL,
  placement = NULL,
  popper_class = NULL,
  popper_style = NULL,
  collapse_class = NULL,
  collapse_style = NULL
)
```

Arguments

...	Its content: any Shiny UI. Components of this package are folded into this one's Vue instance, as `el_button_group()` folds its buttons.

id	Component ID. Auto-generated if NULL.
size	Control the size of avatars in this avatar-group. Element Plus's size (number / 'large' | 'default' | 'small').
shape	Control the shape of avatars in this avatar-group. Element Plus's shape ('circle' | 'square').
collapse_avatars	Whether to collapse avatars. Element Plus's collapse-avatars (boolean).
collapse_avatars_tooltip	Whether show all collapsed avatars when mouse hover text of the collapse-avatar. To use this, collapse-avatars must be true. Element Plus's collapse-avatars-tooltip (boolean).
max_collapse_avatars	The max avatars number to be shown. To use this, collapse-avatars must be true. Element Plus's max-collapse-avatars (number).
effect	Tooltip theme, built-in theme: dark / light. Element Plus's effect ('dark' | 'light' / string).
placement	Placement of tooltip. Element Plus's placement (enum).
popper_class	Custom class name for tooltip. Element Plus's popper-class (string).
popper_style	Custom style for tooltip. Element Plus's popper-style (string / object).
collapse_class	Custom class name for the collapse-avatar. Element Plus's collapse-class (string).
collapse_style	Custom style for the collapse-avatar. Element Plus's collapse-style (string / object).
width	Component width, as a CSS unit.
slots	Named list of Element slot contents. A scoped slot is written with template().
on	Handlers of your own, for an event not reported or to send something else: a named list of JS() functions, one per event – Element's, or a DOM event with Vue's modifiers ("keyup.enter"). Each is called with report and the event's arguments; report(name, value) sets input$<id>_<name>. See el_widget().
session	Shiny session; the current one by default, as for shiny::updateTextInput().

Value

A Shiny UI element.

Shiny inputs

None: it reports nothing.

Updating from the server

update_el_avatar_group() changes the component from the server: every argument of el_avatar_group() that can change once it is drawn, under the same name. One left NULL stays as it is; NA returns it to Element's default.

update_el_avatar_group() is called for its side effect and returns NULL invisibly.

Examples

```
el_avatar_group(
  el_avatar(src = "https://example.com/a.png"),
  el_avatar("B"),
  collapse_avatars = TRUE
)
```

`el_backtop` *Element Plus Back to Top*

Description

A button that appears once the page has been scrolled down, and returns it to the top when clicked.

Usage

```
el_backtop(
  id = NULL,
  content = NULL,
  target = NULL,
  visibility_height = NULL,
  right = NULL,
  bottom = NULL,
  width = NULL,
  slots = NULL,
  events = NULL,
  on = NULL,
  session = NULL
)

update_el_backtop(
  session = shiny::getDefaultReactiveDomain(),
  id,
  visibility_height = NULL,
  right = NULL,
  bottom = NULL,
  target = NULL
)
```

Arguments

`id`	Button ID. Auto-generated if `NULL`.
`content`	Contents of the button. `NULL` for Element's own arrow icon.
`target`	CSS selector of the element that scrolls. `NULL` for the page.
`visibility_height`	Scroll distance in pixels before the button appears. Default `200`.

right	Distance from the right edge, in pixels. Default 40.
bottom	Distance from the bottom edge, in pixels. Default 40.
width	Component width, as a CSS unit.
slots	Named list of Element slot contents, such as list(title = shiny::tags$b("Bold")). A shiny.element component given here is absorbed rather than nested. For a scoped slot, write the template with template().
events	Element's events to report besides those reported unasked, by name: events = "node_drop" reports input$<id>_node_drop. The component's are listed under "Shiny inputs", and by el_events(); a name it does not have is an error.
on	Handlers of your own, for an event not reported or to send something else: a named list of JS() functions, one per event – Element's, or a DOM event with Vue's modifiers ("keyup.enter"). Each is called with report and the event's arguments; report(name, value) sets input$<id>_<name>. See el_widget().
session	In el_backtop(), deprecated: inside a module, wrap id in ns(), as for any Shiny input; a session given here namespaces id once more, with a warning. In update_el_backtop(), the Shiny session, the current one by default, as for shiny::updateTextInput().

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
input$<id>_click	unasked	triggers when click.

The same list as el_events("el_backtop"), which says how an event's arguments travel.

Updating from the server

Server-side update for el_backtop().

Every other argument of el_backtop() that can change once it is drawn is an argument here too, under the same name. One left NULL stays as it is; NA returns it to Element's default.

update_el_backtop() is called for its side effect and returns NULL invisibly.

Examples

```
el_backtop("top")
el_backtop("top", visibility_height = 100, right = 20, bottom = 20)

# Scrolling a panel rather than the page
el_backtop("panel_top", target = "#report")
if (interactive()) {
  # inside a server function
  observeEvent(input$compact, {
```

```
    update_el_backtop(session, "top", right = 10, bottom = 10)
  })
}
```

---

el_badge *Element Plus Badge*

Description

Wraps any content with a numeric badge or red dot in the top-right corner. When used without content, renders a standalone badge element.

Usage

```
el_badge(
  ...,
  value = NULL,
  max = NULL,
  is_dot = FALSE,
  hidden = FALSE,
  type = NULL,
  id = NULL,
  show_zero = TRUE,
  color = NULL,
  offset = NULL,
  badge_style = NULL,
  badge_class = NULL,
  on = NULL
)

update_el_badge(
  session = shiny::getDefaultReactiveDomain(),
  id,
  value = NULL,
  max = NULL,
  is_dot = NULL,
  hidden = NULL,
  type = NULL
)
```

Arguments

...	Content to wrap (e.g. a button or icon).
value	Badge value: number or string. Ignored when is_dot = TRUE.
max	Maximum numeric value to display. When value > max the badge shows "<max>+". Only applies when value is numeric.
is_dot	Show a small dot instead of a number. Default FALSE.

hidden	Whether to hide the badge. Default FALSE.
type	Badge colour type: NULL (red, default), "primary", "success", "warning", "info", "danger".
id	Give the badge an id and update_el_badge() can change it – a count of unread messages, say. A component inside is then folded into the badge's Vue instance, as for el_tooltip(): it keeps reporting, but is reached through update_vue() on the badge's id.
show_zero	Whether a value of 0 is shown. Default TRUE.
color	Background colour of the badge.
offset	Offset of the badge, c(x, y) in pixels.
badge_style, badge_class	Extra CSS – a string or a named list – and class names for the badge.
on	Handlers of your own, for an event not reported or to send something else: a named list of JS() functions, one per event – Element's, or a DOM event with Vue's modifiers ("keyup.enter"). Each is called with report and the event's arguments; report(name, value) sets input$<id>_<name>. See el_widget().
session	Shiny session; the current one by default, as for shiny::updateTextInput().

Value

An htmltools tag, or with an id a Shiny UI element.

Updating from the server

Server-side update for an el_badge() given an id.

update_el_badge() is called for its side effect and returns NULL invisibly.

Examples

```
el_badge(el_button("btn1", "Messages"), value = 5)
el_badge(el_button("btn2", "Alerts"), value = 200, max = 99)
el_badge(el_button("btn3", "Updates"), is_dot = TRUE)

# Updated from the server: update_el_badge(session, "unread", value = 7)
el_badge(el_button("inbox", "Inbox"), value = 3, id = "unread")
if (interactive()) {
  # inside a server function
  observe(update_el_badge(
    session,
    "unread",
    value = unread_count(),
    hidden = unread_count() == 0
  ))
}
```

`el_breadcrumb` *Element Plus Breadcrumb*

Description

A trail of links showing where a page sits.

Usage

```
el_breadcrumb(
  id = NULL,
  items = list(),
  separator = NULL,
  separator_icon = NULL,
  width = NULL,
  slots = NULL,
  on = NULL,
  session = NULL
)

update_el_breadcrumb(
  session = shiny::getDefaultReactiveDomain(),
  id,
  items = NULL,
  separator = NULL,
  separator_icon = NULL
)
```

Arguments

`id`	Breadcrumb ID. Auto-generated if `NULL`.
`items`	The trail, a list of `el_breadcrumb_item()`s – or of `list(label =, to =)`. `to` is optional and makes that step a link; the last step is usually plain text.
`separator`	Separator character. Default `"/"`.
`separator_icon`	Icon component of icon separator. Element Plus's `separator-icon` (string / Component). An icon's name, such as `"Search"`.
`width`	Component width, as a CSS unit.
`slots`	Named list of Element slot contents, such as `list(title = shiny::tags$b("Bold"))`. A shiny.element component given here is absorbed rather than nested. For a scoped slot, write the template with `template()`.
`on`	Handlers of your own, for an event not reported or to send something else: a named list of `JS()` functions, one per event – Element's, or a DOM event with Vue's modifiers (`"keyup.enter"`). Each is called with `report` and the event's arguments; `report(name, value)` sets `input$<id>_<name>`. See `el_widget()`.

session — In `el_breadcrumb()`, deprecated: inside a module, wrap `id` in `ns()`, as for any Shiny input; a session given here namespaces `id` once more, with a warning. In `update_el_breadcrumb()`, the Shiny session, the current one by default, as for `shiny::updateTextInput()`.

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
`input$<id>`	unasked	the `label` of the step last clicked

The same list as `el_events("el_breadcrumb")`, which says how an event's arguments travel.

Steps without a `to` are reported too, so a breadcrumb can drive navigation inside a Shiny app without any routing.

Updating from the server

Server-side update for `el_breadcrumb()`.

Every other argument of `el_breadcrumb()` that can change once it is drawn is an argument here too, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_breadcrumb()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_breadcrumb(
  "trail",
  items = list(
    list(label = "Home"),
    list(label = "Reports"),
    list(label = "March")
  )
)

# An arrow instead of a slash
el_breadcrumb(
  "trail",
  items = list(list(label = "Home"), list(label = "Detail")),
  separator_icon = "ArrowRight"
)
if (interactive()) {
  # inside a server function
  observeEvent(input$open_detail, {
    update_el_breadcrumb(
      session,
      "trail",
```

```
      items = list(
        list(label = "Home"),
        list(label = "Detail")
      )
    )
  })
}
```

`el_breadcrumb_item` *A step of* `el_breadcrumb()`

Description

Element Plus's `el-breadcrumb-item`, for `el_breadcrumb(items =)`.

Usage

```
el_breadcrumb_item(label, to = NULL, replace = NULL)
```

Arguments

`label`	The step's text.
`to`	Where the step links to; the last step usually has none.
`replace`	Navigate without leaving a history record.

Value

A step, for `el_breadcrumb(items =)`.

See Also

Other items: `el_anchor_link()`, `el_carousel_item()`, `el_collapse_item()`, `el_descriptions_item()`, `el_dropdown_item()`, `el_menu_item()`, `el_option()`, `el_skeleton_item()`, `el_step()`, `el_tab_pane()`, `el_table_column()`, `el_table_v2_column()`, `el_timeline_item()`, `el_tour_step()`

Examples

```
el_breadcrumb(
  "bc",
  items = list(
    el_breadcrumb_item("Home", to = "/"),
    el_breadcrumb_item("Promotion list")
  )
)
```

el_button *Element Plus Button with Vue Instance*

Description

Creates an Element Plus button with Vue instance, supporting all Element Plus button variants including `plain`, `round`, `circle`, and `loading` states.

Usage

```
el_button(
  id = NULL,
  label = "Button",
  type = NULL,
  size = NULL,
  plain = NULL,
  round = NULL,
  circle = FALSE,
  loading = FALSE,
  task = FALSE,
  disabled = FALSE,
  icon = NULL,
  native_type = "button",
  autofocus = FALSE,
  auto_insert_space = NULL,
  bg = NULL,
  color = NULL,
  dark = NULL,
  dashed = NULL,
  link = NULL,
  loading_icon = NULL,
  tag = NULL,
  text = NULL,
  class = NULL,
  style = NULL,
  width = NULL,
  slots = NULL,
  on = NULL,
  session = NULL
)

update_el_button(
  session = shiny::getDefaultReactiveDomain(),
  id,
  label = NULL,
  type = NULL,
  size = NULL,
```

```
  plain = NULL,
  round = NULL,
  loading = NULL,
  disabled = NULL,
  circle = NULL,
  icon = NULL,
  autofocus = NULL,
  auto_insert_space = NULL,
  bg = NULL,
  color = NULL,
  dark = NULL,
  dashed = NULL,
  link = NULL,
  loading_icon = NULL,
  tag = NULL,
  text = NULL
)
```

Arguments

id	Button ID. Auto-generated UUID if NULL.
label	Button text, or tags – tagList("Next", el_icon("ArrowRight", class = "el-icon--right")) – drawn as they are. Ignored (and defaults to "") when circle = TRUE.
type	Button type: "default", "primary", "success", "warning", "danger", "info", "text". NULL leaves it to a config provider's button settings, or Element's default.
size	Size: "large", "default" or "small"; NULL follows the form or the page.
plain	Whether to use the plain (hollow) style.
round	Whether to use rounded corners. plain and round left NULL are off, or a config provider's el_config_provider(button =).
circle	Whether to render as a circle button (icon only, no label). Default FALSE.
loading	Whether to show loading spinner. Disables click while active.
task	A task button, as bslib::input_task_button(): clicked, it shows Element's loading state at once, in the browser, until the server has handled the click – the observers it runs are done – or, bound to an shiny::ExtendedTask with bslib::bind_task_button(), until the task is. input$<id> counts the clicks, as for any button. Default FALSE.
disabled	Whether the button is disabled. Default FALSE.
icon	Either an Element icon class name such as "el-icon-search", which Element renders itself and update_el_button() can change, or a tag (for example from el_icon()), which is inserted as button content.
native_type	HTML native button type: "button" (default), "submit", "reset".
autofocus	Whether the button takes focus on page load. Default FALSE.

auto_insert_space
: Automatically insert a space between two chinese characters(this will only take effect when the text length is 2 and all characters are in Chinese.). Element Plus's auto-insert-space (boolean).

bg
: Determine whether the text button background color is always on. Element Plus's bg (boolean).

color
: Custom button color, automatically calculate hover and active color. Works with link/text buttons since. Element Plus's color (string).

dark
: Dark mode, which automatically converts color to dark mode colors. Element Plus's dark (boolean).

dashed
: Determine whether it's a dashed button. Element Plus's dashed (boolean).

link
: Determine whether it's a link button. Element Plus's link (boolean).

loading_icon
: Customize loading icon component. Element Plus's loading-icon (string / Component). An icon's name, such as "Search".

tag
: Custom element tag. Element Plus's tag (string / Component). An icon's name, such as "Search".

text
: Determine whether it's a text button. Element Plus's text (boolean).

class, style
: Extra classes and inline style on the button, as Element passes them to its root.

width
: Component width, as a CSS unit – "200px", "50%", or a number taken as pixels. Element's own markup carries it, so it behaves like the width argument of a Shiny input.

slots
: Named list of Element slot contents, such as list(title = shiny::tags$b("Bold")). A shiny.element component given here is absorbed rather than nested. For a scoped slot, write the template with template().

on
: Handlers of your own, for an event not reported or to send something else: a named list of JS() functions, one per event – Element's, or a DOM event with Vue's modifiers ("keyup.enter"). Each is called with report and the event's arguments; report(name, value) sets input$<id>_<name>. See el_widget().

session
: In el_button(), deprecated: inside a module, wrap id in ns(), as for any Shiny input; a session given here namespaces id once more, with a warning. In update_el_button(), the Shiny session, the current one by default, as for shiny::updateTextInput().

Value

An htmltools tagList with a Vue-managed button component.

Shiny inputs

Input	Reported	Value
input$<id>	unasked	the number of clicks, as actionButton() reports it

The same list as el_events("el_button"), which says how an event's arguments travel.

Counted as shiny::actionButton() counts: 0 on load, and only while neither disabled nor loading is TRUE. It carries the same class, so observeEvent() and req() treat 0 as not yet clicked.

Updating from the server

Server-side update for el_button(). Supports all visual states including size, plain, round, and loading.

Every other argument of el_button() that can change once it is drawn is an argument here too, under the same name. One left NULL stays as it is; NA returns it to Element's default.

update_el_button() is called for its side effect and returns NULL invisibly.

Examples

```
# Basic usage
el_button("btn_primary", "Primary", type = "primary")

# Shiny app example
if (interactive()) {
  library(shiny)
  library(shiny.element)
  ui <- el_page(
    el_button("btn1", "Primary", type = "primary"),
    verbatimTextOutput("count")
  )
  server <- function(input, output, session) {
    output$count <- renderPrint(input$btn1)
  }
  shinyApp(ui, server)
}
if (interactive()) {
  # inside a server function
  observeEvent(input$go, {
    update_el_button(session, "save", loading = TRUE)
  })
}
```

---

el_button_group *Element Plus Button Group*

Description

Buttons joined into one bar, as Element's el-button-group draws them: shared borders, rounded only at the ends. Each el_button() inside keeps reporting its own clicks as input$<its id>.

Usage

```
el_button_group(
  ...,
  id = NULL,
  size = NULL,
  type = NULL,
  direction = NULL,
```

```
  width = NULL,
  on = NULL
)

update_el_button_group(
  session = shiny::getDefaultReactiveDomain(),
  id,
  size = NULL,
  type = NULL,
  direction = NULL
)
```

Arguments

...	Buttons: el_button()s, or raw el$button() tags.
id	Group ID. Auto-generated if NULL.
size, type	The size and type of every button in the group, as Element Plus's group hands them down: size "large", "default" or "small"; type "primary", "success", "warning", "danger" or "info".
direction	"horizontal" (default) or "vertical".
width	Component width, as a CSS unit.
on	Handlers of your own, for an event not reported or to send something else: a named list of JS() functions, one per event – Element's, or a DOM event with Vue's modifiers ("keyup.enter"). Each is called with report and the event's arguments; report(name, value) sets input$<id>_<name>. See el_widget().
session	Shiny session; the current one by default, as for shiny::updateTextInput().

Details

The buttons are folded into the group's Vue instance rather than nested (see el_tooltip()): Element styles the group through its direct children, which a button with a host of its own would not be. So update_el_button() cannot reach them; set their fields with update_vue() on the group's id.

Value

A Shiny UI element.

Updating from the server

update_el_button_group() changes the component from the server: every argument of el_button_group() that can change once it is drawn, under the same name. One left NULL stays as it is; NA returns it to Element's default.

update_el_button_group() is called for its side effect and returns NULL invisibly.

Examples

```
el_button_group(
  el_button("prev", "Previous", icon = "ArrowLeft", type = "primary"),
  el_button("next", "Next", type = "primary")
)
```

`el_calendar` *Element Plus Calendar*

Description

A month of days to pick one from, or a range of weeks to show – and, with `events`, a month planner: each day shows its events, which the user can add, edit, delete and drag to another day (`editable`). Days show times, groups (`calendars`) and a "+N more" beyond `visible_event_count`; the dialog and the popovers take Element's components of your own through slots.

Usage

```
el_calendar(
  id = NULL,
  value = NULL,
  range = NULL,
  label = NULL,
  label_position = c("top", "left", "right"),
  label_width = NULL,
  label_suffix = NULL,
  required = FALSE,
  error = NULL,
  show_message = TRUE,
  inline_message = FALSE,
  controller_type = NULL,
  formatter = NULL,
  width = NULL,
  slots = NULL,
  session = NULL,
  events = NULL,
  editable = FALSE,
  event_labels = NULL,
  calendars = NULL,
  visible_event_count = NULL,
  use_detail_popup = FALSE,
  first_day_of_week = NULL,
  workweek = FALSE
)

update_el_calendar(
  session = shiny::getDefaultReactiveDomain(),
```

```
  id,
  value = NULL,
  range = NULL,
  label = NULL,
  error = NULL,
  controller_type = NULL,
  formatter = NULL,
  events = NULL,
  insert = NULL,
  replace = NULL,
  delete = NULL,
  editable = NULL,
  event_labels = NULL,
  calendars = NULL,
  visible_event_count = NULL,
  use_detail_popup = NULL,
  first_day_of_week = NULL,
  workweek = NULL
)
```

Arguments

id	Calendar ID (auto-generated if NULL)
value	Bound value (Date/string/number)
range	Date range, c("YYYY-MM-DD", "YYYY-MM-DD")
label	A label shown with the component, as Shiny's inputs have: text or a tag. NULL, the default, shows none. It is the component's accessible name too – tied to it with for where the component has a native input that takes the id <id>-input, else with aria-labelledby.
label_position	Where the label sits, as el_form()'s label_position: "top" (the default, as Shiny's labels sit), or beside the component, its text aligned "left" or "right" – which shows once label_width gives the labels a common width.
label_width	Width of a label beside the component, as a CSS unit, so that several line up. Element's label-width.
label_suffix	Text after the label, such as ":". Element's label-suffix.
required	Draw Element's red asterisk before the label. It marks the field; it does not check it – shinyvalidate or el_form() does that.
error	An error message shown under the component in Element's style, the field framed in red. Element's error.
show_message, inline_message	Whether error's message is shown, and whether beside the component rather than under it. Element's show-message and inline-message.
controller_type	How the header switches month and year: "button" (the default) or "select". Element Plus's controller-type.

formatter — With `controller_type = "select"`, a `JS()` function `function(value, type)` returning the label of each option. Element Plus's `formatter`.

width — Component width, as a CSS unit – `"200px"`, `"50%"`, or a number taken as pixels.

slots — Named list of slot contents. Element's: `dateCell` renders one day – Element hands the template `date` and `data`, so write it with `template()` – and replaces the one that draws the events; call `eventsOn(data.day)` in it for the day's events. The events layer's own, for Element's components in what it draws: `event`, the content of an event's tag (scope `{ event, day }`); `eventForm`, the dialog's fields in place of ours (scope `{ form, labels, calendars }`: bind a field with `` `v-model` = "form.location" `` and it travels with the event in _add and _update); `eventDetail`, the popover of `use_detail_popup` (scope `{ event, labels, calendars }`). Write a scope of your own with `template(scope = "{ event }")`.

session — Deprecated. Inside a module, wrap `id` in `ns()`, as for any Shiny input; a session given here namespaces `id` once more, with a warning.

events — Element's events to report besides those reported unasked, by name: `events = "node_drop"` reports `input$<id>_node_drop`. The component's are listed under "Shiny inputs", and by `el_events()`; a name it does not have is an error.

editable — Whether the user can change the events: double-click a day, or drag across several, to add one; click an event to edit or delete it in a dialog; drag it to another day to move it (its times and a span's length kept). An event with `isReadOnly` stays as it is.

event_labels — The dialog's and popovers' words, to change any of: a named list of `add`, `edit` (the dialog's titles), `title`, `allday`, `date`, `end`, `calendar`, `type`, `body` (its fields), `save`, `delete`, `cancel` (its buttons), `more` (the link to a day's hidden events, `{n}` their number) – in Chinese, `list(add = "\u65b0\u5efa\u65e5\u7a0b", save = "\u4fdd\u5b58", more = "\u8fd8\u6709 {n} \u9879")`.

calendars — Groups of events, toastui's: a data.frame, or a list of rows, with `id`, which an event names as its `calendarId`; optionally `name`, shown in the dialog and the popover; `type` or `color`, for its events with none of their own; `isVisible = FALSE`, its events hidden.

visible_event_count — How many events a day shows; the rest are behind a "+N more" that lists them in a popover. `NULL` shows them all, scrolling. toastui's `visibleEventCount`.

use_detail_popup — Whether a click on an event the user cannot edit shows it in a popover: its title, when, calendar and body, or the `eventDetail` slot. toastui's `useDetailPopup`.

first_day_of_week — The day the week starts on, 1 (Monday) to 7 (Sunday), as Element UI's `firstDayOfWeek`. `NULL` keeps Element Plus's, which is Sunday.

workweek — Whether to hide Saturday and Sunday, as toastui's `workweek`.

insert — Events to add, each with its `id`.

replace — Events to put in place of those with the same `id`.

delete — The `id`s of events to delete.

Details

The server owns the events, as toastui's calendar has it: what the user does arrives as a request – `input$<id>_add`, `_update`, `_delete` – and the calendar changes when the server answers with `update_el_calendar()` (`insert`, `replace`, `delete`, or all of them with `events`). So the server can check a change, give a new event its id from a database, or refuse. Without Shiny – a static page – the calendar applies the user's changes itself.

Value

A Shiny UI element.

Shiny inputs

Input	When	Value
`input$<id>`	on load and when a day is picked	the day, `"YYYY-MM-DD"`
`input$<id>_dates`	on load and when another month is shown	`list(current, start, end)`, Dates: the da
`input$<id>_click`	an event is clicked	the event, a list with its days as Dates and its
`input$<id>_add`	the user saves a new event (`editable`)	the event asked for, without an `id`: `list(dat`
`input$<id>_update`	the user saves an edit or drops an event on another day	`list(event, changes)`: the event as it is, a
`input$<id>_delete`	the user deletes an event	the event

The requests change nothing by themselves: answer them with `update_el_calendar()`, as the example does.

Updating from the server

Server-side update for `el_calendar()`: the selected day, the range, and every other argument that can change once the calendar is drawn, under the same name. One left `NULL` stays as it is; `NA` returns a prop to Element's default.

`events` replaces all the events; `insert`, `replace` and `delete` change a few, found by `id`, and send only those – one of the four per call. `calendars` replaces the groups: an `isVisible = FALSE` hides a group's events, as toastui's `cal_proxy_toggle()` does.

`update_el_calendar()` is called for its side effect and returns `NULL` invisibly.

Examples

```
# The default day cell
el_calendar("cal")

# Your own, with whatever Element hands the template
el_calendar(
  "cal",
  slots = list(
    dateCell = template(
      htmltools::HTML("<p>{{ data.day.slice(8) }}</p>"),
      slot = "dateCell",
      scope = "{date, data}"
```

```
    )
  )
)
# Basic usage
el_calendar(id = "calendar1", value = Sys.Date())

# With date range
el_calendar(id = "calendar2", range = c("2025-01-01", "2025-01-31"))

# A planner whose events the server keeps, rendered as an output: each
# request changes them, and the calendar is rendered again -- patched,
# only the events sent
if (interactive()) {
  library(shiny)
  library(shiny.element)
  ui <- el_page(el_calendar_output("plan"))
  server <- function(input, output, session) {
    events <- reactiveVal(data.frame(
      id = 1:2,
      date = Sys.Date() + c(0, 3),
      title = c("Standup", "Review"),
      type = c("primary", "warning")
    ))
    output$plan <- render_el_calendar(
      el_calendar(events = events(), editable = TRUE)
    )
    observeEvent(input$plan_add, {
      new <- input$plan_add
      events(rbind(
        events(),
        data.frame(
          id = max(events()$id) + 1L,
          date = new$date,
          title = new$title,
          type = new$type
        )
      ))
    })
    observeEvent(input$plan_update, {
      d <- events()
      i <- d$id == input$plan_update$event$id
      for (k in intersect(names(input$plan_update$changes), names(d))) {
        d[[k]][i] <- input$plan_update$changes[[k]]
      }
      events(d)
    })
    observeEvent(input$plan_delete, {
      events(events()[events()$id != input$plan_delete$id, ])
    })
  }
  shinyApp(ui, server)
}
if (interactive()) {
```

```
  # inside a server function
  observeEvent(input$go, {
    update_el_calendar(session, "cal", value = "2026-06-01")
  })
  update_el_calendar(session, "cal", controller_type = "select")
}
```

`el_calendar_events` *The events a calendar shows*

Description

What the browser holds, as R: the events last rendered with `render_el_calendar()` or sent with `update_el_calendar()`, with every event `update_el_calendar()` has since inserted, replaced or deleted. The user's requests (`input$<id>_add`, ...) are not in it until the server answers them. A reactive read.

Usage

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

Arguments

`session`	The Shiny session, the current one by default.
`id`	The calendar's id, the output's.

Value

The events, as given; `NULL` for a calendar the server has not rendered or updated.

See Also

`render_el_calendar()`, `update_el_calendar()`.

Examples

```
if (interactive()) {
  library(shiny)
  ui <- el_page(el_calendar_output("plan"), verbatimTextOutput("n"))
  server <- function(input, output, session) {
    output$plan <- render_el_calendar(el_calendar(
      events = data.frame(id = 1, date = Sys.Date(), title = "Review")
    ))
    # the events the calendar shows, read back as R
    output$n <- renderText(nrow(el_calendar_events(id = "plan")))
  }
  shinyApp(ui, server)
}
```

`el_calendar_output` *A calendar as a Shiny output*

Description

The calendar of an app whose events the server reads – from a database, a folder, a computation – as toastui's `calendarOutput()` and `renderCalendar()`: the page holds `el_calendar_output()`, the server renders `el_calendar()` into it with its events. Rendering again with new events patches the calendar in place: the month shown and an open dialog stay, and only what changed is sent.

Usage

```
el_calendar_output(outputId, width = "100%", loading = TRUE)

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

Arguments

`outputId`	The output's id.
`width`	The calendar's width, as a CSS unit.
`loading`	Whether Element's loading mask covers the calendar while Shiny recalculates it, in place of Shiny fading the output.
`expr`	An expression returning `el_calendar()`, given no `id`.
`env, quoted`	As for `shiny::renderUI()`.

Details

The output's id is the calendar's: `input$<id>` is the day picked and `input$<id>_click`, `_dates`, `_add`, `_update`, `_delete` are as for `el_calendar()`; `update_el_calendar()` reaches the calendar by it, and `el_calendar_events()` reads the events it shows.

Value

`el_calendar_output()`, a tag; `render_el_calendar()`, a render function.

See Also

`el_calendar()`, `update_el_calendar()`, `el_calendar_events()`.

Examples

```
if (interactive()) {
  library(shiny)
  archive <- data.frame(
    date = Sys.Date() - c(9, 6, 2),
    title = c("", "", "Validated"),
    body = c("Raw data", "Archive", "Validated data"),
```

```
    color = c("lightgrey", "#EED5B7", "#E9C46B")
  )
  ui <- el_page(el_calendar_output("snapshot"), verbatimTextOutput("picked"))
  server <- function(input, output, session) {
    output$snapshot <- render_el_calendar(
      el_calendar(value = max(archive$date), events = archive)
    )
    output$picked <- renderPrint(input$snapshot_click$date)
  }
  shinyApp(ui, server)
}
```

`el_card` *Element Plus Card*

Description

A card container with optional header and body. Supports `always`, `hover`, and `never` shadow modes.

Usage

```
el_card(
  ...,
  header = NULL,
  body_style = NULL,
  shadow = NULL,
  footer = NULL,
  header_class = NULL,
  body_class = NULL,
  footer_class = NULL,
  class = NULL,
  style = NULL
)
```

Arguments

`...`	Content placed inside the card body.
`header`	Header content (string or tag). `NULL` for no header.
`body_style`	CSS for the card body, a string or a named list. `NULL` for default.
`shadow`	Shadow display trigger: `"always"`, `"hover"` or `"never"`. `NULL`, the default, is Element's `"always"`, or a config provider's el_config_provider(card =).
`footer`	Footer content (string or tag). `NULL` for no footer.
`header_class, body_class, footer_class`	Extra class names for the header, the body and the footer.
`class, style`	Extra classes and inline style on the card itself, as Element passes them to its root.

Value

An `htmltools` tag.

Examples

```
el_card(shiny::tags$p("Card body text."), header = "My Card")
el_card(shiny::tags$p("No shadow."), shadow = "never")
```

`el_carousel` *Element Plus Carousel*

Description

A slideshow of items, horizontal or vertical.

Usage

```
el_carousel(
  id = NULL,
  items = list(),
  height = "300px",
  initial_index = 0,
  autoplay = TRUE,
  interval = 3000,
  trigger = "hover",
  indicator_position = NULL,
  arrow = "hover",
  type = NULL,
  loop = TRUE,
  direction = "horizontal",
  card_scale = NULL,
  motion_blur = NULL,
  pause_on_hover = NULL,
  width = NULL,
  slots = NULL,
  on = NULL,
  session = NULL
)

update_el_carousel(
  session = shiny::getDefaultReactiveDomain(),
  id,
  active = NULL,
  autoplay = NULL,
  interval = NULL,
  height = NULL,
  initial_index = NULL,
```

```
  trigger = NULL,
  indicator_position = NULL,
  arrow = NULL,
  loop = NULL,
  direction = NULL
)
```

Arguments

id	Carousel ID (auto-generated if NULL).
items	A list of slides, each an el_carousel_item() – or a list with content (a tag, tagList or string) and optionally name, used as the value reported when that slide is showing, and label, shown on its indicator. A slide's content may hold this package's components: they are folded into the carousel's own Vue instance and keep reporting their inputs.
height	Slide height, e.g. "300px".
initial_index	Index of the slide shown first, 0-based.
autoplay	Cycle through the slides on a timer.
interval	Milliseconds between slides when autoplay is on.
trigger	What switches slides when an indicator is used: "hover" (default) or "click".
indicator_position	"outside", "none", or NULL for inside.
arrow	When to show the arrows: "hover" (default), "always" or "never".
type	"card" for the stacked card layout, or NULL for plain.
loop	Return to the first slide after the last.
direction	"horizontal" (default) or "vertical".
card_scale	When type is card, scaled size of secondary cards. Element Plus's card-scale (number).
motion_blur	Infuse dynamism and smoothness into the carousel. Element Plus's motion-blur (boolean).
pause_on_hover	Pause autoplay when hover. Element Plus's pause-on-hover (boolean).
width	Component width, as a CSS unit – "200px", "50%", or a number taken as pixels. Element's own markup carries it, so it behaves like the width argument of a Shiny input.
slots	Named list of Element slot contents, such as list(title = shiny::tags$b("Bold")). A shiny.element component given here is absorbed rather than nested. For a scoped slot, write the template with template().
on	Handlers of your own, for an event not reported or to send something else: a named list of JS() functions, one per event – Element's, or a DOM event with Vue's modifiers ("keyup.enter"). Each is called with report and the event's arguments; report(name, value) sets input$<id>_<name>. See el_widget().
session	In el_carousel(), deprecated: inside a module, wrap id in ns(), as for any Shiny input; a session given here namespaces id once more, with a warning. In update_el_carousel(), the Shiny session, the current one by default, as for shiny::updateTextInput().
active	Index of the slide to show, 0-based.

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
`input$<id>`	unasked	the index of the slide showing, from 0
`input$<id>_name`	unasked	its `name`, if it has one

The same list as `el_events("el_carousel")`, which says how an event's arguments travel.

Element methods

Callable with `call_el()`:

- `next()` – Switch to the next slide
- `prev()` – Switch to the previous slide
- `setActiveItem()` – Manually switch slide

Updating from the server

`update_el_carousel()` changes the component from the server.

Every other argument of `el_carousel()` that can change once it is drawn is an argument here too, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_carousel()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_carousel(
  id = "banner",
  height = "200px",
  items = list(
    list(name = "one", content = shiny::tags$h3("First slide")),
    list(name = "two", content = shiny::tags$h3("Second slide")),
    list(name = "three", content = shiny::tags$h3("Third slide"))
  )
)

# Card layout, switching on click rather than hover
el_carousel(
  id = "cards",
  type = "card",
  trigger = "click",
  height = "180px",
  items = lapply(1:4, function(i) list(content = paste("Card", i)))
)
if (interactive()) {
  # inside a server function
```

```
  observeEvent(input$go, {
    update_el_carousel(session, "banner", active = 2)
  })
}
```

`el_carousel_item` *A slide of* `el_carousel()`

Description

Element Plus's `el-carousel-item`, for `el_carousel(items =)`.

Usage

```
el_carousel_item(..., name = NULL, label = NULL, style = NULL, class = NULL)
```

Arguments

`...`	The slide's content.
`name`	The slide's value, `input$<id>` while it is showing.
`label`	Text of the slide's indicator.
`style, class`	The slide's own inline style and classes – a height of its own in a carousel of `height = "auto"`, as upstream's example sets it.

Value

A slide, for `el_carousel(items =)`.

See Also

Other items: `el_anchor_link()`, `el_breadcrumb_item()`, `el_collapse_item()`, `el_descriptions_item()`, `el_dropdown_item()`, `el_menu_item()`, `el_option()`, `el_skeleton_item()`, `el_step()`, `el_tab_pane()`, `el_table_column()`, `el_table_v2_column()`, `el_timeline_item()`, `el_tour_step()`

Examples

```
el_carousel(
  "car",
  items = list(
    el_carousel_item(htmltools::tags$h3("1"), name = "first"),
    el_carousel_item(htmltools::tags$h3("2"), name = "second")
  )
)
```

`el_cascader` *Element Plus Cascader*

Description

Pick a path through nested options – a region, then a country, then a city – from a dropdown of side-by-side columns.

Usage

```
el_cascader(
  id = NULL,
  options = list(),
  value = NULL,
  placeholder = "Please select",
  props = NULL,
  clearable = FALSE,
  filterable = FALSE,
  disabled = FALSE,
  size = NULL,
  show_all_levels = TRUE,
  collapse_tags = FALSE,
  separator = " / ",
  debounce = 300,
  popper_class = NULL,
  filter_method = NULL,
  before_filter = NULL,
  label = NULL,
  label_position = c("top", "left", "right"),
  label_width = NULL,
  label_suffix = NULL,
  required = FALSE,
  error = NULL,
  show_message = TRUE,
  inline_message = FALSE,
  clear_icon = NULL,
  collapse_tags_tooltip = NULL,
  effect = NULL,
  empty_values = NULL,
  fallback_placements = NULL,
  fit_input_width = NULL,
  height = NULL,
  item_size = NULL,
  max_collapse_tags = NULL,
  max_collapse_tags_tooltip_height = NULL,
  persistent = NULL,
  placement = NULL,
```

```
  popper_append_to_body = NULL,
  popper_style = NULL,
  show_checked_strategy = NULL,
  tag_effect = NULL,
  tag_type = NULL,
  teleported = NULL,
  validate_event = NULL,
  value_on_clear = NULL,
  virtual_scroll = NULL,
  width = NULL,
  slots = NULL,
  level_inputs = NULL,
  events = NULL,
  on = NULL,
  session = NULL
)

update_el_cascader(
  session = shiny::getDefaultReactiveDomain(),
  id,
  options = NULL,
  value = NULL,
  placeholder = NULL,
  clearable = NULL,
  filterable = NULL,
  disabled = NULL,
  label = NULL,
  error = NULL,
  props = NULL,
  size = NULL,
  show_all_levels = NULL,
  collapse_tags = NULL,
  separator = NULL,
  debounce = NULL,
  popper_class = NULL,
  filter_method = NULL,
  before_filter = NULL,
  clear_icon = NULL,
  collapse_tags_tooltip = NULL,
  effect = NULL,
  empty_values = NULL,
  fallback_placements = NULL,
  fit_input_width = NULL,
  height = NULL,
  item_size = NULL,
  max_collapse_tags = NULL,
  max_collapse_tags_tooltip_height = NULL,
  persistent = NULL,
```

```
  placement = NULL,
  popper_append_to_body = NULL,
  popper_style = NULL,
  show_checked_strategy = NULL,
  tag_effect = NULL,
  tag_type = NULL,
  teleported = NULL,
  validate_event = NULL,
  value_on_clear = NULL,
  virtual_scroll = NULL
)
```

Arguments

id	Cascader ID (auto-generated if NULL)
options	Nested options, each `list(value =, label =, children =)`. `df_to_cascader_options()` builds them from a data.frame.
value	Initially selected path, as a vector of values from the top level down – or a list of paths with `props = list(multiple = TRUE)`.
placeholder	Placeholder text
props	Element's `props`, as a named list: `multiple`, `checkStrictly`, `expandTrigger` (`"click"` or `"hover"`), `lazy`, `lazyLoad`, and the field names `value`, `label`, `children`, `disabled`, `leaf`. With `lazy = TRUE` and no `lazyLoad` of your own, the server loads each column: see "Shiny inputs".
clearable	Whether clearable
filterable	Whether filterable (searchable)
disabled	Whether disabled
size	Size of cascader: `"large"`, `"default"` or `"small"`.
show_all_levels	Whether to show all levels in input
collapse_tags	Whether to collapse tags in multiple mode
separator	Separator for display
debounce	Debounce delay for filter
popper_class	Extra class name for the dropdown panel.
filter_method	`JS()` function filtering the options as the user types.
before_filter	`JS()` function called before filtering; returning `false` cancels it.
label	A label shown with the component, as Shiny's inputs have: text or a tag. `NULL`, the default, shows none. It is the component's accessible name too – tied to it with `for` where the component has a native input that takes the id `<id>-input`, else with `aria-labelledby`.
label_position	Where the label sits, as `el_form()`'s `label_position`: `"top"` (the default, as Shiny's labels sit), or beside the component, its text aligned `"left"` or `"right"` – which shows once `label_width` gives the labels a common width.

`label_width` Width of a label beside the component, as a CSS unit, so that several line up. Element's `label-width`.

`label_suffix` Text after the label, such as `":"`. Element's `label-suffix`.

`required` Draw Element's red asterisk before the label. It marks the field; it does not check it – shinyvalidate or `el_form()` does that.

`error` An error message shown under the component in Element's style, the field framed in red. Element's `error`.

`show_message, inline_message`
Whether `error`'s message is shown, and whether beside the component rather than under it. Element's `show-message` and `inline-message`.

`clear_icon` Custom clear icon component. Element Plus's `clear-icon` (string / Component). An icon's name, such as `"Search"`.

`collapse_tags_tooltip`
Whether show all selected tags when mouse hover text of collapse-tags. To use this, `collapse-tags` must be true. Element Plus's `collapse-tags-tooltip` (boolean).

`effect` Tooltip theme, built-in theme: `dark` / `light`. Element Plus's `effect` ('dark' | 'light' / string).

`empty_values` Empty values of component, see config-provider. Element Plus's `empty-values` (array).

`fallback_placements`
List of possible positions for Tooltip popper.js. Element Plus's `fallback-placements` (`Placement[]`).

`fit_input_width`
Whether the width of the suggestion panel is the same as the input, if the value is `number`, then the width is fixed. Element Plus's `fit-input-width` (boolean / number).

`height` Menu height for virtual scrolling (px). Element Plus's `height` (number).

`item_size` Node height for virtual scrolling (px). Element Plus's `item-size` (number).

`max_collapse_tags`
The max tags number to be shown. To use this, `collapse-tags` must be true. Element Plus's `max-collapse-tags` (number).

`max_collapse_tags_tooltip_height`
Max height of collapse-tags tooltip. Element Plus's `max-collapse-tags-tooltip-height` (string / number).

`persistent` When dropdown is inactive and `persistent` is `false`, dropdown will be destroyed. Element Plus's `persistent` (boolean).

`placement` Position of dropdown. Element Plus's `placement` (enum).

`popper_append_to_body`
Whether to append the popper menu to body. If the positioning of the popper is wrong, you can try to set this prop to false. Element Plus's `popper-append-to-body` (boolean).

`popper_style` Custom style for Cascader's dropdown and tags' tooltip. Element Plus's `popper-style` (string / object).

`show_checked_strategy`
: Strategy for displaying checked nodes in multiple selection mode. Use `parent` when you want things tidy. Use `child` when every single item matters. Element Plus's `show-checked-strategy` ('parent' | 'child').

`tag_effect`
: Tag effect. Element Plus's `tag-effect` ('light' | 'dark' | 'plain').

`tag_type`
: Tag type. Element Plus's `tag-type` ('success' | 'info' | 'warning' | 'danger').

`teleported`
: Whether cascader popup is teleported. Element Plus's `teleported` (boolean).

`validate_event`
: Whether to trigger form validation. Element Plus's `validate-event` (boolean).

`value_on_clear`
: Clear return value, see config-provider. Element Plus's `value-on-clear` (string / number / boolean / Function). Give it as `JS()`.

`virtual_scroll`
: Whether to enable virtual scrolling for large data. Element Plus's `virtual-scroll` (boolean).

`width`
: Component width, as a CSS unit – `"200px"`, `"50%"`, or a number taken as pixels. Element's own markup carries it, so it behaves like the `width` argument of a Shiny input.

`slots`
: Named list of Element slot contents, such as `list(title = shiny::tags$b("Bold"))`. A shiny.element component given here is absorbed rather than nested. For a scoped slot, write the template with `template()`.

`level_inputs`
: Each level of the path as an input of its own, beside `input$<id>`: input ids, one per level from the top, such as `c("compound", "study", "analysis")` for `input$compound`, `input$study` and `input$analysis`. `NULL`, the default, reports the path alone. See "Shiny inputs".

`events`
: Element's events to report besides those reported unasked, by name: `events = "node_drop"` reports `input$<id>_node_drop`. The component's are listed under "Shiny inputs", and by `el_events()`; a name it does not have is an error.

`on`
: Handlers of your own, for an event not reported or to send something else: a named list of `JS()` functions, one per event – Element's, or a DOM event with Vue's modifiers (`"keyup.enter"`). Each is called with `report` and the event's arguments; `report(name, value)` sets `input$<id>_<name>`. See `el_widget()`.

`session`
: In `el_cascader()`, deprecated: inside a module, wrap `id` in `ns()`, as for any Shiny input; a session given here namespaces `id` once more, with a warning. In `update_el_cascader()`, the Shiny session, the current one by default, as for `shiny::updateTextInput()`.

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
`input$<id>`	unasked	the selected path
`input$<id>_lazy_load`	unasked	with `props = list(lazy = TRUE)`, a column to load; answe
`input$<id>_expand_change`	`events = "expand_change"`	triggers when expand option changes

input$<id>_blur	events = "blur"	triggers when Cascader blurs
input$<id>_focus	events = "focus"	triggers when Cascader focuses
input$<id>_visible_change	events = "visible_change"	triggers when the dropdown appears/disappears
input$<id>_remove_tag	events = "remove_tag"	triggers when remove tag in multiple selection mode
input$<id>_clear	events = "clear"	triggers when the clear icon is clicked in a clearable Select

The same list as `el_events("el_cascader")`, which says how an event's arguments travel.

With `level_inputs`, each level of the path is an input too, named as given – not prefixed with the cascader's id, as the levels are the cascader's value in parts rather than something it does. Each holds that level's value, `NULL` where the path stops short (nothing selected, or `checkStrictly`), and follows the path wherever it changes: on load, when the user picks, when `update_el_cascader()` sets it. Like any input, it notifies its observers only when its value changes, so an observer of `input$compound` does not run again when only the study changes. The ids are the page's: in a module, give them through `ns()`, `level_inputs = ns(c("compound", "study"))`, as the cascader's own id. It needs the value to be one path, so not with `props = list(multiple = TRUE)` or `emitPath = FALSE`.

`input$<id>_lazy_load` carries `level` (0 for the first column), `value` and `path` of the option opened, and `request`. Answer with `el_load_children()`, passing the input back; each child is `list(value =, label =, leaf = TRUE)` for one with nothing below.

Element methods

Callable with `call_el()`:

- `getCheckedNodes()` – Get an array of currently selected node

Updating from the server

`update_el_cascader()` changes the component from the server.

Every other argument of `el_cascader()` that can change once it is drawn is an argument here too, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_cascader()` is called for its side effect and returns `NULL` invisibly.

Examples

```
# Basic cascader usage
cascader_options <- list(
  list(
    value = "guide",
    label = "Guide",
    children = list(
      list(value = "principle", label = "Principle"),
      list(value = "navigation", label = "Navigation")
    )
  ),
  list(
    value = "component",
    label = "Component",
```

```
    children = list(
      list(value = "basic", label = "Basic"),
      list(value = "form", label = "Form", disabled = TRUE)
    )
  )
)

if (interactive()) {
  library(shiny)
  library(shiny.element)
  ui <- el_page(
    el_cascader(
      id = "cascader1",
      options = cascader_options,
      placeholder = "Please select",
      clearable = TRUE
    ),
    verbatimTextOutput("selected")
  )
  server <- function(input, output, session) {
    output$selected <- renderPrint(input$cascader1)
  }
  shinyApp(ui, server)
}

# Advanced: custom props and update
custom_props <- list(
  expandTrigger = "hover",
  multiple = FALSE,
  checkStrictly = FALSE,
  emitPath = TRUE,
  lazy = FALSE,
  value = "value",
  label = "label",
  children = "children"
)

# Update cascader options in server:
# update_el_cascader(session, "cascader1", options = new_options)
if (interactive()) {
  # inside a server function
  observeEvent(input$go, {
    update_el_cascader(session, "region", value = list("zj", "hz"))
  })
}
```

---

el_cascader_panel *Element Plus Cascader Panel*

Description

The panel of an el_cascader() on its own, always open: nested options in side-by-side columns, for when there is room to show them rather than tuck them into a dropdown.

Usage

```
el_cascader_panel(
  id = NULL,
  options = list(),
  value = NULL,
  props = NULL,
  label = NULL,
  label_position = c("top", "left", "right"),
  label_width = NULL,
  label_suffix = NULL,
  required = FALSE,
  error = NULL,
  show_message = TRUE,
  inline_message = FALSE,
  height = NULL,
  item_size = NULL,
  virtual_scroll = NULL,
  width = NULL,
  slots = NULL,
  level_inputs = NULL,
  events = NULL,
  on = NULL,
  session = NULL
)

update_el_cascader_panel(
  session = shiny::getDefaultReactiveDomain(),
  id,
  value = NULL,
  options = NULL,
  label = NULL,
  error = NULL,
  props = NULL,
  height = NULL,
  item_size = NULL,
  virtual_scroll = NULL
)
```

Arguments

id	Panel ID. Auto-generated if NULL.
options	Nested options, each list(value =, label =, children =), as for el_cascader(). df_to_cascader_options() builds them from a data.frame.

`value` Initially selected path, as a vector of values from the top level down – or a list of paths with `props = list(multiple = TRUE)`.

`props` Element's `props`, as a named list: `multiple`, `checkStrictly`, `expandTrigger` (`"click"` or `"hover"`), `lazy`, `lazyLoad`, and the field names `value`, `label`, `children`, `disabled`, `leaf`. With `lazy = TRUE` and no `lazyLoad` of your own, the server loads each column.

`label` A label shown with the component, as Shiny's inputs have: text or a tag. `NULL`, the default, shows none. It is the component's accessible name too – tied to it with `for` where the component has a native input that takes the id `<id>-input`, else with `aria-labelledby`.

`label_position` Where the label sits, as `el_form()`'s `label_position`: `"top"` (the default, as Shiny's labels sit), or beside the component, its text aligned `"left"` or `"right"` – which shows once `label_width` gives the labels a common width.

`label_width` Width of a label beside the component, as a CSS unit, so that several line up. Element's `label-width`.

`label_suffix` Text after the label, such as `":"`. Element's `label-suffix`.

`required` Draw Element's red asterisk before the label. It marks the field; it does not check it – shinyvalidate or `el_form()` does that.

`error` An error message shown under the component in Element's style, the field framed in red. Element's `error`.

`show_message, inline_message`
Whether `error`'s message is shown, and whether beside the component rather than under it. Element's `show-message` and `inline-message`.

`height` Menu height for virtual scrolling (px). Element Plus's `height` (number).

`item_size` Node height for virtual scrolling (px). Element Plus's `item-size` (number).

`virtual_scroll` Whether to enable virtual scrolling for large data. Element Plus's `virtual-scroll` (boolean).

`width` Component width, as a CSS unit.

`slots` Named list of Element slot contents. The default slot, scoped with `{node, data}`, renders one option; write it with `template()`.

`level_inputs` Each level of the path as an input of its own: input ids, one per level from the top. See `el_cascader()`.

`events` Element's events to report besides those reported unasked, by name: `events = "node_drop"` reports `input$<id>_node_drop`. The component's are listed under "Shiny inputs", and by `el_events()`; a name it does not have is an error.

`on` Handlers of your own, for an event not reported or to send something else: a named list of `JS()` functions, one per event – Element's, or a DOM event with Vue's modifiers (`"keyup.enter"`). Each is called with `report` and the event's arguments; `report(name, value)` sets `input$<id>_<name>`. See `el_widget()`.

`session` In `el_cascader_panel()`, deprecated: inside a module, wrap `id` in `ns()`, as for any Shiny input; a session given here namespaces `id` once more, with a warning. In `update_el_cascader_panel()`, the Shiny session, the current one by default, as for `shiny::updateTextInput()`.

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
input$<id>	unasked	the selected path
input$<id>_lazy_load	unasked	with props = list(lazy = TRUE), a column to load; answer v
input$<id>_expand_change	events = "expand_change"	triggers when expand option changes
input$<id>_close	events = "close"	close panel event, provided to Cascader to put away the panel

The same list as `el_events("el_cascader_panel")`, which says how an event's arguments travel.

With `level_inputs`, each level of the path is an input too, as for `el_cascader()`.

Element methods

Callable with `call_el()`:

- `getCheckedNodes()` – the selected options
- `clearCheckedNodes()` – clear the selection

Updating from the server

Server-side update for `el_cascader_panel()`.

Every other argument of `el_cascader_panel()` that can change once it is drawn is an argument here too, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_cascader_panel()` is called for its side effect and returns `NULL` invisibly.

Examples

```
regions <- list(
  list(
    value = "asia",
    label = "Asia",
    children = list(
      list(value = "cn", label = "China"),
      list(value = "jp", label = "Japan")
    )
  ),
  list(
    value = "europe",
    label = "Europe",
    children = list(
      list(value = "fr", label = "France")
    )
  )
)
```

```
el_cascader_panel("where", options = regions, value = c("asia", "jp"))

# Several at once
el_cascader_panel("where", options = regions, props = list(multiple = TRUE))
if (interactive()) {
  # inside a server function
  observeEvent(
    input$reset,
    update_el_cascader_panel(session, "where", value = list())
  )
}
```

---

el_checkbox *Element Plus Checkbox*

Description

One box, ticked or not – Shiny's `shiny::checkboxInput()`, drawn by Element. For several choices, `el_checkbox_group()`.

Usage

```
el_checkbox(
  id = NULL,
  label = NULL,
  value = FALSE,
  indeterminate = NULL,
  disabled = NULL,
  border = NULL,
  size = NULL,
  true_label = NULL,
  false_label = NULL,
  name = NULL,
  checked = NULL,
  aria_controls = NULL,
  aria_label = NULL,
  controls = NULL,
  false_value = NULL,
  tabindex = NULL,
  true_value = NULL,
  validate_event = NULL,
  width = NULL,
  slots = NULL,
  on = NULL,
  session = NULL
)

update_el_checkbox(
```

```
  session = shiny::getDefaultReactiveDomain(),
  id,
  value = NULL,
  label = NULL,
  indeterminate = NULL,
  disabled = NULL,
  border = NULL,
  size = NULL,
  true_label = NULL,
  false_label = NULL,
  name = NULL,
  checked = NULL,
  aria_controls = NULL,
  aria_label = NULL,
  controls = NULL,
  false_value = NULL,
  tabindex = NULL,
  true_value = NULL,
  validate_event = NULL
)
```

Arguments

id	Checkbox ID. Auto-generated if NULL.
label	The box's text, as for shiny::checkboxInput().
value	Whether the box starts ticked. Default FALSE.
indeterminate	Show the box half-ticked – the "check all" box above a partly checked group. Only the look: value is unchanged.
disabled	Whether the box is disabled.
border	Draw the box with a border.
size	Size: "large", "default" or "small"; NULL follows the form or the page.
true_label, false_label	Values to report instead of TRUE and FALSE.
name	Native name attribute.
checked	Element's checked: tick the box when it is created, whatever value says. The same as value = TRUE, kept for code written from Element's documentation.
aria_controls	Same as aria-controls, takes effect when indeterminate is true. Element Plus's aria-controls (string).
aria_label	Native aria-label attribute. Element Plus's aria-label (string).
controls	Same as aria-controls, takes effect when indeterminate is true. Element Plus's controls (string).
false_value	Value of the Checkbox if it's not checked. Element Plus's false-value (string / number).
tabindex	Input tabindex. Element Plus's tabindex (string / number).

true_value	Value of the Checkbox if it's checked. Element Plus's true-value (string / number).
validate_event	Whether to trigger form validation. Element Plus's validate-event (boolean).
width	Component width, as a CSS unit.
slots	Named list of Element slot contents; the default slot replaces label.
on	Handlers of your own, for an event not reported or to send something else: a named list of JS() functions, one per event – Element's, or a DOM event with Vue's modifiers ("keyup.enter"). Each is called with report and the event's arguments; report(name, value) sets input$<id>_<name>. See el_widget().
session	In el_checkbox(), deprecated: inside a module, wrap id in ns(), as for any Shiny input; a session given here namespaces id once more, with a warning. In update_el_checkbox(), the Shiny session, the current one by default, as for shiny::updateTextInput().

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
input$<id>	unasked	TRUE or FALSE, or true_value and false_value

The same list as el_events("el_checkbox"), which says how an event's arguments travel.

Updating from the server

Server-side update for el_checkbox().

Every other argument of el_checkbox() that can change once it is drawn is an argument here too, under the same name. One left NULL stays as it is; NA returns it to Element's default.

update_el_checkbox() is called for its side effect and returns NULL invisibly.

Examples

```
el_checkbox("agree", "I agree to the terms")

# The "check all" box above a group
el_checkbox("all", "Check all", indeterminate = TRUE)

el_checkbox("remember", "Remember me", value = TRUE, border = TRUE)
if (interactive()) {
  # inside a server function: the "check all" box follows the group
  observeEvent(
    input$cities,
    {
      n <- length(input$cities)
```

```
      update_el_checkbox(
        session,
        "all",
        value = n == 4,
        indeterminate = n > 0 && n < 4
      )
    },
    ignoreNULL = FALSE
  )
}
```

---

el_checkbox_group *Element Plus Checkbox Group*

Description

Creates an Element Plus checkbox group with Vue instance, supporting individual checkboxes or button-style variants.

Usage

```
el_checkbox_group(
  id = NULL,
  choices = NULL,
  selected = NULL,
  disabled = FALSE,
  size = NULL,
  min = NULL,
  max = NULL,
  button = FALSE,
  fill = NULL,
  text_color = NULL,
  label = NULL,
  label_position = c("top", "left", "right"),
  label_width = NULL,
  label_suffix = NULL,
  required = FALSE,
  error = NULL,
  show_message = TRUE,
  inline_message = FALSE,
  width = NULL,
  slots = NULL,
  value = NULL,
  options = NULL,
  aria_label = NULL,
  props = NULL,
  tag = NULL,
  type = NULL,
```

```
  validate_event = NULL,
  on = NULL,
  session = NULL
)

update_el_checkbox_group(
  session = shiny::getDefaultReactiveDomain(),
  id,
  selected = NULL,
  choices = NULL,
  disabled = NULL,
  min = NULL,
  max = NULL,
  value = NULL,
  options = NULL,
  label = NULL,
  error = NULL,
  size = NULL,
  fill = NULL,
  text_color = NULL,
  aria_label = NULL,
  props = NULL,
  tag = NULL,
  type = NULL,
  validate_event = NULL
)
```

Arguments

id	Checkbox group ID. Auto-generated UUID if NULL.
choices, options	The choices: a named character vector (c(Label = value)) or a list of el_option()s – which can be disabled – or of list(value = ..., label = ...). choices is Shiny's name for it, options Element's; give either.
selected, value	Character vector of initially checked values; none by default. selected is Shiny's name, value Element's (its v-model); give either.
disabled	Whether the entire group is disabled. Default FALSE.
size	"large", "default" or "small".
min	Minimum number of checked items.
max	Maximum number of checked items.
button	Whether to use button-style checkboxes (el-checkbox-button). Default FALSE.
fill	Border and background colour when button = TRUE and checked.
text_color	Text colour when button = TRUE and checked.
label	A label shown with the component, as Shiny's inputs have: text or a tag. NULL, the default, shows none. It is the component's accessible name too – tied to it with for where the component has a native input that takes the id <id>-input, else with aria-labelledby.

`label_position` Where the label sits, as `el_form()`'s `label_position`: `"top"` (the default, as Shiny's labels sit), or beside the component, its text aligned `"left"` or `"right"` – which shows once `label_width` gives the labels a common width.

`label_width` Width of a label beside the component, as a CSS unit, so that several line up. Element's `label-width`.

`label_suffix` Text after the label, such as `":"`. Element's `label-suffix`.

`required` Draw Element's red asterisk before the label. It marks the field; it does not check it – shinyvalidate or `el_form()` does that.

`error` An error message shown under the component in Element's style, the field framed in red. Element's `error`.

`show_message, inline_message` Whether `error`'s message is shown, and whether beside the component rather than under it. Element's `show-message` and `inline-message`.

`width` Component width, as a CSS unit – `"200px"`, `"50%"`, or a number taken as pixels. Element's own markup carries it, so it behaves like the `width` argument of a Shiny input.

`slots` Named list of Element slot contents, such as `list(title = shiny::tags$b("Bold"))`. A shiny.element component given here is absorbed rather than nested. For a scoped slot, write the template with `template()`.

`aria_label` Native `aria-label` attribute. Element Plus's `aria-label` (string).

`props` Configuration options. Element Plus's `props` (`{ value?: string, label?: string, disabled?: stri`

`tag` Element tag of the checkbox group. Element Plus's `tag` (string).

`type` Component type to render options (e.g. `'button'`). Element Plus's `type` ('checkbox' | 'button').

`validate_event` Whether to trigger form validation. Element Plus's `validate-event` (boolean).

`on` Handlers of your own, for an event not reported or to send something else: a named list of `JS()` functions, one per event – Element's, or a DOM event with Vue's modifiers (`"keyup.enter"`). Each is called with `report` and the event's arguments; `report(name, value)` sets `input$<id>_<name>`. See `el_widget()`.

`session` In `el_checkbox_group()`, deprecated: inside a module, wrap `id` in `ns()`, as for any Shiny input; a session given here namespaces `id` once more, with a warning. In `update_el_checkbox_group()`, the Shiny session, the current one by default, as for `shiny::updateTextInput()`.

Value

An `htmltools` tagList with a Vue-managed checkbox group component.

Shiny inputs

Input	Reported	Value
`input$<id>`	unasked	the values checked

The same list as `el_events("el_checkbox_group")`, which says how an event's arguments travel.

Updating from the server

Server-side update for el_checkbox_group(). Pass only the fields to change; NULL fields are excluded from the update message.

Every other argument of el_checkbox_group() that can change once it is drawn is an argument here too, under the same name. One left NULL stays as it is; NA returns it to Element's default.

update_el_checkbox_group() is called for its side effect and returns NULL invisibly.

Examples

```
el_checkbox_group(
  "cb1",
  choices = c("Option A" = "a", "Option B" = "b")
)

if (interactive()) {
  library(shiny)
  library(shiny.element)
  ui <- el_page(
    el_checkbox_group(
      "cb1",
      choices = c("Apple" = "apple", "Banana" = "banana"),
      selected = "apple"
    ),
    verbatimTextOutput("selected")
  )
  server <- function(input, output, session) {
    output$selected <- renderPrint(input$cb1)
  }
  shinyApp(ui, server)
}
if (interactive()) {
  # inside a server function
  observeEvent(input$go, {
    update_el_checkbox_group(session, "langs", selected = c("r", "py"))
  })
}
```

---

el_check_tag *Element Plus Check Tag*

Description

A tag that toggles on and off when clicked, as a checkbox does.

Usage

```
el_check_tag(
  id,
```

```
  label,
  value = FALSE,
  disabled = NULL,
  type = NULL,
  width = NULL,
  on = NULL
)

update_el_check_tag(
  session = shiny::getDefaultReactiveDomain(),
  id,
  value = NULL,
  label = NULL,
  disabled = NULL,
  type = NULL
)
```

Arguments

id	Tag ID; the state is reported as input$<id>.
label	The tag's text.
value	Whether it starts checked: Element Plus's checked.
disabled	Whether it can be toggled.
type	"primary" (the default), "success", "info", "warning" or "danger".
width	Component width, as a CSS unit.
on	Handlers of your own, for an event not reported or to send something else: a named list of JS() functions, one per event – Element's, or a DOM event with Vue's modifiers ("keyup.enter"). Each is called with report and the event's arguments; report(name, value) sets input$<id>_<name>. See el_widget().
session	Shiny session; the current one by default, as for shiny::updateCheckboxInput().

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
input$<id>	unasked	TRUE while checked

The same list as el_events("el_check_tag"), which says how an event's arguments travel.

Updating from the server

`update_el_check_tag()` changes the component from the server.

Every other argument of `el_check_tag()` that can change once it is drawn is an argument here too, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_check_tag()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_check_tag("pinned", "Pinned", value = TRUE)
el_check_tag("urgent", "Urgent", type = "danger")
if (interactive()) {
  # inside a server function
  observeEvent(
    input$clear,
    update_el_check_tag(session, "pinned", value = FALSE)
  )
}
```

---

el_col *Element Plus Layout Column*

Description

Emits `<div class="el-col el-col-N">` directly; see `el_row()` for why.

Usage

```
el_col(
  ...,
  span = 24,
  offset = NULL,
  push = NULL,
  pull = NULL,
  xs = NULL,
  sm = NULL,
  md = NULL,
  lg = NULL,
  xl = NULL,
  tag = "div",
  class = NULL,
  style = NULL
)
```

Arguments

...	Column content.
span	Column span out of 24. Defaults to 24, as in Element Plus.
offset	Columns to offset by.
push	Columns to push right.
pull	Columns to pull left.
xs, sm, md, lg, xl	Responsive spans. Either a number (the span) or a list such as `list(span = 12, offset = 6)`.
tag	HTML element to render, as Element's `tag`. Default `"div"`; `"ul"` and `"li"` suit a grid of list items.
class	Extra CSS classes.
style	Extra inline style.

Value

A Shiny UI element.

Examples

```
el_col(span = 12, "half width")
el_col(span = 6, offset = 6, "quarter, pushed right")
el_col(xs = 24, sm = 12, md = 8, "responsive")
el_col(md = list(span = 12, offset = 6), "responsive with offset")
```

---

el_collapse *Element Plus Collapse / Accordion*

Description

Collapsible panels. Several can be open at once unless `accordion = TRUE`.

Usage

```
el_collapse(
  id = NULL,
  items = list(),
  value = character(0),
  accordion = FALSE,
  expand_icon_position = "right",
  before_collapse = NULL,
  session = NULL
)

update_el_collapse(
  session = shiny::getDefaultReactiveDomain(),
```

```
  id,
  value = NULL,
  accordion = NULL,
  expand_icon_position = NULL,
  before_collapse = NULL
)
```

Arguments

id	Collapse ID. Auto-generated UUID if NULL.
items	A list of panels, each an el_collapse_item() – or a named list with the same fields: **name** Unique panel identifier (string). Required. **title** Panel header text. Required. **content** Panel body. Any tag or tagList, including this package's own components. **disabled** Whether the header is disabled. Default FALSE. **icon** The expand icon, by name (default "ArrowRight"), or a tag.
value	Character vector of initially open panel names. In accordion mode only the first is used.
accordion	Single-open accordion mode. Default FALSE.
expand_icon_position	Where each header's icon sits: "right" (the default) or "left".
before_collapse	JS() function function(name), run before a panel opens or closes: return false, or a promise that resolves to false, to keep it as it is.
session	In el_collapse(), deprecated: inside a module, wrap id in ns(), as for any Shiny input; a session given here namespaces id once more, with a warning. In update_el_collapse(), the Shiny session, the current one by default, as for shiny::updateTextInput().

Details

Rendered as plain markup carrying Element's own classes, driven by a Shiny input binding rather than a Vue instance. That is what lets a panel hold other components from this package: a Vue instance mounted here would rebuild the DOM underneath them, detaching them from the server. See .claude/docs/lessons.md.

Value

An htmltools tag.

Shiny inputs

Input	Reported	Value
input$<id>	unasked	the names of the open panels

The same list as `el_events("el_collapse")`, which says how an event's arguments travel.

Updating from the server

Server-side update for `el_collapse()`: the open panels, `accordion`, `expand_icon_position` and `before_collapse` (`NA` for none).

`update_el_collapse()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_collapse(
  "col1",
  items = list(
    list(name = "p1", title = "Panel 1", content = shiny::tags$p("Content 1")),
    list(name = "p2", title = "Panel 2", content = shiny::tags$p("Content 2"))
  ),
  value = "p1"
)

# A panel can hold other components
el_collapse(
  "col2",
  items = list(
    list(
      name = "f",
      title = "Filters",
      content = shiny::tagList(el_input("q"), el_switch("live"))
    )
  )
)
if (interactive()) {
  # inside a server function
  observeEvent(input$go, {
    update_el_collapse(session, "panels", value = "filters")
  })
}
```

---

el_collapse_item *A panel of* `el_collapse()`

Description

Element Plus's `el-collapse-item`, for `el_collapse(items =)`.

Usage

```
el_collapse_item(title, ..., name = title, icon = NULL, disabled = NULL)
```

Arguments

title	Title of the panel.
...	The panel's content.
name	The panel's value, reported in input$<id> while it is open. Defaults to title.
icon	The expand icon, by name (Element's default "ArrowRight"), or a tag.
disabled	Whether the panel is disabled.

Value

A panel, for el_collapse(items =).

See Also

Other items: el_anchor_link(), el_breadcrumb_item(), el_carousel_item(), el_descriptions_item(), el_dropdown_item(), el_menu_item(), el_option(), el_skeleton_item(), el_step(), el_tab_pane(), el_table_column(), el_table_v2_column(), el_timeline_item(), el_tour_step()

Examples

```
el_collapse(
  "c",
  items = list(
    el_collapse_item("Consistency", "Consistent with real life."),
    el_collapse_item("Feedback", "Operation feedback.", disabled = TRUE)
  )
)
```

---

el_color_picker *Element Plus Color Picker*

Description

A colour picker input that returns a CSS colour string.

Usage

```
el_color_picker(
  id = NULL,
  value = NULL,
  disabled = FALSE,
  size = NULL,
  show_alpha = FALSE,
  color_format = NULL,
  predefine = NULL,
  popper_class = NULL,
  label = NULL,
```

```
  label_position = c("top", "left", "right"),
  label_width = NULL,
  label_suffix = NULL,
  required = FALSE,
  error = NULL,
  show_message = TRUE,
  inline_message = FALSE,
  append_to = NULL,
  aria_label = NULL,
  clearable = NULL,
  empty_values = NULL,
  persistent = NULL,
  popper_style = NULL,
  tabindex = NULL,
  teleported = NULL,
  validate_event = NULL,
  value_on_clear = NULL,
  width = NULL,
  slots = NULL,
  events = NULL,
  on = NULL,
  session = NULL
)

update_el_color_picker(
  session = shiny::getDefaultReactiveDomain(),
  id,
  value = NULL,
  disabled = NULL,
  label = NULL,
  error = NULL,
  size = NULL,
  show_alpha = NULL,
  color_format = NULL,
  predefine = NULL,
  popper_class = NULL,
  append_to = NULL,
  aria_label = NULL,
  clearable = NULL,
  empty_values = NULL,
  persistent = NULL,
  popper_style = NULL,
  tabindex = NULL,
  teleported = NULL,
  validate_event = NULL,
  value_on_clear = NULL
)
```

Arguments

id	Color picker ID. Auto-generated UUID if NULL.
value	Initial colour value (CSS hex/rgb string). NULL for empty.
disabled	Whether the picker is disabled. Default FALSE.
size	Size: "large", "default" or "small"; NULL follows the form or the page.
show_alpha	Whether to show an alpha channel slider. Default FALSE. When TRUE, the returned value is an rgba(...) string.
color_format	Output format: NULL (auto), "hex", "rgb", "hsv", "hsl".
predefine	Character vector of preset colour swatches. NULL for none.
popper_class	Extra class name for the dropdown panel.
label	A label shown with the component, as Shiny's inputs have: text or a tag. NULL, the default, shows none. It is the component's accessible name too – tied to it with for where the component has a native input that takes the id <id>-input, else with aria-labelledby.
label_position	Where the label sits, as el_form()'s label_position: "top" (the default, as Shiny's labels sit), or beside the component, its text aligned "left" or "right" – which shows once label_width gives the labels a common width.
label_width	Width of a label beside the component, as a CSS unit, so that several line up. Element's label-width.
label_suffix	Text after the label, such as ":". Element's label-suffix.
required	Draw Element's red asterisk before the label. It marks the field; it does not check it – shinyvalidate or el_form() does that.
error	An error message shown under the component in Element's style, the field framed in red. Element's error.
show_message, inline_message	Whether error's message is shown, and whether beside the component rather than under it. Element's show-message and inline-message.
append_to	Which element the color-picker panel appends to. Element Plus's append-to (CSSSelector / HTMLElement).
aria_label	ColorPicker aria-label. Element Plus's aria-label (string).
clearable	Whether to show clear button. Element Plus's clearable (boolean).
empty_values	Empty values of component, see config-provider. Element Plus's empty-values (array).
persistent	When color-picker inactive and persistent is false, the color panel will be destroyed. Element Plus's persistent (boolean).
popper_style	Custom style for ColorPicker's dropdown. Element Plus's popper-style (string / object).
tabindex	ColorPicker tabindex. Element Plus's tabindex (string / number).
teleported	Whether color-picker popper is teleported to the body. Element Plus's teleported (boolean).
validate_event	Whether to trigger form validation. Element Plus's validate-event (boolean).

`value_on_clear` Clear return value, see config-provider. Element Plus's `value-on-clear` (string / number / boolean / Function). Give it as `JS()`.

`width` Component width, as a CSS unit – `"200px"`, `"50%"`, or a number taken as pixels. Element's own markup carries it, so it behaves like the `width` argument of a Shiny input.

`slots` Named list of Element slot contents, such as `list(title = shiny::tags$b("Bold"))`. A shiny.element component given here is absorbed rather than nested. For a scoped slot, write the template with `template()`.

`events` Element's events to report besides those reported unasked, by name: `events = "node_drop"` reports `input$<id>_node_drop`. The component's are listed under "Shiny inputs", and by `el_events()`; a name it does not have is an error.

`on` Handlers of your own, for an event not reported or to send something else: a named list of `JS()` functions, one per event – Element's, or a DOM event with Vue's modifiers (`"keyup.enter"`). Each is called with `report` and the event's arguments; `report(name, value)` sets `input$<id>_<name>`. See `el_widget()`.

`session` In `el_color_picker()`, deprecated: inside a module, wrap `id` in `ns()`, as for any Shiny input; a session given here namespaces `id` once more, with a warning. In `update_el_color_picker()`, the Shiny session, the current one by default, as for `shiny::updateTextInput()`.

Value

An `htmltools` tagList with a Vue-managed color-picker component.

Shiny inputs

Input	Reported	Value
`input$<id>`	unasked	the colour
`input$<id>_active_change`	`events = "active_change"`	the colour being picked, at most every 200 ms
`input$<id>_blur`	`events = "blur"`	triggers when Component blurs
`input$<id>_clear`	`events = "clear"`	triggers when the clear button is clicked
`input$<id>_focus`	`events = "focus"`	triggers when Component focuses

The same list as `el_events("el_color_picker")`, which says how an event's arguments travel.

Updating from the server

Server-side update for `el_color_picker()`.

Every other argument of `el_color_picker()` that can change once it is drawn is an argument here too, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_color_picker()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_color_picker("cp1", value = "#409EFF")
el_color_picker("cp2", show_alpha = TRUE, predefine = c("#ff4500", "#ff8c00"))
if (interactive()) {
  # inside a server function
  observeEvent(input$go, {
    update_el_color_picker(session, "shade", value = "#67C23A")
  })
}
```

---

el_color_picker_panel *Element Plus Color Picker Panel*

Description

The colour picker's panel, always open, without the trigger.

Usage

```
el_color_picker_panel(
  id = NULL,
  value = NULL,
  border = NULL,
  disabled = NULL,
  show_alpha = NULL,
  color_format = NULL,
  predefine = NULL,
  validate_event = NULL,
  hue_slider_class = NULL,
  hue_slider_style = NULL,
  label = NULL,
  label_position = c("top", "left", "right"),
  label_width = NULL,
  label_suffix = NULL,
  required = FALSE,
  error = NULL,
  show_message = TRUE,
  inline_message = FALSE,
  width = NULL,
  slots = NULL,
  on = NULL
)

update_el_color_picker_panel(
  session = shiny::getDefaultReactiveDomain(),
  id,
  value = NULL,
```

```
  disabled = NULL,
  label = NULL,
  error = NULL,
  border = NULL,
  show_alpha = NULL,
  color_format = NULL,
  predefine = NULL,
  validate_event = NULL,
  hue_slider_class = NULL,
  hue_slider_style = NULL
)
```

Arguments

id	Component ID. Auto-generated if NULL.
value	Binding value: Element Plus's model-value, reported as input$<id>.
border	Whether the color picker panel is bordered. Element Plus's border (boolean).
disabled	Whether to disable the color picker. Element Plus's disabled (boolean).
show_alpha	Whether to display the alpha slider. Element Plus's show-alpha (boolean).
color_format	Color format of v-model. Element Plus's color-format (enum).
predefine	Predefined color options. Element Plus's predefine (string[]).
validate_event	Whether to trigger form validation. Element Plus's validate-event (boolean).
hue_slider_class	Class names will be passed to hue-slider. Element Plus's hue-slider-class (string | string[] | Record<string, boolean>).
hue_slider_style	Styles will be passed to hue-slider. Element Plus's hue-slider-style (string / StyleValue).
label	A label shown with the component, as Shiny's inputs have: text or a tag. NULL, the default, shows none. It is the component's accessible name too – tied to it with for where the component has a native input that takes the id <id>-input, else with aria-labelledby.
label_position	Where the label sits, as el_form()'s label_position: "top" (the default, as Shiny's labels sit), or beside the component, its text aligned "left" or "right" – which shows once label_width gives the labels a common width.
label_width	Width of a label beside the component, as a CSS unit, so that several line up. Element's label-width.
label_suffix	Text after the label, such as ":". Element's label-suffix.
required	Draw Element's red asterisk before the label. It marks the field; it does not check it – shinyvalidate or el_form() does that.
error	An error message shown under the component in Element's style, the field framed in red. Element's error.
show_message, inline_message	Whether error's message is shown, and whether beside the component rather than under it. Element's show-message and inline-message.

width	Component width, as a CSS unit.
slots	Named list of Element slot contents: footer. A scoped slot is written with template().
on	Handlers of your own, for an event not reported or to send something else: a named list of JS() functions, one per event – Element's, or a DOM event with Vue's modifiers ("keyup.enter"). Each is called with report and the event's arguments; report(name, value) sets input$<id>_<name>. See el_widget().
session	Shiny session; the current one by default, as for shiny::updateTextInput().

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
input$<id>	unasked	the colour

The same list as el_events("el_color_picker_panel"), which says how an event's arguments travel.

Element methods

Callable with call_el(): update().

Updating from the server

Server-side update for el_color_picker_panel().

Every other argument of el_color_picker_panel() that can change once it is drawn is an argument here too, under the same name. One left NULL stays as it is; NA returns it to Element's default.

update_el_color_picker_panel() is called for its side effect and returns NULL invisibly.

Examples

```
el_color_picker_panel("brand", value = "#409EFF")
if (interactive()) {
  # inside a server function
  observeEvent(
    input$reset,
    update_el_color_picker_panel(session, "x", value = NULL)
  )
}
```

el_config_provider *Element Plus Config Provider*

Description

Settings for the components inside it: their size, the button and link defaults, the card's, the dialog's and message's, the empty values, and their language. The page's own – its locale, size, z-index – are el_page()'s.

Usage

```
el_config_provider(
  ...,
  id = NULL,
  locale = NULL,
  size = NULL,
  button = NULL,
  link = NULL,
  dialog = NULL,
  message = NULL,
  experimental_features = NULL,
  empty_values = NULL,
  value_on_clear = NULL,
  table = NULL,
  card = NULL,
  width = NULL,
  slots = NULL
)

update_el_config_provider(
  session = shiny::getDefaultReactiveDomain(),
  id,
  locale = NULL,
  card = NULL,
  size = NULL,
  button = NULL,
  link = NULL,
  dialog = NULL,
  message = NULL,
  experimental_features = NULL,
  empty_values = NULL,
  value_on_clear = NULL,
  table = NULL
)
```

Arguments

`...`	Its content: any Shiny UI. Components of this package are folded into this one's Vue instance, as `el_button_group()` folds its buttons.
`id`	Component ID. Auto-generated if `NULL`.
`locale`	The language of the components inside it, as `el_locale_dependency()` names one (`"zh-cn"`), where `el_page(locale =)` sets the page's. Element Plus's `locale`.
`size`	Global component size. Element Plus's `size` ('large' | 'default' | 'small').
`button`	Button related configuration, see the following table. Element Plus's `button` (object).
`link`	Link related configuration, see the following table. Element Plus's `link` (`{type?: string, underline?:`
`dialog`	Dialog related configuration, see the following table. Element Plus's `dialog` (object).
`message`	Message related configuration, see the following table. Element Plus's `message` (`{max?: number}`).
`experimental_features`	Features at experimental stage to be added, all features are default to be set to false. Element Plus's `experimental-features` (object).
`empty_values`	Global empty values of components. Element Plus's `empty-values` (array).
`value_on_clear`	Global clear return value. Element Plus's `value-on-clear` (string / number / boolean / Function). Give it as `JS()`.
`table`	Table related configuration, see the following table. Element Plus's `table` (object). Give it as `JS()`.
`card`	Card related configuration: `list(shadow = "hover")`, for the cards inside it that set no shadow of their own. Element Plus's `card`.
`width`	Component width, as a CSS unit.
`slots`	Named list of Element slot contents. A scoped slot is written with `template()`.
`session`	Shiny session; the current one by default, as for `shiny::updateTextInput()`.

Value

A Shiny UI element.

Shiny inputs

None: it reports nothing.

Updating from the server

`update_el_config_provider()` changes the component from the server: every argument of `el_config_provider()` that can change once it is drawn, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_config_provider()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_config_provider(
  size = "small",
  button = list(autoInsertSpace = TRUE),
  el_button("a", "OK"),
  el_input("b")
)
```

`el_container` *Element Plus Container*

Description

Emits `<section class="el-container">` directly, as Element does.

Usage

```
el_container(
  ...,
  id = NULL,
  direction = NULL,
  style = NULL,
  class = NULL,
  session = NULL
)
```

Arguments

`...`	Child components, typically `el_header()`, `el_aside()`, `el_main()` and `el_footer()`.
`id`	Optional container id.
`direction`	`"horizontal"` or `"vertical"`. Defaults to vertical when a direct child is a header or footer, matching Element Plus.
`style`	Extra inline style.
`class`	Extra CSS classes.
`session`	Deprecated. Inside a module, wrap `id` in `ns()`, as for any Shiny input; a session given here namespaces `id` once more, with a warning.

Details

Element's container styles are plain CSS, so no Vue instance is needed – and none is wanted: one mounted over the container would recompile the components placed inside it and detach them.

Value

A Shiny UI element.

Examples

```
# Header above a sidebar and main area
el_container(
  el_header("Title"),
  el_container(
    el_aside(width = "200px", "Sidebar"),
    el_main("Content")
  )
)

# Nested inputs work, unlike with the previous Vue template approach
el_container(
  el_header(el_switch("dark_mode", value = FALSE)),
  el_main(el_slider("amount", value = 50))
)
```

---

el_date_picker *Element Plus Date Picker Component*

Description

Creates an Element Plus date picker with Vue instance, supporting single date, datetime, month, year, week, and date range selection modes.

Usage

```
el_date_picker(
  id = NULL,
  value = NULL,
  type = "date",
  value_format = NULL,
  format = NULL,
  placeholder = NULL,
  start_placeholder = NULL,
  end_placeholder = NULL,
  clearable = TRUE,
  disabled = FALSE,
  editable = TRUE,
  readonly = FALSE,
  range_separator = "-",
  size = NULL,
  name = NULL,
  prefix_icon = NULL,
  clear_icon = NULL,
  popper_class = NULL,
  default_value = NULL,
  default_time = NULL,
  unlink_panels = NULL,
```

```
  validate_event = NULL,
  label = NULL,
  label_position = c("top", "left", "right"),
  label_width = NULL,
  label_suffix = NULL,
  required = FALSE,
  error = NULL,
  show_message = TRUE,
  inline_message = FALSE,
  width = NULL,
  slots = NULL,
  arrow_control = NULL,
  automatic_dropdown = NULL,
  cell_class_name = NULL,
  date_format = NULL,
  disabled_date = NULL,
  disabled_hours = NULL,
  disabled_minutes = NULL,
  disabled_seconds = NULL,
  empty_values = NULL,
  fallback_placements = NULL,
  placement = NULL,
  popper_options = NULL,
  popper_style = NULL,
  shortcuts = NULL,
  show_confirm = NULL,
  show_footer = NULL,
  show_now = NULL,
  show_week_number = NULL,
  single_panel = NULL,
  teleported = NULL,
  time_format = NULL,
  value_on_clear = NULL,
  events = NULL,
  on = NULL,
  session = NULL
)

update_el_date_picker(
  session = shiny::getDefaultReactiveDomain(),
  id,
  value = NULL,
  disabled = NULL,
  type = NULL,
  clearable = NULL,
  readonly = NULL,
  placeholder = NULL,
  label = NULL,
```

```
  error = NULL,
  value_format = NULL,
  format = NULL,
  start_placeholder = NULL,
  end_placeholder = NULL,
  editable = NULL,
  range_separator = NULL,
  size = NULL,
  name = NULL,
  prefix_icon = NULL,
  clear_icon = NULL,
  popper_class = NULL,
  unlink_panels = NULL,
  validate_event = NULL,
  arrow_control = NULL,
  automatic_dropdown = NULL,
  cell_class_name = NULL,
  date_format = NULL,
  disabled_date = NULL,
  disabled_hours = NULL,
  disabled_minutes = NULL,
  disabled_seconds = NULL,
  empty_values = NULL,
  fallback_placements = NULL,
  placement = NULL,
  popper_options = NULL,
  popper_style = NULL,
  shortcuts = NULL,
  show_confirm = NULL,
  show_footer = NULL,
  show_now = NULL,
  show_week_number = NULL,
  single_panel = NULL,
  teleported = NULL,
  time_format = NULL,
  value_on_clear = NULL
)
```

Arguments

id	Date picker ID. Auto-generated UUID if NULL.
value	Initial value. A Date object, a string in the format matching value_format, or a two-element character vector for range types. NULL (default) leaves the picker empty.
type	Picker type: "date" (default), "datetime", "daterange", "datetimerange", "month", "year", "week".
value_format	Format string returned to Shiny when a date is selected, in day.js's tokens, as Element Plus takes it (e.g., "YYYY-MM-DD"). NULL (default) is "YYYY-MM-DD

HH:mm:ss" for "datetime" and "datetimerange" and "YYYY-MM-DD" for the rest. Element UI's tokens – "yyyy-MM-dd", "timestamp" – are translated.

format — Display format shown in the input box, in day.js's tokens. NULL (default) is Element's for the type: "YYYY-MM-DD HH:mm:ss" for a datetime, "YYYY-MM" for a month, "YYYY" for a year, and so on.

placeholder — Placeholder text for non-range types.

start_placeholder — Placeholder for the start input in range types.

end_placeholder — Placeholder for the end input in range types.

clearable — Whether to show the clear button. Default TRUE.

disabled — Whether the picker is disabled. Default FALSE.

editable — Whether the user can type directly in the input. Default TRUE.

readonly — Whether the picker is read-only. Default FALSE.

range_separator — Separator string displayed between start and end in range types. Default "-".

size — Size: "large", "default" or "small"; NULL follows the form or the page.

name — Native name attribute.

prefix_icon — Icon class shown at the start of the input.

clear_icon — Icon class of the clear button.

popper_class — Extra class name for the picker panel.

default_value — Date the panel opens on when nothing is selected.

default_time — Time part used when a date is picked, as "HH:mm:ss".

unlink_panels — Whether the two panels of a range picker move independently.

validate_event — Whether a change triggers form validation. Default TRUE.

label — A label shown with the component, as Shiny's inputs have: text or a tag. NULL, the default, shows none. It is the component's accessible name too – tied to it with for where the component has a native input that takes the id <id>-input, else with aria-labelledby.

label_position — Where the label sits, as el_form()'s label_position: "top" (the default, as Shiny's labels sit), or beside the component, its text aligned "left" or "right" – which shows once label_width gives the labels a common width.

label_width — Width of a label beside the component, as a CSS unit, so that several line up. Element's label-width.

label_suffix — Text after the label, such as ":". Element's label-suffix.

required — Draw Element's red asterisk before the label. It marks the field; it does not check it – shinyvalidate or el_form() does that.

error — An error message shown under the component in Element's style, the field framed in red. Element's error.

show_message, inline_message — Whether error's message is shown, and whether beside the component rather than under it. Element's show-message and inline-message.

`width` Component width, as a CSS unit – `"200px"`, `"50%"`, or a number taken as pixels. Element's own markup carries it, so it behaves like the `width` argument of a Shiny input.

`slots` Named list of Element slot contents, such as `list(title = shiny::tags$b("Bold"))`. A shiny.element component given here is absorbed rather than nested. For a scoped slot, write the template with `template()`.

`arrow_control` Whether to pick time using arrow buttons. Element Plus's `arrow-control` (boolean).

`automatic_dropdown` This prop decides if the date picker panel pops up when the input is focused. (The default value will be set to false in version 3.0). Element Plus's `automatic-dropdown` (boolean).

`cell_class_name` Set custom className. Element Plus's `cell-class-name` ((data: Date) => string).

`date_format` Optional, format of the date displayed in input's inner panel. Element Plus's `date-format` (string).

`disabled_date` A function determining if a date is disabled with that date as its parameter. Should return a Boolean. Element Plus's `disabled-date` ((data: Date) => boolean).

`disabled_hours` To specify the array of hours that cannot be selected. Element Plus's `disabled-hours` ((role: string, comparingDate?: Dayjs) => number[]).

`disabled_minutes` To specify the array of minutes that cannot be selected. Element Plus's `disabled-minutes` ((hour: number, role: string, comparingDate?: Dayjs) => number[]).

`disabled_seconds` To specify the array of seconds that cannot be selected. Element Plus's `disabled-seconds` (Function). Give it as `JS()`.

`empty_values` Empty values of component, see config-provider. Element Plus's `empty-values` (array).

`fallback_placements` List of possible positions for Tooltip popper.js. Element Plus's `fallback-placements` (`Placement[]`).

`placement` Position of dropdown. Element Plus's `placement`.

`popper_options` Customized popper option see more at popper.js. Element Plus's `popper-options` (`Partial<PopperOptions>`).

`popper_style` Custom style for DatePicker's dropdown. Element Plus's `popper-style` (string / object).

`shortcuts` An object array to set shortcut options. Element Plus's `shortcuts` (`Array<{ text: string, value: Dat` Give it as `JS()`.

`show_confirm` Whether to show the confirm button. Element Plus's `show-confirm` (boolean).

`show_footer` Whether to show footer where the date picker is one `'dates' | 'months' | 'years' | 'quarters'`. Element Plus's `show-footer` (boolean).

`show_now` Whether to show the now button. Element Plus's `show-now` (boolean).

show_week_number
: Show the week number besides the week. Element Plus's `show-week-number` (boolean).

single_panel
: Show only one panel in range-picker. Element Plus's `single-panel` (boolean).

teleported
: Whether date-picker dropdown is teleported to the body. Element Plus's `teleported` (boolean).

time_format
: Optional, format of the time displayed in input's inner panel. Element Plus's `time-format` (string).

value_on_clear
: Clear return value, see config-provider. Element Plus's `value-on-clear` (string / number / boolean / Function). Give it as `JS()`.

events
: Element's events to report besides those reported unasked, by name: `events = "node_drop"` reports `input$<id>_node_drop`. The component's are listed under "Shiny inputs", and by `el_events()`; a name it does not have is an error.

on
: Handlers of your own, for an event not reported or to send something else: a named list of `JS()` functions, one per event – Element's, or a DOM event with Vue's modifiers (`"keyup.enter"`). Each is called with `report` and the event's arguments; `report(name, value)` sets `input$<id>_<name>`. See `el_widget()`.

session
: In `el_date_picker()`, deprecated: inside a module, wrap `id` in `ns()`, as for any Shiny input; a session given here namespaces `id` once more, with a warning. In `update_el_date_picker()`, the Shiny session, the current one by default, as for `shiny::updateTextInput()`.

Value

An `htmltools` tagList with a Vue-managed date picker component.

Element methods

Callable with `call_el()`:

- `focus()` – Focus the Input component

Shiny inputs

Input	Reported	Value
`input$<id>`	unasked	the date, two for a range
`input$<id>_blur`	`events = "blur"`	triggers when Input blurs
`input$<id>_focus`	`events = "focus"`	triggers when Input focuses
`input$<id>_calendar_change`	`events = "calendar_change"`	triggers when the calendar selected date is changed. Onl
`input$<id>_clear`	`events = "clear"`	triggers when a clear button is clicked
`input$<id>_panel_change`	`events = "panel_change"`	triggers when the navigation button click.
`input$<id>_visible_change`	`events = "visible_change"`	triggers when the DatePicker's dropdown appears/disapp

The same list as `el_events("el_date_picker")`, which says how an event's arguments travel.

For `type` `"date"`, `"dates"` and `"daterange"` with the default `value_format`, `input$<id>` is a `Date` (two for a range, several for `"dates"`), as `shiny::dateInput()` gives one; `NULL` while empty.

Any other type or `value_format` reports the text the picker produces, in that format – you asked for that format, so it is not converted.

Updating from the server

Server-side update for `el_date_picker()`. Supports updating value, disabled state, type, clearable, readonly, and placeholder text.

Every other argument of `el_date_picker()` that can change once it is drawn is an argument here too, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_date_picker()` is called for its side effect and returns `NULL` invisibly.

Examples

```
# Basic date picker
el_date_picker("dp1")

# Pre-filled with today's date
el_date_picker("dp2", value = Sys.Date())

# Date range picker
el_date_picker(
  "dp3",
  type = "daterange",
  start_placeholder = "Start date",
  end_placeholder = "End date"
)

# Shiny app example
if (interactive()) {
  library(shiny)
  library(shiny.element)
  ui <- el_page(
    el_date_picker("dp1", placeholder = "Pick a date"),
    verbatimTextOutput("selected")
  )
  server <- function(input, output, session) {
    output$selected <- renderPrint(input$dp1)
  }
  shinyApp(ui, server)
}
if (interactive()) {
  # inside a server function
  observeEvent(input$go, {
    update_el_date_picker(session, "when", value = "2026-06-01")
  })
}
```

el_date_picker_panel *Element Plus Date Picker Panel*

Description

The date picker's panel, always open, without the input.

Usage

```
el_date_picker_panel(
  id = NULL,
  value = NULL,
  border = NULL,
  disabled = NULL,
  clearable = NULL,
  editable = NULL,
  type = NULL,
  default_value = NULL,
  default_time = NULL,
  value_format = NULL,
  date_format = NULL,
  time_format = NULL,
  unlink_panels = NULL,
  single_panel = NULL,
  disabled_date = NULL,
  shortcuts = NULL,
  cell_class_name = NULL,
  show_footer = NULL,
  show_confirm = NULL,
  show_week_number = NULL,
  label = NULL,
  label_position = c("top", "left", "right"),
  label_width = NULL,
  label_suffix = NULL,
  required = FALSE,
  error = NULL,
  show_message = TRUE,
  inline_message = FALSE,
  width = NULL,
  slots = NULL,
  events = NULL,
  on = NULL
)

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

```
  value = NULL,
  disabled = NULL,
  label = NULL,
  error = NULL,
  border = NULL,
  clearable = NULL,
  editable = NULL,
  type = NULL,
  value_format = NULL,
  date_format = NULL,
  time_format = NULL,
  unlink_panels = NULL,
  single_panel = NULL,
  disabled_date = NULL,
  shortcuts = NULL,
  cell_class_name = NULL,
  show_footer = NULL,
  show_confirm = NULL,
  show_week_number = NULL
)
```

Arguments

id	Component ID. Auto-generated if NULL.
value	Binding value, if it is an range picker, the length of the array should be 2: Element Plus's model-value, reported as input$<id>.
border	Whether the date picker is bordered. Element Plus's border (boolean).
disabled	Whether DatePicker is disabled. Element Plus's disabled (boolean).
clearable	Whether to show clear button. Element Plus's clearable (boolean).
editable	Whether the input is editable. Element Plus's editable (boolean).
type	Type of the picker. quarter, quarters, and quarterrange are supported. Element Plus's type (enum).
default_value	Optional, default date of the calendar. Element Plus's default-value (Date | [Date, Date]).
default_time	Optional, the time value to use when selecting date range. Element Plus's default-time (Date | [Date, Date]).
value_format	Optional, format of binding value. If not specified, the binding value will be a Date object. Element Plus's value-format (string).
date_format	Optional, format of the date displayed in input's inner panel. Element Plus's date-format (string).
time_format	Optional, format of the time displayed in input's inner panel. Element Plus's time-format (string).
unlink_panels	Unlink two date-panels in range-picker. Element Plus's unlink-panels (boolean).
single_panel	Show only one panel in range-picker. Element Plus's single-panel (boolean).

`disabled_date` A function determining if a date is disabled with that date as its parameter. Should return a Boolean. Element Plus's `disabled-date` ((data: Date) => boolean). Give it as `JS()`.

`shortcuts` An object array to set shortcut options. Element Plus's `shortcuts` (`Array<{ text: string, value: Dat` Give it as `JS()`.

`cell_class_name`
Set custom className. Element Plus's `cell-class-name` ((data: Date) => string). Give it as `JS()`.

`show_footer` Whether to show footer where the date picker is one `'dates' | 'months' | 'years' | 'quarters' | 'datetime' | 'datetimerange'`. Element Plus's `show-footer` (boolean).

`show_confirm` Whether to show the confirm button. Element Plus's `show-confirm` (boolean).

`show_week_number`
Show the week number besides the week. Element Plus's `show-week-number` (boolean).

`label` A label shown with the component, as Shiny's inputs have: text or a tag. `NULL`, the default, shows none. It is the component's accessible name too – tied to it with `for` where the component has a native input that takes the id `<id>-input`, else with `aria-labelledby`.

`label_position` Where the label sits, as `el_form()`'s `label_position`: `"top"` (the default, as Shiny's labels sit), or beside the component, its text aligned `"left"` or `"right"` – which shows once `label_width` gives the labels a common width.

`label_width` Width of a label beside the component, as a CSS unit, so that several line up. Element's `label-width`.

`label_suffix` Text after the label, such as `":"`. Element's `label-suffix`.

`required` Draw Element's red asterisk before the label. It marks the field; it does not check it – shinyvalidate or `el_form()` does that.

`error` An error message shown under the component in Element's style, the field framed in red. Element's `error`.

`show_message, inline_message`
Whether `error`'s message is shown, and whether beside the component rather than under it. Element's `show-message` and `inline-message`.

`width` Component width, as a CSS unit.

`slots` Named list of Element slot contents: `prev-month`, `next-month`, `prev-year`, `next-year`. A scoped slot is written with `template()`.

`events` Element's events to report besides those reported unasked, by name: `events = "node_drop"` reports `input$<id>_node_drop`. The component's are listed under "Shiny inputs", and by `el_events()`; a name it does not have is an error.

`on` Handlers of your own, for an event not reported or to send something else: a named list of `JS()` functions, one per event – Element's, or a DOM event with Vue's modifiers (`"keyup.enter"`). Each is called with `report` and the event's arguments; `report(name, value)` sets `input$<id>_<name>`. See `el_widget()`.

`session` Shiny session; the current one by default, as for `shiny::updateTextInput()`.

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
`input$<id>`	unasked	the date, two for a range
`input$<id>_calendar_change`	`events = "calendar_change"`	triggers when the calendar selected date is changed. Onl
`input$<id>_panel_change`	`events = "panel_change"`	triggers when the navigation button click.
`input$<id>_clear`	`events = "clear"`	triggers when a clear button is clicked

The same list as `el_events("el_date_picker_panel")`, which says how an event's arguments travel.

Updating from the server

Server-side update for `el_date_picker_panel()`.

Every other argument of `el_date_picker_panel()` that can change once it is drawn is an argument here too, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_date_picker_panel()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_date_picker_panel("day", value = Sys.Date(), value_format = "YYYY-MM-DD")
if (interactive()) {
  # inside a server function
  observeEvent(
    input$reset,
    update_el_date_picker_panel(session, "x", value = NULL)
  )
}
```

---

el_descriptions *Element Plus Descriptions*

Description

A read-only grid of labelled values – the detail view of a record.

Usage

```
el_descriptions(
  id = NULL,
  items = list(),
  title = NULL,
  extra = NULL,
  column = NULL,
  direction = NULL,
  border = NULL,
  size = NULL,
  label_width = NULL,
  width = NULL,
  slots = NULL,
  on = NULL,
  session = NULL
)

update_el_descriptions(
  session = shiny::getDefaultReactiveDomain(),
  id,
  title = NULL,
  extra = NULL,
  column = NULL,
  direction = NULL,
  border = NULL,
  size = NULL,
  label_width = NULL
)
```

Arguments

id	Component ID. Auto-generated if NULL.
items	The fields, each an el_descriptions_item() – or a list(label =, content =) – or a named list or vector whose names are the labels. content may be any Shiny UI, a shiny.element component included, which is absorbed rather than nested. An item may also carry Element Plus's item props: span, rowspan, width, min_width, label_width, align, label_align, class_name and label_class_name. A label that is markup rather than text fills the item's label slot.
title	Heading above the grid.
extra	Text at the heading's right end. For something interactive, use slots = list(extra = ...).
column	Items per row. Default 3.
direction	"horizontal" (default) puts each label beside its value; "vertical" puts it above.
border	Whether to draw cell borders.
size	Size: "large", "default" or "small"; NULL follows the form or the page.

label_width — Label width of every column. Element Plus's label-width (string / number).

width — Component width, as a CSS unit.

slots — Named list of Element slot contents: title, extra.

on — Handlers of your own, for an event not reported or to send something else: a named list of JS() functions, one per event – Element's, or a DOM event with Vue's modifiers ("keyup.enter"). Each is called with report and the event's arguments; report(name, value) sets input$<id>_<name>. See el_widget().

session — In el_descriptions(), deprecated: inside a module, wrap id in ns(), as for any Shiny input; a session given here namespaces id once more, with a warning. In update_el_descriptions(), the Shiny session, the current one by default, as for shiny::updateTextInput().

Value

A Shiny UI element.

Updating from the server

Server-side update for el_descriptions(). The items themselves are markup, so replace them by re-rendering; this changes the settings around them.

update_el_descriptions() is called for its side effect and returns NULL invisibly.

Examples

```
el_descriptions(
  "user",
  title = "Account",
  border = TRUE,
  items = list(
    list(label = "Name", content = "Ada Lovelace"),
    list(label = "Plan", content = el_tag("plan", "Pro", type = "success")),
    list(label = "Address", content = "12 St James's Square, London", span = 2)
  )
)

# The quick form: names are labels
el_descriptions("car", items = as.list(mtcars[1, 1:6]))
if (interactive()) {
  # inside a server function
  observeEvent(input$narrow, {
    update_el_descriptions(session, "user", column = 1)
  })
}
```

`el_descriptions_item` *A field of* `el_descriptions()`

Description

Element Plus's `el-descriptions-item`, for `el_descriptions(items =)`.

Usage

```
el_descriptions_item(
  label,
  ...,
  span = NULL,
  rowspan = NULL,
  width = NULL,
  min_width = NULL,
  label_width = NULL,
  align = NULL,
  label_align = NULL,
  class_name = NULL,
  label_class_name = NULL
)
```

Arguments

`label`	Label text, or markup for the item's label slot.
`...`	The field's content: text, tags, this package's components.
`span`	Number of columns the field spans.
`rowspan`	Number of rows the field spans.
`width`	Column width; the widest in a column wins. Without a border it includes the label.
`min_width`	Minimum column width; columns with `width` keep it, the others share the rest in proportion.
`label_width`	Width of the label, over the descriptions' own.
`align`	Content alignment: `"left"`, `"center"` or `"right"`.
`label_align`	Label alignment, `align`'s when not given.
`class_name`	Class of the content cell.
`label_class_name`	Class of the label cell.

Value

A field, for `el_descriptions(items =)`.

See Also

Other items: el_anchor_link(), el_breadcrumb_item(), el_carousel_item(), el_collapse_item(), el_dropdown_item(), el_menu_item(), el_option(), el_skeleton_item(), el_step(), el_tab_pane(), el_table_column(), el_table_v2_column(), el_timeline_item(), el_tour_step()

Examples

```
el_descriptions(
  "d",
  items = list(
    el_descriptions_item("Username", "kooriookami"),
    el_descriptions_item("Address", "No.1188, Wuzhong Avenue", span = 2)
  )
)
```

---

el_dialog *Element Plus Dialog*

Description

A modal dialog.

Usage

```
el_dialog(
  id = NULL,
  title = "",
  content = NULL,
  footer = NULL,
  visible = FALSE,
  width = "50%",
  top = "15vh",
  fullscreen = FALSE,
  modal = TRUE,
  close_on_click_modal = TRUE,
  close_on_press_escape = TRUE,
  show_close = TRUE,
  center = FALSE,
  lock_scroll = TRUE,
  custom_class = NULL,
  append_to_body = FALSE,
  destroy_on_close = FALSE,
  before_close = NULL,
  modal_penetrable = FALSE,
  close_icon = NULL,
  align_center = NULL,
  draggable = NULL,
```

```
  overflow = NULL,
  modal_class = NULL,
  header_class = NULL,
  body_class = NULL,
  footer_class = NULL,
  append_to = NULL,
  open_delay = NULL,
  close_delay = NULL,
  z_index = NULL,
  header_aria_level = "2",
  transition = NULL,
  events = NULL,
  session = NULL
)

update_el_dialog(
  session = shiny::getDefaultReactiveDomain(),
  id,
  visible = NULL,
  title = NULL,
  width = NULL,
  top = NULL,
  fullscreen = NULL,
  modal = NULL,
  close_on_click_modal = NULL,
  close_on_press_escape = NULL,
  show_close = NULL,
  center = NULL,
  lock_scroll = NULL,
  draggable = NULL,
  overflow = NULL,
  align_center = NULL
)
```

Arguments

id	Dialog ID. Auto-generated UUID if NULL.
title	Header text. "" renders the header bar without a title.
content	Dialog body. Any tag or tagList, including this package's own components.
footer	Footer content, usually buttons. NULL for none.
visible	Whether it starts open: Element Plus's model-value.
width	Dialog width, e.g. "50%" or "600px".
top	Distance from the top of the viewport. Ignored when fullscreen = TRUE or align_center = TRUE.
fullscreen	Fill the viewport.
modal	Show the backdrop.

`close_on_click_modal`
: Close when the backdrop is clicked.

`close_on_press_escape`
: Close on Escape.

`show_close`
: Show the close button in the header.

`center`
: Centre the header and footer.

`lock_scroll`
: Whether the page stops scrolling while it is open.

`custom_class`
: Extra class name for the panel.

`append_to_body`
: `append_to = "body"`.

`destroy_on_close`
: Re-create the content each time it opens, and remove it when it closes: inputs inside start from their initial values again.

`before_close`
: `htmltools::JS()` function `function(done)`, run when the user closes it – by the cross, the backdrop or Escape; call `done()` to let it close.

`modal_penetrable`
: Let clicks through the backdrop to the page beneath, when `modal = FALSE`.

`close_icon`
: The close button's icon, by name. Default `"Close"`.

`align_center`
: Centre the dialog in the viewport, vertically too.

`draggable`
: Let the dialog be dragged by its header.

`overflow`
: With `draggable`, let it be dragged past the viewport. `align_center`, `draggable` and `overflow` left `NULL` are off, or a config provider's el_config_provider(dialog =).

`modal_class`, `header_class`, `body_class`, `footer_class`
: Extra class names for the backdrop, the header, the body and the footer.

`append_to`
: A CSS selector for where the overlay goes when it opens, so a container's `overflow` or `transform` cannot clip it. Its components keep working; they are moved, not re-created.

`open_delay`, `close_delay`
: Milliseconds to wait before opening and closing.

`z_index`
: The overlay's z-index, instead of the next one Element Plus hands out.

`header_aria_level`
: The title's `aria-level`. Default `"2"`.

`transition`
: The name of the transition it plays, instead of Element's `"dialog-fade"`.

`events`
: Element's events to report besides those reported unasked, by name: `events = "node_drop"` reports `input$<id>_node_drop`. The component's are listed under "Shiny inputs", and by `el_events()`; a name it does not have is an error.

`session`
: In `el_dialog()`, deprecated: inside a module, wrap `id` in `ns()`, as for any Shiny input; a session given here namespaces `id` once more, with a warning. In `update_el_dialog()`, the Shiny session, the current one by default, as for `shiny::updateTextInput()`.

Details

Rendered as plain markup carrying Element Plus's own classes, driven by a Shiny input binding rather than a Vue instance, so the body can hold other components from this package.

Value

An `htmltools` tag.

Shiny inputs

Input	Reported	Value
`input$<id>`	unasked	`TRUE` while it is open
`input$<id>_open`	`events = "open"`	triggers when the Dialog opens
`input$<id>_opened`	`events = "opened"`	triggers when the Dialog opening animation ends
`input$<id>_close`	`events = "close"`	triggers when the Dialog closes
`input$<id>_closed`	unasked	triggers when the Dialog closing animation ends
`input$<id>_open_auto_focus`	`events = "open_auto_focus"`	triggers after Dialog opens and content focused
`input$<id>_close_auto_focus`	`events = "close_auto_focus"`	triggers after Dialog closed and content focused

The same list as `el_events("el_dialog")`, which says how an event's arguments travel.

Shiny routes an input binding's messages by element id, so the value is `input$<id>` itself, as for `el_tabs()` and `el_collapse()`.

Element methods

Callable with `call_el()`: `resetPosition()` puts a dragged dialog back.

Updating from the server

Server-side update for `el_dialog()`: whether it shows, its title and width, and how it looks and behaves – `top`, `fullscreen`, `modal`, `close_on_click_modal`, `close_on_press_escape`, `show_close`, `center`, `lock_scroll`, `draggable`, `overflow`, `align_center` – as Element's props, reactive there.

`update_el_dialog()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_dialog(
  "d1",
  title = "Confirm",
  content = shiny::tags$p("Are you sure?"),
  footer = el_button("ok", "OK", type = "primary")
)

# The body can hold other components
el_dialog(
  "d2",
  title = "Filters",
  draggable = TRUE,
  content = shiny::tagList(el_input("q"), el_switch("live"))
)
if (interactive()) {
  # inside a server function
  observeEvent(input$go, {
```

```
    update_el_dialog(session, "confirm", visible = TRUE)
  })
}
```

`el_divider` *Element Plus Divider*

Description

Renders a horizontal or vertical dividing line, optionally with inline text.

Usage

```
el_divider(
  content = NULL,
  direction = "horizontal",
  content_position = "center",
  border_style = "solid",
  class = NULL,
  style = NULL
)
```

Arguments

`content`	Optional text/tag placed inside the divider. Only for `direction = "horizontal"`.
`direction`	Divider orientation: `"horizontal"` (default) or `"vertical"`.
`content_position`	Position of inline text when `content` is supplied: `"center"` (default), `"left"`, or `"right"`.
`border_style`	The line's style, as CSS `border-style`: `"solid"` (the default), `"dashed"`, `"dotted"`.
`class, style`	Extra classes and inline style, as Element passes them to its root – `style = "height: auto"` for a vertical divider that stretches in a flex row.

Value

An `htmltools` tag.

Examples

```
el_divider()
el_divider("Title Text", content_position = "left")
el_divider(direction = "vertical")
```

el_drawer *Element Plus Drawer*

Description

A panel that slides in from an edge of the viewport.

Usage

```
el_drawer(
  id = NULL,
  title = "",
  content = NULL,
  visible = FALSE,
  direction = "rtl",
  size = "30%",
  modal = TRUE,
  with_header = TRUE,
  show_close = TRUE,
  close_on_press_escape = TRUE,
  custom_class = NULL,
  append_to_body = FALSE,
  destroy_on_close = FALSE,
  before_close = NULL,
  footer = NULL,
  resizable = FALSE,
  modal_penetrable = FALSE,
  close_on_click_modal = TRUE,
  lock_scroll = TRUE,
  modal_class = NULL,
  header_class = NULL,
  body_class = NULL,
  footer_class = NULL,
  append_to = NULL,
  open_delay = NULL,
  close_delay = NULL,
  z_index = NULL,
  header_aria_level = "2",
  events = NULL,
  session = NULL
)

update_el_drawer(
  session = shiny::getDefaultReactiveDomain(),
  id,
  visible = NULL,
  title = NULL,
```

```
  size = NULL,
  direction = NULL,
  modal = NULL,
  with_header = NULL,
  show_close = NULL,
  close_on_click_modal = NULL,
  close_on_press_escape = NULL,
  lock_scroll = NULL
)
```

Arguments

id	Drawer ID. Auto-generated UUID if NULL.
title	Header text.
content	Drawer body. Any tag or tagList, including this package's own components.
visible	Whether it starts open: Element Plus's model-value.
direction	Edge it slides from: "rtl" (from the right, the default), "ltr", "ttb" or "btt".
size	Width for a horizontal drawer, height for a vertical one.
modal	Show the backdrop.
with_header	Show the header bar.
show_close	Show the close button in the header.
close_on_press_escape	Close on Escape.
custom_class	Extra class name for the panel.
append_to_body	append_to = "body".
destroy_on_close	Re-create the content each time it opens, and remove it when it closes: inputs inside start from their initial values again.
before_close	htmltools::JS() function function(done), run when the user closes it – by the cross, the backdrop or Escape; call done() to let it close.
footer	Footer content, usually buttons. NULL for none.
resizable	Let the drawer be resized by dragging its inner edge.
modal_penetrable	Let clicks through the backdrop to the page beneath, when modal = FALSE.
close_on_click_modal	Close when the backdrop is clicked. (Element UI's wrapper-closable.)
lock_scroll	Whether the page stops scrolling while it is open.
modal_class, header_class, body_class, footer_class	Extra class names for the backdrop, the header, the body and the footer.
append_to	A CSS selector for where the overlay goes when it opens, so a container's overflow or transform cannot clip it. Its components keep working; they are moved, not re-created.

`open_delay`, `close_delay`
: Milliseconds to wait before opening and closing.

`z_index`
: The overlay's z-index, instead of the next one Element Plus hands out.

`header_aria_level`
: The title's `aria-level`. Default `"2"`.

`events`
: Element's events to report besides those reported unasked, by name: `events = "node_drop"` reports `input$<id>_node_drop`. The component's are listed under "Shiny inputs", and by `el_events()`; a name it does not have is an error.

`session`
: In `el_drawer()`, deprecated: inside a module, wrap `id` in `ns()`, as for any Shiny input; a session given here namespaces `id` once more, with a warning. In `update_el_drawer()`, the Shiny session, the current one by default, as for `shiny::updateTextInput()`.

Details

Rendered as plain markup carrying Element Plus's own classes, driven by a Shiny input binding rather than a Vue instance, so the body can hold other components from this package.

Value

An `htmltools` tag.

Shiny inputs

Input	Reported	Value
`input$<id>`	unasked	`TRUE` while it is open
`input$<id>_open`	`events = "open"`	Triggered before Drawer opening animation begins
`input$<id>_opened`	`events = "opened"`	Triggered after Drawer opening animation ended
`input$<id>_close`	`events = "close"`	Triggered before Drawer closing animation begins
`input$<id>_closed`	unasked	Triggered after Drawer closing animation ended
`input$<id>_open_auto_focus`	`events = "open_auto_focus"`	triggers after Drawer opens and content focused
`input$<id>_close_auto_focus`	`events = "close_auto_focus"`	triggers after Drawer closed and content focused
`input$<id>_resize_start`	`events = "resize_start"`	the size, in pixels
`input$<id>_resize`	`events = "resize"`	the size, in pixels
`input$<id>_resize_end`	`events = "resize_end"`	the size, in pixels

The same list as `el_events("el_drawer")`, which says how an event's arguments travel.

Element methods

Callable with `call_el()`: `handleClose()` closes it the way the user would, through `before_close` (`closeDrawer()`, Element UI's name, too).

Updating from the server

Server-side update for `el_drawer()`: whether it shows, its title, its size, the edge it slides from (`direction`), and how it behaves – `modal`, `with_header`, `show_close`, `close_on_click_modal`, `close_on_press_escape`, `lock_scroll` – as Element's props, reactive there.

`update_el_drawer()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_drawer("w1", title = "Settings", content = shiny::tags$p("Body"))

# Sliding up from the bottom, holding other components
el_drawer(
  "w2",
  title = "Filters",
  direction = "btt",
  size = "40%",
  content = shiny::tagList(el_input("q"), el_switch("live"))
)
if (interactive()) {
  # inside a server function
  observeEvent(input$go, {
    update_el_drawer(session, "settings", visible = TRUE)
  })
}
```

---

el_dropdown *Element Plus Dropdown Menu*

Description

A dropdown menu triggered by hover or click. Each menu item fires a command that is reported as a Shiny input.

Usage

```
el_dropdown(
  id = NULL,
  trigger_label = "Dropdown",
  items = list(),
  trigger = "hover",
  type = NULL,
  size = NULL,
  split_button = FALSE,
  hide_on_click = TRUE,
  placement = "bottom-end",
  disabled = FALSE,
  show_timeout = NULL,
  hide_timeout = NULL,
```

```
  tabindex = NULL,
  append_to = NULL,
  button_props = NULL,
  effect = NULL,
  max_height = NULL,
  persistent = NULL,
  popper_class = NULL,
  popper_options = NULL,
  popper_style = NULL,
  role = NULL,
  show_arrow = NULL,
  teleported = NULL,
  trigger_keys = NULL,
  virtual_ref = NULL,
  virtual_triggering = NULL,
  width = NULL,
  slots = NULL,
  events = NULL,
  on = NULL,
  session = NULL
)

update_el_dropdown(
  session = shiny::getDefaultReactiveDomain(),
  id,
  disabled = NULL,
  trigger = NULL,
  type = NULL,
  size = NULL,
  split_button = NULL,
  hide_on_click = NULL,
  placement = NULL,
  show_timeout = NULL,
  hide_timeout = NULL,
  tabindex = NULL,
  append_to = NULL,
  button_props = NULL,
  effect = NULL,
  max_height = NULL,
  persistent = NULL,
  popper_class = NULL,
  popper_options = NULL,
  popper_style = NULL,
  role = NULL,
  show_arrow = NULL,
  teleported = NULL,
  trigger_keys = NULL,
  virtual_ref = NULL,
```

```
  virtual_triggering = NULL
)
```

Arguments

id	Dropdown ID. Auto-generated UUID if NULL.
trigger_label	The dropdown's trigger. Text gets a down arrow after it; a tag – an icon, an avatar – is used as it is. Default "Dropdown".
items	A list of menu entries, each an el_dropdown_item() – or a named list with the same fields: **command** Command value sent to input$<id> on click. Required. **label** Display text. Defaults to command. **icon** Icon class string (e.g. "el-icon-edit"). Optional. **disabled** Whether the item is disabled. Default FALSE. **divided** Whether to show a divider above this item. Default FALSE.
trigger	Trigger event: "hover" (default) or "click".
type	Button type when split_button = TRUE: "primary", etc.
size	Size: "large", "default" or "small"; NULL follows the form or the page.
split_button	Whether to render as a split button (main + dropdown arrow). Default FALSE.
hide_on_click	Whether to close the menu after an item is clicked. Default TRUE.
placement	Dropdown placement: "bottom-end" (default), "bottom", "bottom-start", "top", "top-start", "top-end".
disabled	Whether the entire dropdown is disabled. Default FALSE.
show_timeout	Delay in ms before the menu appears, for trigger = "hover".
hide_timeout	Delay in ms before the menu hides, for trigger = "hover".
tabindex	Tab index of the dropdown trigger.
append_to	Which element the dropdown CONTENT appends to. Element Plus's append-to (CSSSelector / HTMLElement).
button_props	Props for the button component, refer to Button Attributes. Element Plus's button-props (object).
effect	Tooltip theme, built-in theme: dark / light. Element Plus's effect ('dark' | 'light' / string).
max_height	The max height of menu. Element Plus's max-height (string / number).
persistent	When dropdown inactive and persistent is false , dropdown menu will be destroyed. Element Plus's persistent (boolean).
popper_class	Custom class name for Dropdown's dropdown. Element Plus's popper-class (string / object).
popper_options	Popper.js parameters. Element Plus's popper-options (object).
popper_style	Custom style for Dropdown's dropdown. Element Plus's popper-style (string / object).
role	The ARIA role attribute for the dropdown menu. Depending on the use case, you may want to change this to 'navigation'. Element Plus's role (enum).

`show_arrow` Whether the tooltip content has an arrow. Element Plus's `show-arrow` (boolean).

`teleported` Whether the dropdown popup is teleported to the body. Element Plus's `teleported` (boolean).

`trigger_keys` Specify which keys on the keyboard can trigger when pressed. Element Plus's `trigger-keys` (`string[]`).

`virtual_ref` A CSS selector, `"#help-icon"`, for an element elsewhere on the page that the dropdown is attached to, in place of a trigger of its own. Element Plus's `virtual-ref` takes the element itself; the selector is looked up in the browser.

`virtual_triggering` Whether virtual triggering is enabled. Element Plus's `virtual-triggering` (boolean); `TRUE` when `virtual_ref` is given.

`width` Component width, as a CSS unit – `"200px"`, `"50%"`, or a number taken as pixels. Element's own markup carries it, so it behaves like the `width` argument of a Shiny input.

`slots` Named list of Element slot contents, such as `list(title = shiny::tags$b("Bold"))`. A shiny.element component given here is absorbed rather than nested. For a scoped slot, write the template with `template()`.

`events` Element's events to report besides those reported unasked, by name: `events = "node_drop"` reports `input$<id>_node_drop`. The component's are listed under "Shiny inputs", and by `el_events()`; a name it does not have is an error.

`on` Handlers of your own, for an event not reported or to send something else: a named list of `JS()` functions, one per event – Element's, or a DOM event with Vue's modifiers (`"keyup.enter"`). Each is called with `report` and the event's arguments; `report(name, value)` sets `input$<id>_<name>`. See `el_widget()`.

`session` In `el_dropdown()`, deprecated: inside a module, wrap `id` in `ns()`, as for any Shiny input; a session given here namespaces `id` once more, with a warning. In `update_el_dropdown()`, the Shiny session, the current one by default, as for `shiny::updateTextInput()`.

Value

An `htmltools` tagList with a Vue-managed dropdown component.

Shiny inputs

Input	Reported	Value
`input$<id>`	unasked	the `command` of the item clicked, as an event
`input$<id>_count`	unasked	the number of items clicked
`input$<id>_click`	unasked	if split-button is true, triggers when left button is clicked
`input$<id>_visible_change`	`events = "visible_change"`	triggers when the dropdown appears/disappears, the param

The same list as `el_events("el_dropdown")`, which says how an event's arguments travel.

Updating from the server

Server-side update for `el_dropdown()`.

Every other argument of `el_dropdown()` that can change once it is drawn is an argument here too, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_dropdown()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_dropdown(
  "dd1",
  "Actions",
  items = list(
    list(command = "edit", label = "Edit", icon = "el-icon-edit"),
    list(command = "copy", label = "Copy", icon = "el-icon-document"),
    list(
      command = "delete",
      label = "Delete",
      icon = "el-icon-delete",
      divided = TRUE
    )
  )
)
if (interactive()) {
  # inside a server function
  observeEvent(input$go, {
    update_el_dropdown(session, "actions", disabled = TRUE)
  })
}
```

`el_dropdown_item` *An entry of* `el_dropdown()`*'s menu*

Description

Element Plus's `el-dropdown-item`, for `el_dropdown(items =)`.

Usage

```
el_dropdown_item(
  command,
  label = command,
  icon = NULL,
  disabled = NULL,
  divided = NULL
)
```

Arguments

command	The value input$<id> takes when the entry is clicked.
label	The entry's text. Defaults to command.
icon	The entry's icon, by name.
disabled	Whether the entry is disabled.
divided	Whether a divider is drawn above the entry.

Value

An entry, for el_dropdown(items =).

See Also

Other items: el_anchor_link(), el_breadcrumb_item(), el_carousel_item(), el_collapse_item(), el_descriptions_item(), el_menu_item(), el_option(), el_skeleton_item(), el_step(), el_tab_pane(), el_table_column(), el_table_v2_column(), el_timeline_item(), el_tour_step()

Examples

```
el_dropdown(
  "dd",
  "Dropdown List",
  items = list(
    el_dropdown_item("apple", "Apple"),
    el_dropdown_item("pear", "Pear", disabled = TRUE),
    el_dropdown_item("plum", "Plum", divided = TRUE)
  )
)
```

---

el_empty *Element Plus Empty*

Description

A placeholder for a view with nothing in it yet.

Usage

```
el_empty(
  id = NULL,
  ...,
  description = NULL,
  image = NULL,
  image_size = NULL,
  width = NULL,
  slots = NULL,
  on = NULL,
```

```
  session = NULL
)

update_el_empty(
  session = shiny::getDefaultReactiveDomain(),
  id,
  description = NULL,
  image = NULL
)
```

Arguments

id	Component ID. Auto-generated if NULL.
...	Content under the description – usually a button that fixes the emptiness. A shiny.element component here is absorbed, not nested.
description	Text under the picture. NULL for Element's own ("No Data", in the page's locale).
image	URL of a picture to use instead of Element's.
image_size	Width of the picture, in pixels.
width	Component width, as a CSS unit.
slots	Named list of Element slot contents: image, description.
on	Handlers of your own, for an event not reported or to send something else: a named list of JS() functions, one per event – Element's, or a DOM event with Vue's modifiers ("keyup.enter"). Each is called with report and the event's arguments; report(name, value) sets input$<id>_<name>. See el_widget().
session	In el_empty(), deprecated: inside a module, wrap id in ns(), as for any Shiny input; a session given here namespaces id once more, with a warning. In update_el_empty(), the Shiny session, the current one by default, as for shiny::updateTextInput().

Value

A Shiny UI element.

Updating from the server

Server-side update for el_empty().

update_el_empty() is called for its side effect and returns NULL invisibly.

Examples

```
el_empty("none", description = "No reports yet")

el_empty(
  "none",
  description = "No reports yet",
  el_button("create", "Create one", type = "primary")
)
```

```
if (interactive()) {
  # inside a server function
  observeEvent(input$search, {
    update_el_empty(
      session,
      "none",
      description = paste("Nothing matches", input$search)
    )
  })
}
```

`el_events` *The Shiny inputs a component reports*

Description

Every input a component can report, from one list: the ones it always reports – its value, and the requests the server answers – and the Element Plus events it forwards, some of them unasked (`default`) and the rest when asked for with the component's `events` argument. Each is `input$<id>_<event>`, the event in snake_case, namespaced in a module as the component's id is.

Usage

```
el_events(component)
```

Arguments

`component`	A component function, or its name: `el_tree`, `"el_tree"` or `"tree"`.

Details

A component's help page lists the same in its "Shiny inputs" section, and an unknown name given to `events` is an error that lists them.

For an event not in the list – a key with a modifier, a DOM event on the element Element draws – or to send something else than what an event carries, give the component a handler of your own with `on`: see `el_widget()`.

An event's arguments are sent as they are when nothing is said in `value`: one as itself, several as `list(arg1, arg2, ...)`, none as `TRUE`. Arguments that cannot travel – DOM nodes, a component's internal objects – are dropped; a keyboard event is sent as `list(key, code, ctrl, shift, alt, meta)`. Every event input is sent with `priority = "event"`: an observer runs even when the value is the same as before.

Value

A data frame, one row per input, of class `el_events`:

input The input, `input$<id>` or `input$<id>_<name>`.

event What to give `events` to have it reported: Element's event in snake_case; `NA` for an input always reported.

default `TRUE` when it is reported without being asked for.

about When it is sent: Element's description of the event.

value What it holds. Empty for an event whose arguments are sent as they are (see Details).

args The event's arguments in Element.

Examples

```
el_events("el_tree")
el_events(el_input)

# asking for an event
el_input("q", events = "keydown") # input$q_keydown
```

`el_feedback_close` *Close a message or a notification*

Description

Closes the one opened under `id`, or every one currently showing when `id` is `NULL` – Element's `close()` and `closeAll()`.

Usage

```
el_message_close(session = shiny::getDefaultReactiveDomain(), id = NULL)

el_notification_close(session = shiny::getDefaultReactiveDomain(), id = NULL)
```

Arguments

`session`	Shiny session; the current one by default, as for `shiny::updateTextInput()`.
`id`	The `id` it was opened with, or `NULL` for all of them.

Value

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

Examples

```
if (interactive()) {
  # inside a server function
  observeEvent(input$start, {
    el_notification(session, "Uploading...", duration = 0, id = "up")
  })
  observeEvent(input$done, {
    el_notification_close(session, "up")
  })
}
```

el_feedback_dependency *Feedback Handler Dependency*

Description

Loads the JavaScript handlers for `el_notification()` and `el_message()`. Automatically included by `use_element()` and `el_page()`.

Usage

```
el_feedback_dependency()
```

Value

An htmlDependency object.

Examples

```
el_feedback_dependency()
```

el_footer *Element Plus Footer*

Description

Element Plus Footer

Usage

```
el_footer(..., height = "60px", style = NULL, class = NULL)
```

Arguments

`...`	Content.
`height`	Footer height. Defaults to `"60px"`, as in Element Plus, which sets it as the CSS variable `--el-<part>-<size>`.
`style`	Extra inline style.
`class`	Extra CSS classes.

Value

A Shiny UI element.

Examples

```
el_footer("(c) 2026")
el_footer(height = "40px", "Compact footer")
```

`el_form` *Element Plus Form*

Description

A form that owns its state, the way Element Plus intends: one Vue instance holding a `model` of all field values plus the validation rules, with async-validator running them on blur or change.

Usage

```
el_form(
  ...,
  id = NULL,
  label_width = "100px",
  label_position = "right",
  inline = FALSE,
  size = NULL,
  submit_label = "Submit",
  reset_label = NULL,
  disabled = NULL,
  show_message = NULL,
  inline_message = NULL,
  status_icon = NULL,
  hide_required_asterisk = NULL,
  label_suffix = NULL,
  validate_on_rule_change = NULL,
  require_asterisk_position = NULL,
  scroll_into_view_options = NULL,
  scroll_to_error = NULL,
  width = NULL,
  slots = NULL,
  events = NULL,
  on = NULL,
  session = NULL
)

update_el_form(
  session = shiny::getDefaultReactiveDomain(),
  id,
  model = NULL,
  rules = NULL,
  label_width = NULL,
  label_position = NULL,
  fields = NULL,
  errors = NULL,
  inline = NULL,
  size = NULL,
```

```
  submit_label = NULL,
  reset_label = NULL,
  disabled = NULL,
  status_icon = NULL,
  hide_required_asterisk = NULL,
  validate_on_rule_change = NULL,
  require_asterisk_position = NULL,
  scroll_into_view_options = NULL,
  scroll_to_error = NULL
)
```

Arguments

`...`	Fields, from `el_form_field()`.
`id`	Form ID (auto-generated if NULL).
`label_width`	Label column width, e.g. `"100px"`.
`label_position`	`"right"` (default), `"left"` or `"top"`.
`inline`	Lay the fields out in a row.
`size`	Size: `"large"`, `"default"` or `"small"`; `NULL` follows the form or the page.
`submit_label`	Submit button text. `NULL` renders no button, in which case drive the form with `el_form_validate()`.
`reset_label`	Reset button text. `NULL` renders no button.
`disabled`	Whether every control in the form is disabled.
`show_message`	Whether to show validation messages. Default `TRUE`.
`inline_message`	Whether to show validation messages inline.
`status_icon`	Whether to show a validation status icon in each field.
`hide_required_asterisk`	Whether to hide the asterisk next to required fields' labels.
`label_suffix`	Suffix appended to every label.
`validate_on_rule_change`	Whether changing the rules triggers validation immediately.
`require_asterisk_position`	Position of asterisk. Element Plus's `require-asterisk-position` ('left' | 'right').
`scroll_into_view_options`	When validation fails, it scrolls to the first error item based on the scrollIntoView option. scrollIntoView. Element Plus's `scroll-into-view-options` (ScrollIntoViewOptions / boolean).
`scroll_to_error`	When validation fails, scroll to the first error form entry. Element Plus's `scroll-to-error` (boolean).
`width`	Component width, as a CSS unit – `"200px"`, `"50%"`, or a number taken as pixels. Element's own markup carries it, so it behaves like the `width` argument of a Shiny input.

slots — Named list of Element slot contents, such as `list(title = shiny::tags$b("Bold"))`. A shiny.element component given here is absorbed rather than nested. For a scoped slot, write the template with `template()`.

events — Element's events to report besides those reported unasked, by name: `events = "node_drop"` reports `input$<id>_node_drop`. The component's are listed under "Shiny inputs", and by `el_events()`; a name it does not have is an error.

on — Handlers of your own, for an event not reported or to send something else: a named list of `JS()` functions, one per event – Element's, or a DOM event with Vue's modifiers (`"keyup.enter"`). Each is called with `report` and the event's arguments; `report(name, value)` sets `input$<id>_<name>`. See `el_widget()`.

session — In `el_form()`, deprecated: inside a module, wrap `id` in `ns()`, as for any Shiny input; a session given here namespaces `id` once more, with a warning. In `update_el_form()`, the Shiny session, the current one by default, as for `shiny::updateTextInput()`.

model — New field values. Merged into the existing model, so a partial list only changes the fields it names.

rules — New validation rules, as a named list of `el_rule()` lists keyed by `prop`. Replaces the rule set.

fields — The form's fields, as `el_form_field()`s – the whole list, in order, so a field can be added, removed or moved: Element's "add or delete form items dynamically". A field already in the form keeps what was entered; a new one starts from its `value`; a removed one leaves the model. Each field's rules come with it.

errors — Error messages from the server, as a named list keyed by `prop` – `list(email = "That address is taken")` – shown on the field as Element's `error` shows them. `""` clears one.

Details

Fields are declared with `el_form_field()` rather than composed from the package's standalone input components. Those are each their own Vue instance, which puts them outside the form-item's component tree, where Element's event chain and shared model cannot reach them.

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
`input$<id>`	unasked	the whole model, on load and on every submit
`input$<id>_valid`	unasked	`TRUE` when the last submit passed validation
`input$<id>_submit`	unasked	a submit counter
`input$<id>_validate`	`events = "validate"`	triggers after a form item is validated

The same list as `el_events("el_form")`, which says how an event's arguments travel.

The model is deliberately *not* sent on every keystroke: that is the point of the form owning its state rather than each field reporting separately.

Element methods

Callable with `call_el()`:

- `clearValidate()` – Clear validation message for certain fields. The parameter is prop name or an array of prop names of the...
- `clearValidate()` – Remove validation status of the field
- `resetField()` – Reset current field and remove validation result
- `resetFields()` – Reset all the fields and remove validation result
- `validate()` – Validate the whole form. Takes a callback as a param. After validation, the callback will be executed with...
- `validateField()` – Validate one or several form items

Updating from the server

`update_el_form()` changes the component from the server.

Every other argument of `el_form()` that can change once it is drawn is an argument here too, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_form()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_form(
  id = "signup",
  label_width = "100px",
  el_form_field(
    "name",
    "input",
    label = "Name",
    rules = el_rule(required = TRUE, message = "Name is required")
  ),
  el_form_field(
    "age",
    "input-number",
    label = "Age",
    value = 18,
    min = 0,
    max = 150
  ),
  el_form_field(
    "city",
    "select",
    label = "City",
    choices = c(Beijing = "bj", Shanghai = "sh"),
    rules = el_rule(
      required = TRUE,
```

```
      message = "Pick a city",
      trigger = "change"
    )
  )
)

if (interactive()) {
  library(shiny)
  library(shiny.element)
  ui <- el_page(
    el_form(
      id = "signup",
      label_width = "100px",
      reset_label = "Reset",
      el_form_field(
        "name",
        "input",
        label = "Name",
        rules = el_rule(required = TRUE, message = "Required")
      ),
      el_form_field(
        "email",
        "input",
        label = "Email",
        rules = el_rule(type = "email", message = "Invalid email")
      )
    ),
    verbatimTextOutput("out")
  )
  server <- function(input, output, session) {
    output$out <- renderPrint({
      req(input$signup_submit)
      if (!isTRUE(input$signup_valid)) {
        return("Please fix the errors above")
      }
      input$signup
    })
  }
  shinyApp(ui, server)
}
if (interactive()) {
  # Prefill the form from the server
  update_el_form(session, "signup", model = list(name = "Ada", age = 36))

  # A check only the server can make
  update_el_form(
    session,
    "signup",
    errors = list(email = "That address is taken")
  )
}
```

el_form_clear_validate *Clear an Element Plus Form's validation messages*

Description

Leaves the values alone and only removes the error state.

Usage

```
el_form_clear_validate(
  session = shiny::getDefaultReactiveDomain(),
  id,
  props = NULL
)
```

Arguments

session	Shiny session; the current one by default, as for shiny::updateTextInput().
id	Form ID (un-namespaced).
props	Fields to clear. NULL clears all of them.

Value

Called for its side effect; returns NULL invisibly.

Examples

```
if (interactive()) {
  # inside a server function
  observeEvent(input$go, {
    el_form_clear_validate(session, "signup")
  })
}
```

el_form_field *Declare a form field*

Description

Declare a form field

Usage

```
el_form_field(
  prop,
  type = "input",
  label = NULL,
  value = NULL,
  choices = NULL,
  rules = NULL,
  report = FALSE,
  button = FALSE,
  border = NULL,
  ...,
  options = NULL
)
```

Arguments

prop	Field name. Keys the form's model and is what el_rule()s and validation messages refer to.
type	Control type: one of "input", "textarea", "password", "input-number", "select", "radio-group", "checkbox-group", "checkbox" (one box, TRUE or FALSE), "switch", "slider", "date-picker", "time-picker", "time-select", "rate", "cascader", "cascader-panel", "color-picker", "autocomplete", "transfer", "select-v2", "segmented", "input-tag", "mention" or "mention-textarea". Their props go in ...: a cascader's options, a transfer's data.
label	Label text.
value	Initial value. Defaults to the type's empty value, which is also what resetFields() restores.
choices, options	Options for "select", "radio-group" and "checkbox-group"; suggestions for "autocomplete", filtered as the user types. A named vector c(Label = value) or a list of list(value=, label=). For "checkbox", the box's text is label. choices is Shiny's name, options Element's; give either, as to el_select().
rules	A single el_rule() or a list of them.
report	Whether the field also reports its value on load and on every change, as input$<form id>_<prop>, as a standalone input would – for a control that changes the page, such as a size switch. The form's own input$<form id> still waits for a submit. A field added later by update_el_form()(fields =) reports with the submit only.
button, border	For "radio-group" and "checkbox-group": draw the choices as buttons (el-radio-button, el-checkbox-button), or as boxes with a border.
...	Further props. Element's form-item props – required, error, label_width, label_position, size, show_message, inline_message, validate_status, for, and label_html/error_html for markup of your own, a string or htmltools tags – go on the field's form item; anything else on the control, e.g. placeholder, min, max, disabled. Names are converted to camelCase.

Value

A field declaration, for `el_form()`.

Examples

```
el_form_field(
  "name",
  "input",
  label = "Name",
  rules = el_rule(required = TRUE, message = "Required")
)
el_form_field(
  "age",
  "input-number",
  label = "Age",
  value = 18,
  min = 0,
  max = 150
)
el_form_field(
  "city",
  "select",
  label = "City",
  choices = c(Beijing = "bj", Shanghai = "sh")
)
```

---

el_form_item *Several form fields under one label*

Description

Element's form item holding more than one control, each with a form item of its own for its value and rules, laid out in columns – a date beside a time, as Element's own examples write it.

Usage

```
el_form_item(label = NULL, ..., spans = NULL, required = NULL, gutter = NULL)
```

Arguments

`label`	Label text.
`...`	Fields, from `el_form_field()` (whose `label` is not shown), and strings drawn between them, such as `"-"`.
`spans`	Column spans out of 24, one per entry in `...`. By default a string takes 2 and the fields share the rest.
`required`	Whether to mark the label as required. The fields' own `el_rule()`s do the checking.
`gutter`	Space between the columns, in pixels, as `el_row()`'s.

Value

A field declaration, for `el_form()`.

Examples

```
el_form(
  el_form_item(
    "Activity time",
    el_form_field("date1", "date-picker", style = "width: 100%"),
    "-",
    el_form_field("date2", "time-picker", style = "width: 100%")
  )
)
```

---

el_form_reset *Reset an Element Plus Form*

Description

Restores every field to the `value` it was declared with and clears the validation state. Note this is Element Plus's `resetFields()`: it restores the *initial* values, not empty ones, so a field declared with `value = 18` resets to 18.

Usage

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

Arguments

`session`	Shiny session; the current one by default, as for `shiny::updateTextInput()`.
`id`	Form ID (un-namespaced).

Value

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

Examples

```
if (interactive()) {
  # inside a server function
  observeEvent(input$go, {
    el_form_reset(session, "signup")
  })
}
```

el_form_validate *Validate an Element Plus Form from the server*

Description

Runs the form's rules and reports the outcome the same way a submit does, so the same `observeEvent` handles both. Use it when the form has no submit button of its own (`submit_label = NULL`).

Usage

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

Arguments

`session`	Shiny session; the current one by default, as for `shiny::updateTextInput()`.
`id`	Form ID (un-namespaced).

Value

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

Examples

```
if (interactive()) {
  # inside a server function
  observeEvent(input$go, {
    el_form_validate(session, "signup")
  })
}
```

el_header *Element Plus Header*

Description

Element Plus Header

Usage

```
el_header(..., height = "60px", style = NULL, class = NULL)
```

Arguments

`...`	Content.
`height`	Header height. Defaults to `"60px"`, as in Element Plus, which sets it as the CSS variable `--el-<part>-<size>`.
`style`	Extra inline style.
`class`	Extra CSS classes.

Value

A Shiny UI element.

Examples

```
el_header("Dashboard")
el_header(height = "80px", el_button("refresh", "Refresh"))
```

`el_icon` *Element Plus icon*

Description

An icon from Element Plus's set, Font Awesome, or a plain tag. Follows the same dispatch pattern as `shiny::icon()`, with accessibility attributes inspired by `bsicons::bs_icon()`.

Usage

```
el_icon(
  name,
  size = NULL,
  class = NULL,
  title = NULL,
  a11y = c("auto", "deco", "sem", "none"),
  lib = c("element-plus", "font-awesome", "none"),
  ...,
  color = NULL
)
```

Arguments

`name`	Icon name, as Element Plus spells it – `"Search"`, `"ArrowRight"` – or in any of the forms that reach the same name: `"search"`, `"arrow-right"`, `"arrow right"`, and Element UI's `"el-icon-arrow-right"`.
`size`	CSS size string (e.g. `"1.5em"`, `"20px"`), as Element Plus's `<el-icon size>`. `NULL` (default) follows the surrounding text.
`class`	Additional CSS class(es) to append.
`title`	Accessible title string. When provided, it also drives `a11y` (see below).
`a11y`	Accessibility mode. One of: `"auto"` **(default)** `"deco"` when `title` is `NULL`, `"sem"` otherwise. `"deco"` Decorative icon: adds `aria-hidden="true"` and `role="img"`. `"sem"` Semantic icon: adds `aria-label` (using `title` or `name`) and `role="img"`. `"none"` No accessibility attributes added.
`lib`	Icon library. One of:

	`"element-plus"` **(default)** Element Plus's icon set. `"font-awesome"` Delegates to `fontawesome::fa_i()`. Requires the `fontawesome` package. `"none"` Renders a plain `<i>` tag with no icon class.
`...`	Additional HTML attributes passed to the `<i>` tag.
`color`	Icon colour, as Element Plus's `<el-icon color>`. `NULL` follows the surrounding text.

Details

Element Plus's icons are SVG components (`@element-plus/icons-vue`, bundled): the tag is an `<i class="el-icon">` naming its icon, drawn by the page wherever it lands – inside a component or not.

Value

An `htmltools` tag object.

Examples

```
el_icon("Search")
el_icon("edit", size = "1.5em")
el_icon("Delete", title = "Delete item", color = "#f56c6c")
el_icon("close", a11y = "deco")
```

---

el_image *Element Plus Image*

Description

An image with a fit mode, optional lazy loading, and an optional full-screen preview.

Usage

```
el_image(
  id = NULL,
  src = NULL,
  fit = NULL,
  alt = NULL,
  lazy = NULL,
  scroll_container = NULL,
  preview_src_list = NULL,
  z_index = NULL,
  initial_index = NULL,
  close_on_press_escape = NULL,
  crossorigin = NULL,
  hide_on_click_modal = NULL,
```

```
  infinite = NULL,
  loading = NULL,
  max_scale = NULL,
  min_scale = NULL,
  preview_teleported = NULL,
  referrerpolicy = NULL,
  scale = NULL,
  show_progress = NULL,
  zoom_rate = NULL,
  width = NULL,
  class = NULL,
  style = NULL,
  slots = NULL,
  events = NULL,
  on = NULL,
  session = NULL
)

update_el_image(
  session = shiny::getDefaultReactiveDomain(),
  id,
  src = NULL,
  fit = NULL,
  preview_src_list = NULL,
  alt = NULL,
  lazy = NULL,
  scroll_container = NULL,
  z_index = NULL,
  initial_index = NULL,
  close_on_press_escape = NULL,
  crossorigin = NULL,
  hide_on_click_modal = NULL,
  infinite = NULL,
  loading = NULL,
  max_scale = NULL,
  min_scale = NULL,
  preview_teleported = NULL,
  referrerpolicy = NULL,
  scale = NULL,
  show_progress = NULL,
  zoom_rate = NULL
)
```

Arguments

id	Image ID. Auto-generated if NULL.
src	Image URL.
fit	How the image fills its box: "fill", "contain", "cover", "none" or "scale-down".

`alt` Alternative text.

`lazy` Whether to load the image only once it scrolls into view.

`scroll_container` CSS selector of the scrolling element to watch when `lazy = TRUE`. `NULL` watches the nearest scrollable parent.

`preview_src_list` Character vector of image URLs to show in a full-screen preview when the image is clicked.

`z_index` Stacking order of the preview. Default `2000`.

`initial_index` Which image of `preview_src_list` the preview opens on, 0-based.

`close_on_press_escape` Whether the image-viewer can be closed by pressing ESC. Element Plus's `close-on-press-escape` (boolean).

`crossorigin` Native attribute crossorigin. Element Plus's `crossorigin` ('' | 'anonymous' | 'use-credentials').

`hide_on_click_modal` When enabling preview, use this flag to control whether clicking on backdrop can exit preview mode. Element Plus's `hide-on-click-modal` (boolean).

`infinite` Whether the viewer preview is infinite. Element Plus's `infinite` (boolean).

`loading` Indicates how the browser should load the image, same as native. Element Plus's `loading` ('eager' | 'lazy').

`max_scale` The max scale of the image viewer zoom event. Element Plus's `max-scale` (number).

`min_scale` The min scale of the image viewer zoom event. Element Plus's `min-scale` (number).

`preview_teleported` Whether to append image-viewer to body. A nested parent element attribute transform should have this attribute set to `true`. Element Plus's `preview-teleported` (boolean).

`referrerpolicy` Native attribute referrerPolicy. Element Plus's `referrerpolicy` (string).

`scale` The preview image scale. Element Plus's `scale` (number).

`show_progress` Whether to display the preview image progress content. Element Plus's `show-progress` (boolean).

`zoom_rate` The zoom rate of the image viewer zoom event. Element Plus's `zoom-rate` (number).

`width` Component width, as a CSS unit.

`class, style` Extra classes and inline style on the image's box, as Element passes them through: `style = "width: 100px; height: 100px"` gives `fit` a box to fit the picture to.

`slots` Named list of Element slot contents, such as `list(title = shiny::tags$b("Bold"))`. A shiny.element component given here is absorbed rather than nested. For a scoped slot, write the template with `template()`.

`events` Element's events to report besides those reported unasked, by name: `events = "node_drop"` reports `input$<id>_node_drop`. The component's are listed under "Shiny inputs", and by `el_events()`; a name it does not have is an error.

on — Handlers of your own, for an event not reported or to send something else: a named list of `JS()` functions, one per event – Element's, or a DOM event with Vue's modifiers (`"keyup.enter"`). Each is called with `report` and the event's arguments; `report(name, value)` sets `input$<id>_<name>`. See `el_widget()`.

session — In `el_image()`, deprecated: inside a module, wrap `id` in `ns()`, as for any Shiny input; a session given here namespaces `id` once more, with a warning. In `update_el_image()`, the Shiny session, the current one by default, as for `shiny::updateTextInput()`.

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
`input$<id>_load`	`events = "load"`	same as native load.
`input$<id>_error`	unasked	same as native error.
`input$<id>_close`	`events = "close"`	trigger when clicking on close button or when hide-on-click-modal enabled clic
`input$<id>_show`	`events = "show"`	trigger when the viewer displays
`input$<id>_switch`	`events = "switch"`	trigger when switching images.

The same list as `el_events("el_image")`, which says how an event's arguments travel.

Updating from the server

Server-side update for `el_image()`.

Every other argument of `el_image()` that can change once it is drawn is an argument here too, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_image()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_image("photo", src = "https://example.org/a.png", width = 200)
el_image("photo", src = "a.png", fit = "cover", lazy = TRUE)

# Click to open a gallery
el_image(
  "photo",
  src = "a.png",
  preview_src_list = c("a.png", "b.png", "c.png")
)
if (interactive()) {
  # inside a server function
  observeEvent(input$next_photo, {
    update_el_image(session, "photo", src = photo_url())
  })
}
```

el_image_viewer *Element Plus Image Viewer*

Description

A full-screen viewer for a list of images, with zoom and rotation.

Usage

```
el_image_viewer(
  id = NULL,
  url_list = NULL,
  visible = FALSE,
  z_index = NULL,
  initial_index = NULL,
  infinite = NULL,
  hide_on_click_modal = NULL,
  teleported = NULL,
  zoom_rate = NULL,
  scale = NULL,
  min_scale = NULL,
  max_scale = NULL,
  close_on_press_escape = NULL,
  show_progress = NULL,
  width = NULL,
  slots = NULL,
  events = NULL,
  on = NULL
)

update_el_image_viewer(
  session = shiny::getDefaultReactiveDomain(),
  id,
  visible = NULL,
  url_list = NULL,
  initial_index = NULL,
  z_index = NULL,
  infinite = NULL,
  hide_on_click_modal = NULL,
  teleported = NULL,
  zoom_rate = NULL,
  scale = NULL,
  min_scale = NULL,
  max_scale = NULL,
  close_on_press_escape = NULL,
  show_progress = NULL
)
```

Arguments

id	Component ID. Auto-generated if NULL.
url_list	Preview link list. Element Plus's url-list (string[]).
visible	Whether it starts open. Open and close it later with update_el_image_viewer().
z_index	Preview backdrop z-index. Element Plus's z-index (number / string).
initial_index	The initial preview image index, less than or equal to the length of url-list. Element Plus's initial-index (number).
infinite	Whether preview is infinite. Element Plus's infinite (boolean).
hide_on_click_modal	Whether user can emit close event when clicking backdrop. Element Plus's hide-on-click-modal (boolean).
teleported	Whether to append image itself to body. A nested parent element attribute transform should have this attribute set to true. Element Plus's teleported (boolean).
zoom_rate	The zoom rate of the image viewer zoom event. Element Plus's zoom-rate (number).
scale	The preview image scale. Element Plus's scale (number).
min_scale	The min scale of the image viewer zoom event. Element Plus's min-scale (number).
max_scale	The max scale of the image viewer zoom event. Element Plus's max-scale (number).
close_on_press_escape	Whether the image-viewer can be closed by pressing ESC. Element Plus's close-on-press-escape (boolean).
show_progress	Whether to display the preview image progress content. Element Plus's show-progress (boolean).
width	Component width, as a CSS unit.
slots	Named list of Element slot contents: progress, toolbar, viewer-error. A scoped slot is written with template().
events	Element's events to report besides those reported unasked, by name: events = "node_drop" reports input$<id>_node_drop. The component's are listed under "Shiny inputs", and by el_events(); a name it does not have is an error.
on	Handlers of your own, for an event not reported or to send something else: a named list of JS() functions, one per event – Element's, or a DOM event with Vue's modifiers ("keyup.enter"). Each is called with report and the event's arguments; report(name, value) sets input$<id>_<name>. See el_widget().
session	Shiny session; the current one by default, as for shiny::updateTextInput().

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
`input$<id>_close`	unasked	fires as it closes
`input$<id>_error`	unasked	same as native error.
`input$<id>_switch`	`events = "switch"`	trigger when switching images.
`input$<id>_rotate`	`events = "rotate"`	trigger when rotating images.

The same list as `el_events("el_image_viewer")`, which says how an event's arguments travel.

Element methods

Callable with `call_el()`: `setActiveItem()`.

Updating from the server

Open or close an `el_image_viewer()`, or give it other images.

Every other argument of `el_image_viewer()` that can change once it is drawn is an argument here too, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_image_viewer()` is called for its side effect and returns `NULL` invisibly.

Examples

```
if (interactive()) {
  # inside a server function
  observeEvent(
    input$show,
    update_el_image_viewer(session, "photos", visible = TRUE)
  )
}
```

---

el_infinite_scroll *Element Plus Infinite Scroll*

Description

A scrolling area that asks the server for more as the user nears the bottom. Element implements this as a directive rather than a component, so the area is a container you put content into.

Usage

```
el_infinite_scroll(
  id = NULL,
  ...,
  height = "300px",
  disabled = NULL,
```

```
  delay = NULL,
  distance = NULL,
  immediate = NULL,
  width = NULL,
  slots = NULL,
  on = NULL,
  session = NULL
)

update_el_infinite_scroll(
  session = shiny::getDefaultReactiveDomain(),
  id,
  disabled = NULL,
  delay = NULL,
  distance = NULL
)
```

Arguments

id	Container ID. Auto-generated if NULL.
...	Content of the scrolling area. Any Shiny UI, including shiny.element components – those are folded into this container's Vue instance rather than nested inside it, so their inputs keep reporting. Their update_el_*() no longer reaches them, though.
height	Height of the area, as a CSS unit. Needed for it to scroll at all. Default "300px".
disabled	Whether loading is suspended. Set it from the server while a request is in flight, and again when there is nothing left to fetch.
delay	Throttle between checks, in milliseconds. Default 200.
distance	How near the bottom to get before asking, in pixels. Default 0.
immediate	Whether to ask once on load, in case the content does not fill the area. Default TRUE.
width	Component width, as a CSS unit.
slots	Named list of Element slot contents, such as list(title = shiny::tags$b("Bold")). A shiny.element component given here is absorbed rather than nested. For a scoped slot, write the template with template().
on	Handlers of your own, for an event not reported or to send something else: a named list of JS() functions, one per event – Element's, or a DOM event with Vue's modifiers ("keyup.enter"). Each is called with report and the event's arguments; report(name, value) sets input$<id>_<name>. See el_widget().
session	In el_infinite_scroll(), deprecated: inside a module, wrap id in ns(), as for any Shiny input; a session given here namespaces id once more, with a warning. In update_el_infinite_scroll(), the Shiny session, the current one by default, as for shiny::updateTextInput().

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
`input$<id>_load`	unasked	rises by one each time more content is wanted

The same list as `el_events("el_infinite_scroll")`, which says how an event's arguments travel.

Observe `input$<id>_load`, fetch the next page, and render it into a `shiny::uiOutput()` inside the area.

Updating from the server

Server-side update for `el_infinite_scroll()`. Setting `disabled` is how a feed stops asking once everything has been sent.

`update_el_infinite_scroll()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_infinite_scroll("feed", shiny::uiOutput("rows"), height = "400px")

if (interactive()) {
  library(shiny)
  library(shiny.element)

  ui <- el_page(el_infinite_scroll("feed", uiOutput("rows")))

  server <- function(input, output, session) {
    shown <- reactiveVal(20)
    observeEvent(input$feed_load, {
      shown(min(shown() + 20, nrow(iris)))
    })
    output$rows <- renderUI({
      lapply(seq_len(shown()), function(i) tags$p(paste("Row", i)))
    })
  }
  shinyApp(ui, server)
}
if (interactive()) {
  # inside a server function
  observeEvent(input$feed_load, {
    if (all_rows_sent()) {
      update_el_infinite_scroll(session, "feed", disabled = TRUE)
    }
  })
}
```

el_input *Element Plus Input with Vue Instance*

Description

Creates an Element Plus input component (<el-input>) with a Vue instance, supporting text, textarea, and password modes, plus clearable, prefix/suffix icons, word-limit display, and autosize textarea.

Usage

```
el_input(
  id = NULL,
  value = "",
  placeholder = NULL,
  type = "text",
  size = NULL,
  disabled = FALSE,
  readonly = FALSE,
  clearable = FALSE,
  show_password = FALSE,
  show_word_limit = FALSE,
  maxlength = NULL,
  rows = NULL,
  autosize = FALSE,
  prefix_icon = NULL,
  suffix_icon = NULL,
  label = NULL,
  autocomplete = NULL,
  autofocus = NULL,
  name = NULL,
  form = NULL,
  minlength = NULL,
  max = NULL,
  min = NULL,
  step = NULL,
  resize = NULL,
  tabindex = NULL,
  validate_event = NULL,
  label_position = c("top", "left", "right"),
  label_width = NULL,
  label_suffix = NULL,
  required = FALSE,
  error = NULL,
  show_message = TRUE,
  inline_message = FALSE,
  aria_label = NULL,
```

```
  clear_icon = NULL,
  count_graphemes = NULL,
  formatter = NULL,
  input_style = NULL,
  inputmode = NULL,
  parser = NULL,
  word_limit_position = NULL,
  width = NULL,
  slots = NULL,
  events = NULL,
  on = NULL,
  session = NULL
)

update_el_input(
  session = shiny::getDefaultReactiveDomain(),
  id,
  value = NULL,
  placeholder = NULL,
  disabled = NULL,
  readonly = NULL,
  type = NULL,
  size = NULL,
  clearable = NULL,
  show_password = NULL,
  label = NULL,
  error = NULL,
  show_word_limit = NULL,
  maxlength = NULL,
  rows = NULL,
  autosize = NULL,
  prefix_icon = NULL,
  suffix_icon = NULL,
  autocomplete = NULL,
  autofocus = NULL,
  name = NULL,
  form = NULL,
  minlength = NULL,
  max = NULL,
  min = NULL,
  step = NULL,
  resize = NULL,
  tabindex = NULL,
  validate_event = NULL,
  aria_label = NULL,
  clear_icon = NULL,
  count_graphemes = NULL,
  formatter = NULL,
```

```
  input_style = NULL,
  inputmode = NULL,
  parser = NULL,
  word_limit_position = NULL
)
```

Arguments

id	Input ID. Auto-generated UUID if NULL.
value	Initial input value. Default "".
placeholder	Placeholder text. NULL means no placeholder attribute.
type	Input type: "text" (default), "textarea", "password".
size	Size: "large", "default" or "small"; NULL follows the form or the page.
disabled	Whether the input is disabled. Default FALSE.
readonly	Whether the input is read-only. Default FALSE.
clearable	Whether to show a clear button. Default FALSE.
show_password	Whether to show the password toggle icon. Only meaningful when type = "password". Default FALSE.
show_word_limit	Whether to show a word-count badge. Requires maxlength to be set. Default FALSE.
maxlength	Maximum character count. NULL means no limit.
rows	Number of rows for type = "textarea". NULL uses the default.
autosize	Whether to auto-size the textarea height. Either TRUE/FALSE or a named list list(minRows = 2, maxRows = 4). Default FALSE.
prefix_icon	Icon class for the prefix slot (e.g. "el-icon-search"). NULL means no icon.
suffix_icon	Icon class for the suffix slot (e.g. "el-icon-date"). NULL means no icon.
label	A label shown with the component, as Shiny's inputs have: text or a tag. NULL, the default, shows none. It is the component's accessible name too – tied to it with for where the component has a native input that takes the id <id>-input, else with aria-labelledby.
autocomplete	Native autocomplete attribute. Default "off".
autofocus	Whether the input takes focus on page load. Default FALSE.
name	Native name attribute.
form	Native form attribute.
minlength	Minimum input length.
max	Native max attribute, for number-like types.
min	Native min attribute, for number-like types.
step	Native step attribute, for number-like types.
resize	Resize behaviour of a textarea: "none", "both", "horizontal" or "vertical".
tabindex	Tab index of the input.

`validate_event` Whether a change triggers form validation. Default `TRUE`.

`label_position` Where the label sits, as `el_form()`'s `label_position`: `"top"` (the default, as Shiny's labels sit), or beside the component, its text aligned `"left"` or `"right"` – which shows once `label_width` gives the labels a common width.

`label_width` Width of a label beside the component, as a CSS unit, so that several line up. Element's `label-width`.

`label_suffix` Text after the label, such as `":"`. Element's `label-suffix`.

`required` Draw Element's red asterisk before the label. It marks the field; it does not check it – shinyvalidate or `el_form()` does that.

`error` An error message shown under the component in Element's style, the field framed in red. Element's `error`.

`show_message, inline_message`
Whether `error`'s message is shown, and whether beside the component rather than under it. Element's `show-message` and `inline-message`.

`aria_label` Same as `aria-label` in native input. Element Plus's `aria-label` (string).

`clear_icon` Custom clear icon component. Element Plus's `clear-icon` (string / Component). An icon's name, such as `"Search"`.

`count_graphemes`
Custom function to count graphemes; when set, native `maxlength`/`minlength` constraints are bypassed. Component uses `Intl.Segmenter` (Chrome 87+, Firefox 125+, Safari 14.1+) for proper grapheme clustering; older browsers fall back to `Array.from()` for code-point iteration. Element Plus's `count-graphemes` ((value: string) => number).

`formatter` Specifies the format of the value presented input.(only works when `type` is 'text'). Element Plus's `formatter` ((value: string | number) => string).

`input_style` The style of the input element or textarea element. Element Plus's `input-style` (`string / CSSProperties | CSSProperties[] | string[]`).

`inputmode` Same as `inputmode` in native input. Element Plus's `inputmode` (string).

`parser` Specifies the value extracted from formatter input.(only works when `type` is 'text'). Element Plus's `parser` ((value: string) => string).

`word_limit_position`
Word count position, valid when `show-word-limit` is true. Element Plus's `word-limit-position` ('inside' | 'outside').

`width` Component width, as a CSS unit – `"200px"`, `"50%"`, or a number taken as pixels. Element's own markup carries it, so it behaves like the `width` argument of a Shiny input.

`slots` Named list of Element slot contents, such as `list(title = shiny::tags$b("Bold"))`. A shiny.element component given here is absorbed rather than nested. For a scoped slot, write the template with `template()`.

`events` Element's events to report besides those reported unasked, by name: `events = "node_drop"` reports `input$<id>_node_drop`. The component's are listed under "Shiny inputs", and by `el_events()`; a name it does not have is an error.

on	Handlers of your own, for an event not reported or to send something else: a named list of JS() functions, one per event – Element's, or a DOM event with Vue's modifiers ("keyup.enter"). Each is called with report and the event's arguments; report(name, value) sets input$<id>_<name>. See el_widget().
session	In el_input(), deprecated: inside a module, wrap id in ns(), as for any Shiny input; a session given here namespaces id once more, with a warning. In update_el_input(), the Shiny session, the current one by default, as for shiny::updateTextInput().

Value

An htmltools tagList with a Vue-managed input component.

Element methods

Callable with call_el():

- blur() – Blur the input element
- focus() – Focus the input element
- select() – Select the text in input element

Shiny inputs

Input	Reported	Value
input$<id>	unasked	the text, as it is typed (debounced by 250 ms)
input$<id>_input	events = "input"	triggers when the Input value change
input$<id>_blur	events = "blur"	triggers when Input blurs
input$<id>_focus	events = "focus"	triggers when Input focuses
input$<id>_clear	events = "clear"	triggers when the Input is cleared by clicking the cl
input$<id>_compositionend	events = "compositionend"	triggers when the composition ends
input$<id>_compositionstart	events = "compositionstart"	triggers when the composition starts
input$<id>_compositionupdate	events = "compositionupdate"	triggers when the composition is updated
input$<id>_keydown	events = "keydown"	the key: list(key, code, ctrl, shift, alt, met
input$<id>_mouseenter	events = "mouseenter"	triggers when the mouse enters the Input element
input$<id>_mouseleave	events = "mouseleave"	triggers when the mouse leaves the Input element

The same list as el_events("el_input"), which says how an event's arguments travel.

The text is debounced by 250 ms, as shiny::textInput()'s is, and reported after an update_el_input() too.

Updating from the server

Server-side update for el_input(). Sends a custom message to update named fields on the Vue instance.

Every other argument of el_input() that can change once it is drawn is an argument here too, under the same name. One left NULL stays as it is; NA returns it to Element's default.

update_el_input() is called for its side effect and returns NULL invisibly.

Examples

```
# Basic text input
el_input("name", placeholder = "Enter your name")

# Clearable search input with icon
el_input(
  "search",
  placeholder = "Search...",
  clearable = TRUE,
  prefix_icon = "el-icon-search"
)

# Shiny app example
if (interactive()) {
  library(shiny)
  library(shiny.element)
  ui <- el_page(
    el_input("txt", placeholder = "Type something"),
    verbatimTextOutput("val")
  )
  server <- function(input, output, session) {
    output$val <- renderPrint(input$txt)
  }
  shinyApp(ui, server)
}
if (interactive()) {
  # inside a server function
  observeEvent(input$go, {
    update_el_input(session, "name", value = "Ada")
  })
}
```

---

el_input_number *Element Plus Input Number*

Description

A numeric input with increment/decrement buttons.

Usage

```
el_input_number(
  id = NULL,
  value = 0,
  min = -Inf,
  max = Inf,
  step = 1,
  step_strictly = FALSE,
  precision = NULL,
```

```
  size = NULL,
  disabled = FALSE,
  controls = TRUE,
  controls_position = "",
  placeholder = NULL,
  label = NULL,
  name = NULL,
  label_position = c("top", "left", "right"),
  label_width = NULL,
  label_suffix = NULL,
  required = FALSE,
  error = NULL,
  show_message = TRUE,
  inline_message = FALSE,
  align = NULL,
  aria_label = NULL,
  disabled_scientific = NULL,
  formatter = NULL,
  inputmode = NULL,
  parser = NULL,
  readonly = NULL,
  tabindex = NULL,
  validate_event = NULL,
  value_on_clear = NULL,
  width = NULL,
  slots = NULL,
  events = NULL,
  on = NULL,
  session = NULL
)

update_el_input_number(
  session = shiny::getDefaultReactiveDomain(),
  id,
  value = NULL,
  min = NULL,
  max = NULL,
  disabled = NULL,
  label = NULL,
  error = NULL,
  step = NULL,
  step_strictly = NULL,
  precision = NULL,
  size = NULL,
  controls = NULL,
  controls_position = NULL,
  placeholder = NULL,
  name = NULL,
```

```
  align = NULL,
  aria_label = NULL,
  disabled_scientific = NULL,
  formatter = NULL,
  inputmode = NULL,
  parser = NULL,
  readonly = NULL,
  tabindex = NULL,
  validate_event = NULL,
  value_on_clear = NULL
)
```

Arguments

id	Input ID. Auto-generated UUID if NULL.
value	Initial numeric value. Default 0.
min	Minimum allowed value. Default -Inf.
max	Maximum allowed value. Default Inf.
step	Step increment. Default 1.
step_strictly	Force the value to be a multiple of step. Default FALSE.
precision	Decimal precision (non-negative integer). NULL for auto.
size	Size: "large", "default" or "small"; NULL follows the form or the page.
disabled	Whether the component is disabled. Default FALSE.
controls	Whether to show the +/- control buttons. Default TRUE.
controls_position	Button layout: "" (default, left-right) or "right" (both on the right).
placeholder	Placeholder text. NULL for none.
label	A label shown with the component, as Shiny's inputs have: text or a tag. NULL, the default, shows none. It is the component's accessible name too – tied to it with for where the component has a native input that takes the id <id>-input, else with aria-labelledby.
name	Native name attribute of the inner input.
label_position	Where the label sits, as el_form()'s label_position: "top" (the default, as Shiny's labels sit), or beside the component, its text aligned "left" or "right" – which shows once label_width gives the labels a common width.
label_width	Width of a label beside the component, as a CSS unit, so that several line up. Element's label-width.
label_suffix	Text after the label, such as ":". Element's label-suffix.
required	Draw Element's red asterisk before the label. It marks the field; it does not check it – shinyvalidate or el_form() does that.
error	An error message shown under the component in Element's style, the field framed in red. Element's error.

show_message, inline_message
: Whether error's message is shown, and whether beside the component rather than under it. Element's show-message and inline-message.

align
: Alignment for the inner input text. Element Plus's align ('left' | 'center' | 'right').

aria_label
: Same as aria-label in native input. Element Plus's aria-label (string).

disabled_scientific
: Disables input of scientific notation (e.g. 'e'). Element Plus's disabled-scientific (boolean).

formatter
: Specifies the format of the value presented in the input. Element Plus's formatter ((value: string) => string).

inputmode
: Same as inputmode in native input. Element Plus's inputmode (string).

parser
: Specifies the value extracted from the formatted input. Element Plus's parser ((value: string) => string).

readonly
: Same as readonly in native input. Element Plus's readonly (boolean).

tabindex
: Same as tabindex in native input. Element Plus's tabindex (string / number).

validate_event
: Whether to trigger form validation. Element Plus's validate-event (boolean).

value_on_clear
: Value should be set when input box is cleared. Element Plus's value-on-clear (number / null / 'min' | 'max').

width
: Component width, as a CSS unit – "200px", "50%", or a number taken as pixels. Element's own markup carries it, so it behaves like the width argument of a Shiny input.

slots
: Named list of Element slot contents, such as list(title = shiny::tags$b("Bold")). A shiny.element component given here is absorbed rather than nested. For a scoped slot, write the template with template().

events
: Element's events to report besides those reported unasked, by name: events = "node_drop" reports input$<id>_node_drop. The component's are listed under "Shiny inputs", and by el_events(); a name it does not have is an error.

on
: Handlers of your own, for an event not reported or to send something else: a named list of JS() functions, one per event – Element's, or a DOM event with Vue's modifiers ("keyup.enter"). Each is called with report and the event's arguments; report(name, value) sets input$<id>_<name>. See el_widget().

session
: In el_input_number(), deprecated: inside a module, wrap id in ns(), as for any Shiny input; a session given here namespaces id once more, with a warning. In update_el_input_number(), the Shiny session, the current one by default, as for shiny::updateTextInput().

Value

An htmltools tagList with a Vue-managed input-number component.

Element methods

Callable with call_el():

- focus() – Focus the Input component
- select() – Select the text in input element

Shiny inputs

Input	Reported	Value
`input$<id>`	unasked	the number
`input$<id>_blur`	`events = "blur"`	triggers when Input blurs
`input$<id>_focus`	`events = "focus"`	triggers when Input focuses

The same list as `el_events("el_input_number")`, which says how an event's arguments travel.

Updating from the server

Server-side update for `el_input_number()`.

Every other argument of `el_input_number()` that can change once it is drawn is an argument here too, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_input_number()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_input_number("n1", value = 5, min = 0, max = 100)
el_input_number("n2", value = 1.5, step = 0.5, precision = 1)
if (interactive()) {
  # inside a server function
  observeEvent(input$go, {
    update_el_input_number(session, "age", value = 42)
  })
}
```

---

el_input_otp *Element Plus Input OTP*

Description

A one-time password or verification code, typed one character per field.

Usage

```
el_input_otp(
  id = NULL,
  value = NULL,
  length = NULL,
  validator = NULL,
  inputmode = NULL,
  type = NULL,
  size = NULL,
  mask = NULL,
  disabled = NULL,
```

```
  separator = NULL,
  validate_event = NULL,
  readonly = NULL,
  aria_label = NULL,
  label = NULL,
  label_position = c("top", "left", "right"),
  label_width = NULL,
  label_suffix = NULL,
  required = FALSE,
  error = NULL,
  show_message = TRUE,
  inline_message = FALSE,
  width = NULL,
  slots = NULL,
  events = NULL,
  on = NULL
)

update_el_input_otp(
  session = shiny::getDefaultReactiveDomain(),
  id,
  value = NULL,
  disabled = NULL,
  label = NULL,
  error = NULL,
  length = NULL,
  validator = NULL,
  inputmode = NULL,
  type = NULL,
  size = NULL,
  mask = NULL,
  separator = NULL,
  validate_event = NULL,
  readonly = NULL,
  aria_label = NULL
)
```

Arguments

id	Component ID. Auto-generated if NULL.
value	The value of the OTP fields. Since numbers must not have leading zeros, modelValue is allowed to be a number only during initialization: Element Plus's model-value, reported as input$<id>.
length	The OTP fields length. Element Plus's length (number).
validator	Custom validator function. Element Plus's validator ((char: string, index: number) => boolean). Give it as JS().
inputmode	Native inputmode attribute. Element Plus's inputmode (string).

type	The type of the OTP fields. Element Plus's type ('outlined' | 'filled' | 'underlined').
size	The size of the OTP fields. Element Plus's size ('large' | 'default' | 'small').
mask	Whether to enable password mode. Element Plus's mask (boolean).
disabled	Whether the OTP fields are disabled. Element Plus's disabled (boolean).
separator	The separator between OTP fields. Element Plus's separator (string / VNode / () => string | VNode). Give it as JS().
validate_event	Whether to trigger form validation. Element Plus's validate-event (boolean).
readonly	Same as readonly in native input. Element Plus's readonly (boolean).
aria_label	Native aria-label attribute. Element Plus's aria-label (string).
label	A label shown with the component, as Shiny's inputs have: text or a tag. NULL, the default, shows none. It is the component's accessible name too – tied to it with for where the component has a native input that takes the id <id>-input, else with aria-labelledby.
label_position	Where the label sits, as el_form()'s label_position: "top" (the default, as Shiny's labels sit), or beside the component, its text aligned "left" or "right" – which shows once label_width gives the labels a common width.
label_width	Width of a label beside the component, as a CSS unit, so that several line up. Element's label-width.
label_suffix	Text after the label, such as ":". Element's label-suffix.
required	Draw Element's red asterisk before the label. It marks the field; it does not check it – shinyvalidate or el_form() does that.
error	An error message shown under the component in Element's style, the field framed in red. Element's error.
show_message, inline_message	Whether error's message is shown, and whether beside the component rather than under it. Element's show-message and inline-message.
width	Component width, as a CSS unit.
slots	Named list of Element slot contents: separator. A scoped slot is written with template().
events	Element's events to report besides those reported unasked, by name: events = "node_drop" reports input$<id>_node_drop. The component's are listed under "Shiny inputs", and by el_events(); a name it does not have is an error.
on	Handlers of your own, for an event not reported or to send something else: a named list of JS() functions, one per event – Element's, or a DOM event with Vue's modifiers ("keyup.enter"). Each is called with report and the event's arguments; report(name, value) sets input$<id>_<name>. See el_widget().
session	Shiny session; the current one by default, as for shiny::updateTextInput().

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
input$<id>	unasked	the value
input$<id>_change	events = "change"	Triggers when the value changes after input blur
input$<id>_finish	unasked	Fires when all fields have been filled
input$<id>_focus	events = "focus"	Triggers when input is focused
input$<id>_blur	events = "blur"	Triggers when input is blurred

The same list as `el_events("el_input_otp")`, which says how an event's arguments travel.

Element methods

Callable with `call_el()`: `focus()`, `blur()`.

Updating from the server

Server-side update for `el_input_otp()`.

Every other argument of `el_input_otp()` that can change once it is drawn is an argument here too, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_input_otp()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_input_otp("code", length = 6)
el_input_otp("pin", length = 4, mask = TRUE, type = "underlined")
if (interactive()) {
  # inside a server function
  observeEvent(input$reset, update_el_input_otp(session, "x", value = NULL))
}
```

---

el_input_tag *Element Plus Input Tag*

Description

A text input that turns each entry into a tag: keywords, e-mail addresses, labels.

Usage

```
el_input_tag(
  id = NULL,
  value = list(),
  max = NULL,
  tag_type = NULL,
  tag_effect = NULL,
```

```
  effect = NULL,
  trigger = NULL,
  draggable = NULL,
  delimiter = NULL,
  size = NULL,
  collapse_tags = NULL,
  collapse_tags_tooltip = NULL,
  save_on_blur = NULL,
  clearable = NULL,
  clear_icon = NULL,
  disabled = NULL,
  validate_event = NULL,
  readonly = NULL,
  autofocus = NULL,
  tabindex = NULL,
  max_collapse_tags = NULL,
  maxlength = NULL,
  minlength = NULL,
  placeholder = NULL,
  autocomplete = NULL,
  aria_label = NULL,
  label = NULL,
  label_position = c("top", "left", "right"),
  label_width = NULL,
  label_suffix = NULL,
  required = FALSE,
  error = NULL,
  show_message = TRUE,
  inline_message = FALSE,
  width = NULL,
  slots = NULL,
  events = NULL,
  on = NULL
)

update_el_input_tag(
  session = shiny::getDefaultReactiveDomain(),
  id,
  value = NULL,
  disabled = NULL,
  label = NULL,
  error = NULL,
  max = NULL,
  tag_type = NULL,
  tag_effect = NULL,
  effect = NULL,
  trigger = NULL,
  draggable = NULL,
```

```
  delimiter = NULL,
  size = NULL,
  collapse_tags = NULL,
  collapse_tags_tooltip = NULL,
  save_on_blur = NULL,
  clearable = NULL,
  clear_icon = NULL,
  validate_event = NULL,
  readonly = NULL,
  autofocus = NULL,
  tabindex = NULL,
  max_collapse_tags = NULL,
  maxlength = NULL,
  minlength = NULL,
  placeholder = NULL,
  autocomplete = NULL,
  aria_label = NULL
)
```

Arguments

id	Component ID. Auto-generated if NULL.
value	Binding value: Element Plus's model-value, reported as input$<id>.
max	Max number tags that can be enter. Element Plus's max (number).
tag_type	Tag type. Element Plus's tag-type ('' | 'success' | 'info' | 'warning' | 'danger').
tag_effect	Tag effect. Element Plus's tag-effect ('' | 'light' | 'dark' | 'plain').
effect	Tooltip theme, built-in theme: dark / light. Element Plus's effect ('dark' | 'light' / string).
trigger	The key to trigger input tag. Element Plus's trigger ('Enter' | 'Space').
draggable	Whether tags can be dragged. Element Plus's draggable (boolean).
delimiter	Add a tag when a delimiter is matched. Element Plus's delimiter (string / regex).
size	Input box size. Element Plus's size ('large' | 'default' | 'small').
collapse_tags	Whether to collapse tags to a text when multiple selecting. Element Plus's collapse-tags (boolean).
collapse_tags_tooltip	Whether show all selected tags when mouse hover text of collapse-tags. To use this, collapse-tags must be true. Element Plus's collapse-tags-tooltip (boolean).
save_on_blur	Whether to save the input value when the input loses focus. Element Plus's save-on-blur (boolean).
clearable	Whether to show clear button. Element Plus's clearable (boolean).
clear_icon	Custom clear icon component. Element Plus's clear-icon (string / Component). An icon's name, such as "Search".

`disabled` Whether to disable input-tag. Element Plus's `disabled` (boolean).

`validate_event` Whether to trigger form validation. Element Plus's `validate-event` (boolean).

`readonly` Same as `readonly` in native input. Element Plus's `readonly` (boolean).

`autofocus` Same as `autofocus` in native input. Element Plus's `autofocus` (boolean).

`tabindex` Same as `tabindex` in native input. Element Plus's `tabindex` (string / number).

`max_collapse_tags`
The max tags number to be shown. To use this, collapse-tags must be true. Element Plus's `max-collapse-tags` (number).

`maxlength` Same as `maxlength` in native input. Element Plus's `maxlength` (string / number).

`minlength` Same as `minlength` in native input. Element Plus's `minlength` (string / number).

`placeholder` Placeholder of input. Element Plus's `placeholder` (string).

`autocomplete` Same as `autocomplete` in native input. Element Plus's `autocomplete` (string).

`aria_label` Native `aria-label` attribute. Element Plus's `aria-label` (string).

`label` A label shown with the component, as Shiny's inputs have: text or a tag. `NULL`, the default, shows none. It is the component's accessible name too – tied to it with `for` where the component has a native input that takes the id `<id>-input`, else with `aria-labelledby`.

`label_position` Where the label sits, as `el_form()`'s `label_position`: `"top"` (the default, as Shiny's labels sit), or beside the component, its text aligned `"left"` or `"right"` – which shows once `label_width` gives the labels a common width.

`label_width` Width of a label beside the component, as a CSS unit, so that several line up. Element's `label-width`.

`label_suffix` Text after the label, such as `":"`. Element's `label-suffix`.

`required` Draw Element's red asterisk before the label. It marks the field; it does not check it – shinyvalidate or `el_form()` does that.

`error` An error message shown under the component in Element's style, the field framed in red. Element's `error`.

`show_message, inline_message`
Whether `error`'s message is shown, and whether beside the component rather than under it. Element's `show-message` and `inline-message`.

`width` Component width, as a CSS unit.

`slots` Named list of Element slot contents: `tag`, `prefix`, `suffix`. A scoped slot is written with `template()`.

`events` Element's events to report besides those reported unasked, by name: `events = "node_drop"` reports `input$<id>_node_drop`. The component's are listed under "Shiny inputs", and by `el_events()`; a name it does not have is an error.

`on` Handlers of your own, for an event not reported or to send something else: a named list of `JS()` functions, one per event – Element's, or a DOM event with Vue's modifiers (`"keyup.enter"`). Each is called with `report` and the event's arguments; `report(name, value)` sets `input$<id>_<name>`. See `el_widget()`.

`session` Shiny session; the current one by default, as for `shiny::updateTextInput()`.

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
`input$<id>`	unasked	the tags
`input$<id>_input`	`events = "input"`	triggers when the input value change
`input$<id>_add_tag`	`events = "add_tag"`	triggers when a tag is added
`input$<id>_remove_tag`	`events = "remove_tag"`	triggers when a tag is removed
`input$<id>_drag_tag`	`events = "drag_tag"`	triggers when a tag is dragged
`input$<id>_focus`	`events = "focus"`	triggers when InputTag focuses
`input$<id>_blur`	`events = "blur"`	triggers when InputTag blurs
`input$<id>_clear`	`events = "clear"`	triggers when the clear icon is clicked

The same list as `el_events("el_input_tag")`, which says how an event's arguments travel.

Element methods

Callable with `call_el()`: `focus()`, `blur()`.

Updating from the server

Server-side update for `el_input_tag()`.

Every other argument of `el_input_tag()` that can change once it is drawn is an argument here too, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_input_tag()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_input_tag(
  "keywords",
  value = c("shiny", "element"),
  placeholder = "Add a keyword"
)
if (interactive()) {
  # inside a server function
  observeEvent(input$reset, update_el_input_tag(session, "x", value = NULL))
}
```

el_layout_css_dependency *Element Plus Layout CSS Dependency*

Description

Provides default CSS styles for Element-UI layout and grid components, including el-container, el-header, el-main, el-footer, el-aside, el-row, el-col, etc. This dependency is automatically attached by el_page() for consistent layout appearance.

Usage

```
el_layout_css_dependency()
```

Value

An htmlDependency object.

Examples

```
el_layout_css_dependency()
```

el_link *Element Plus Link*

Description

A styled hyperlink that follows Element Plus's design language.

Usage

```
el_link(
  label = "Link",
  href = NULL,
  type = NULL,
  underline = NULL,
  disabled = FALSE,
  icon = NULL,
  id = NULL,
  ...,
  target = "_self",
  on = NULL
)

update_el_link(
  session = shiny::getDefaultReactiveDomain(),
```

```
  id,
  label = NULL,
  href = NULL,
  type = NULL,
  underline = NULL,
  disabled = NULL,
  icon = NULL
)
```

Arguments

label	Link text. Accepts a string or HTML tag.
href	URL target. NULL for a non-navigating link.
type	Link colour type: "default", "primary", "success", "warning", "danger", "info".
underline	When the link is underlined: "hover", "always" or "never". TRUE and FALSE, Element UI's form, are "hover" and "never". type and underline left NULL are "default" and "hover", or for a link with an id a config provider's el_config_provider(link =).
disabled	Whether the link is disabled. Default FALSE.
icon	Icon class string (e.g. "el-icon-edit"). Placed before the label. NULL for none.
id	Give the link an id and it reports its clicks, as shiny::actionLink() does: input$<id> counts them, 0 on load, and update_el_link() changes it. Without one it is a plain link.
...	Additional HTML attributes passed to the <a> tag (a plain link only).
target	Where the link opens, as an <a>'s target. Default "_self".
on	Handlers of your own, for an event not reported or to send something else: a named list of JS() functions, one per event – Element's, or a DOM event with Vue's modifiers ("keyup.enter"). Each is called with report and the event's arguments; report(name, value) sets input$<id>_<name>. See el_widget().
session	Shiny session; the current one by default, as for shiny::updateActionLink().

Value

An htmltools <a> tag, or with an id a Shiny UI element.

Shiny inputs

Input	Reported	Value
input$<id>	unasked	with an id, the number of clicks, as actionLink() reports it

The same list as el_events("el_link"), which says how an event's arguments travel.

Updating from the server

Server-side update for an `el_link()` given an `id`.

`update_el_link()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_link("Visit GitHub", href = "https://github.com", type = "primary")
el_link("Disabled", disabled = TRUE)
el_link("With icon", icon = "el-icon-edit", type = "success")

# An action link: input$more counts its clicks
el_link("Show more", id = "more", type = "primary")
if (interactive()) {
  # inside a server function
  observeEvent(input$more, update_el_link(session, "more", label = "Show less"))
}
```

---

el_loading *Element Plus Loading Mask*

Description

Cover the page, or one element, while something is being worked out. Each mask is named, and stays until `el_loading_close()` is called with the same `id`.

Usage

```
el_loading(
  session = shiny::getDefaultReactiveDomain(),
  id = "default",
  text = NULL,
  target = NULL,
  fullscreen = NULL,
  lock = NULL,
  body = NULL,
  spinner = NULL,
  background = NULL,
  custom_class = NULL,
  svg = NULL,
  svg_view_box = NULL,
  before_close = NULL
)
```

Arguments

`session`	Shiny session; the current one by default, as for `shiny::updateTextInput()`.
`id`	Name for this mask, used to close it again.

`text`	Text shown under the spinner.
`target`	CSS selector of the element to cover. `NULL` covers the page.
`fullscreen`	Whether the mask covers the viewport. Default `TRUE` when no `target` is given.
`lock`	Whether to stop the page scrolling underneath.
`body`	Whether the mask is inserted into body rather than the target.
`spinner`	Class name of a custom spinner.
`background`	Background colour of the mask.
`custom_class`	Extra class name.
`svg, svg_view_box`	A spinner of your own, as SVG path markup and its `viewBox`.
`before_close`	`JS()` function returning whether the mask may close.

Value

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

Shiny inputs

- `input$<id>_closed` – fires once the mask has closed.

Examples

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

  ui <- el_page(el_button("go", "Fetch"), el_table("out"))

  server <- function(input, output, session) {
    observeEvent(input$go, {
      el_loading(session, "fetching", text = "Fetching rows...")
      on.exit(el_loading_close(session, "fetching"), add = TRUE)
      update_el_table(session, "out", data = slow_query())
    })
  }
  shinyApp(ui, server)
}
```

---

el_loading_close *Close a loading mask*

Description

Closes the mask `el_loading()` opened under this `id`. Closing one that is not open does nothing.

Usage

```
el_loading_close(session = shiny::getDefaultReactiveDomain(), id = "default")
```

Arguments

session	Shiny session; the current one by default, as for shiny::updateTextInput().
id	The id the mask was opened with.

Value

Called for its side effect; returns NULL invisibly.

Examples

```
if (interactive()) {
  # inside a server function
  observeEvent(input$done, {
    el_loading_close(session, "fetching")
  })
}
```

---

el_load_children *Answer a component that asked the server for data*

Description

Element loads some components a piece at a time: a lazy tree its nodes' children, a lazy cascader its next column, a lazy tree table its rows' children. Upstream, a JavaScript function you write fetches them. Here the server does: the component asks through an input – input$<id>_load for a tree or table, input$<id>_lazy_load for a cascader – and waits until this function answers.

Usage

```
el_load_children(
  session = shiny::getDefaultReactiveDomain(),
  id,
  request,
  children = list(),
  reject = FALSE
)
```

Arguments

session	Shiny session; the current one by default, as for shiny::updateTextInput().
id	The component's ID (un-namespaced).
request	The question, as it arrived in input$<id>_load or input$<id>_lazy_load – or just its request number.

children	What to load. For a tree, nodes such as `list(id =, label =, leaf = TRUE)`; for a cascader, options such as `list(value =, label =, leaf = TRUE)`; for a table, rows, as a data.frame or a list. An empty list means there is nothing below.
reject	`TRUE` to answer that the load failed, Element Plus's `reject()`: a tree node stops spinning and can be loaded again when it is next expanded. A cascader or table takes it as nothing below.

Details

Each question carries a `request` number, so answers find their way back even when several are open at once; pass the question back whole.

Value

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

Examples

```
if (interactive()) {
  library(shiny)
  ui <- el_page(el_tree(
    "files",
    lazy = TRUE,
    node_key = "id",
    props = list(isLeaf = "leaf")
  ))
  server <- function(input, output, session) {
    observeEvent(input$files_load, {
      q <- input$files_load
      dir <- if (q$level == 0) "~" else q$key
      entries <- list.files(dir, full.names = TRUE)
      el_load_children(
        id = "files",
        request = q,
        children = lapply(entries, function(f) {
          list(id = f, label = basename(f), leaf = !dir.exists(f))
        })
      )
    })
  }
  shinyApp(ui, server)
}
```

---

el_locales *Collapse Handler Dependency Languages Element Plus can use for its built-in text*

Description

Collapse Handler Dependency Languages Element Plus can use for its built-in text

Usage

```
el_locales()
```

Value

A character vector of locale codes, such as `"en"` and `"zh-TW"`, any of which can be passed as `locale` to `el_page()`.

Examples

```
el_locales()
```

`el_locale_dependency` *Element Plus Locale Dependency*

Description

Element Plus's own build is in English: a pagination control's total, a date picker's buttons, a select's placeholder, a table's empty message. Loading a locale file and handing it to Element Plus, as `app.use(ElementPlus, {locale})` does, switches all of it.

Usage

```
el_locale_dependency(locale = NULL)
```

Arguments

`locale`	Language to switch to, such as `"fr"`, `"zh-cn"` or `"pt-br"`. `NULL` or `"en"` leaves Element Plus's built-in English.

Details

All 67 of Element Plus's locales are bundled; `el_locales()` lists them. The ones used most – English, Chinese, Japanese, Korean, French, German, Spanish, Portuguese (Brazil), Russian – are files; the others are packed in one archive and taken out the first time one is used in a session. Codes are matched without regard to case, so Element UI's `"zh-CN"` is Element Plus's `"zh-cn"`.

Value

A list of htmlDependency objects, or `NULL` for the built-in locale.

Examples

```
el_locales()

# English is Element Plus's own
el_page(el_select("city", choices = c("Beijing", "Shanghai")))

# Anything Element Plus ships
el_page(locale = "ja", el_select("city", choices = c("Beijing", "Shanghai")))

# Or set it for the whole session, here undone afterwards
old <- options(shiny.element.locale = "zh-cn")
el_page(el_select("city", choices = c("Beijing", "Shanghai")))
options(old)
```

---

el_main *Element Plus Main*

Description

Element Plus Main

Usage

```
el_main(..., style = NULL, class = NULL)
```

Arguments

`...`	Content.
`style`	Extra inline style.
`class`	Extra CSS classes.

Value

A Shiny UI element.

Examples

```
el_main("Body content")
el_main(el_table(data = head(iris, 3)))
```

el_mention *Element Plus Mention*

Description

A text input that offers options after a trigger character, as @mentions do.

Usage

```
el_mention(
  id = NULL,
  value = NULL,
  options = NULL,
  props = NULL,
  prefix = NULL,
  split = NULL,
  filter_option = NULL,
  placement = NULL,
  show_arrow = NULL,
  offset = NULL,
  whole = NULL,
  check_is_whole = NULL,
  loading = NULL,
  popper_class = NULL,
  popper_style = NULL,
  popper_options = NULL,
  placeholder = NULL,
  disabled = NULL,
  type = NULL,
  rows = NULL,
  label = NULL,
  label_position = c("top", "left", "right"),
  label_width = NULL,
  label_suffix = NULL,
  required = FALSE,
  error = NULL,
  show_message = TRUE,
  inline_message = FALSE,
  width = NULL,
  slots = NULL,
  events = NULL,
  on = NULL
)

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

```
  value = NULL,
  disabled = NULL,
  label = NULL,
  error = NULL,
  options = NULL,
  props = NULL,
  prefix = NULL,
  split = NULL,
  filter_option = NULL,
  placement = NULL,
  show_arrow = NULL,
  offset = NULL,
  whole = NULL,
  check_is_whole = NULL,
  loading = NULL,
  popper_class = NULL,
  popper_style = NULL,
  popper_options = NULL,
  placeholder = NULL,
  type = NULL,
  rows = NULL
)
```

Arguments

id	Component ID. Auto-generated if NULL.
value	Input value: Element Plus's model-value, reported as input$<id>.
options	The options: a named vector c(Label = value), a vector, or a list of list(value =, label =, disabled =), as Element Plus takes them.
props	Configuration options. Element Plus's props (MentionOptionProps).
prefix	Prefix character to trigger mentions. The string length must be exactly 1. Element Plus's prefix (string / string[]).
split	Character to split mentions. The string length must be exactly 1. Element Plus's split (string).
filter_option	Customize filter option logic. Element Plus's filter-option (false / (pattern: string, option: MentionOption) => boolean). Give it as JS().
placement	Set popup placement. Element Plus's placement ('bottom' | 'top').
show_arrow	Whether the dropdown panel has an arrow. Element Plus's show-arrow (boolean).
offset	Offset of the dropdown panel. Element Plus's offset (number).
whole	When backspace is pressed to delete, whether the mention content is deleted as a whole. Element Plus's whole (boolean).
check_is_whole	When backspace is pressed to delete, check if the mention is a whole. Element Plus's check-is-whole ((pattern: string, prefix: string) => boolean). Give it as JS().

`loading` Whether the dropdown panel of mentions is in a loading state. Element Plus's `loading` (boolean).

`popper_class` Custom class name for dropdown panel. Element Plus's `popper-class` (string / object).

`popper_style` Custom style for dropdown panel. Element Plus's `popper-style` (string / object).

`popper_options` Popper.js parameters. Element Plus's `popper-options` (object).

`placeholder`, `disabled`, `type`, `rows`
As for `el_input()`: the placeholder text, whether it can be changed, and `type = "textarea"` with its number of rows for a longer message.

`label` A label shown with the component, as Shiny's inputs have: text or a tag. `NULL`, the default, shows none. It is the component's accessible name too – tied to it with `for` where the component has a native input that takes the id `<id>-input`, else with `aria-labelledby`.

`label_position` Where the label sits, as `el_form()`'s `label_position`: `"top"` (the default, as Shiny's labels sit), or beside the component, its text aligned `"left"` or `"right"` – which shows once `label_width` gives the labels a common width.

`label_width` Width of a label beside the component, as a CSS unit, so that several line up. Element's `label-width`.

`label_suffix` Text after the label, such as `":"`. Element's `label-suffix`.

`required` Draw Element's red asterisk before the label. It marks the field; it does not check it – shinyvalidate or `el_form()` does that.

`error` An error message shown under the component in Element's style, the field framed in red. Element's `error`.

`show_message`, `inline_message`
Whether `error`'s message is shown, and whether beside the component rather than under it. Element's `show-message` and `inline-message`.

`width` Component width, as a CSS unit.

`slots` Named list of Element slot contents: `label`, `loading`, `header`, `footer`. A scoped slot is written with `template()`.

`events` Element's events to report besides those reported unasked, by name: `events = "node_drop"` reports `input$<id>_node_drop`. The component's are listed under "Shiny inputs", and by `el_events()`; a name it does not have is an error.

`on` Handlers of your own, for an event not reported or to send something else: a named list of `JS()` functions, one per event – Element's, or a DOM event with Vue's modifiers (`"keyup.enter"`). Each is called with `report` and the event's arguments; `report(name, value)` sets `input$<id>_<name>`. See `el_widget()`.

`session` Shiny session; the current one by default, as for `shiny::updateTextInput()`.

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
input$<id>	unasked	the text
input$<id>_search	unasked	list(pattern, prefix); answer with update_el_mention(op
input$<id>_select	unasked	list(option, prefix)
input$<id>_whole_remove	events = "whole_remove"	list(pattern, prefix)

The same list as `el_events("el_mention")`, which says how an event's arguments travel.

Updating from the server

Server-side update for `el_mention()`.

Every other argument of `el_mention()` that can change once it is drawn is an argument here too, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_mention()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_mention("msg", options = c("Ada", "Grace", "Linus"), placeholder = "Type @")
if (interactive()) {
  # inside a server function
  observeEvent(input$reset, update_el_mention(session, "x", value = NULL))
}
```

---

el_menu *Element Plus Menu*

Description

A navigation menu, vertical or horizontal, with submenus nested to any depth.

Usage

```
el_menu(
  id = NULL,
  items = list(),
  default_active = NULL,
  mode = "vertical",
  collapse = FALSE,
  unique_opened = FALSE,
  background_color = NULL,
  text_color = NULL,
  active_text_color = NULL,
  default_openeds = NULL,
  menu_trigger = NULL,
```

```
  collapse_transition = NULL,
  router = NULL,
  close_on_click_outside = NULL,
  ellipsis = NULL,
  ellipsis_icon = NULL,
  hide_timeout = NULL,
  persistent = NULL,
  popper_class = NULL,
  popper_effect = NULL,
  popper_offset = NULL,
  popper_style = NULL,
  show_timeout = NULL,
  class = NULL,
  style = NULL,
  width = NULL,
  slots = NULL,
  events = NULL,
  on = NULL,
  session = NULL
)

update_el_menu(
  session = shiny::getDefaultReactiveDomain(),
  id,
  default_active = NULL,
  collapse = NULL,
  mode = NULL,
  unique_opened = NULL,
  background_color = NULL,
  text_color = NULL,
  active_text_color = NULL,
  menu_trigger = NULL,
  collapse_transition = NULL,
  router = NULL,
  close_on_click_outside = NULL,
  ellipsis = NULL,
  ellipsis_icon = NULL,
  hide_timeout = NULL,
  persistent = NULL,
  popper_class = NULL,
  popper_effect = NULL,
  popper_offset = NULL,
  popper_style = NULL,
  show_timeout = NULL
)
```

Arguments

id	Menu ID (auto-generated if NULL).

items — A list of items, each an el_menu_item(), el_sub_menu() or el_menu_item_group() – or a list with index (the value reported when selected), label (or title, Element's name for it), and optionally icon (an icon's name, such as "House"), disabled, route (for router = TRUE), or children for a submenu. A submenu may also carry Element Plus's sub-menu props: popper_class, popper_style, show_timeout, hide_timeout, teleported, popper_offset, and its expand and collapse icons (expand_close_icon, expand_open_icon, collapse_close_icon, collapse_open_icon). An item with group = TRUE becomes a titled group of its children rather than a submenu.

Clicking an item reports input$<id> (the index selected) and input$<id>_item_click (the index clicked).

default_active — Index of the initially selected item: Element Plus's default-active. In update_el_menu(), the item to select.

mode — "vertical" (default) or "horizontal".

collapse — Collapse to icons only. Vertical menus only.

unique_opened — Keep only one submenu open at a time.

background_color, text_color, active_text_color — Menu colours.

default_openeds — Character vector of sub-menu indexes open at start.

menu_trigger — How a horizontal sub-menu opens: "hover" (default) or "click".

collapse_transition — Whether to animate collapsing. Default TRUE.

router — Whether to use vue-router mode, taking each index as a path.

close_on_click_outside — Optional, whether menu is collapsed when clicking outside. Element Plus's close-on-click-outside (boolean).

ellipsis — Whether the menu is ellipsis (available only in horizontal mode). Element Plus's ellipsis (boolean).

ellipsis_icon — Custom ellipsis icon (available only in horizontal mode and ellipsis is true). Element Plus's ellipsis-icon (string / Component). An icon's name, such as "Search".

hide_timeout — Control timeout for all menus before hiding. Element Plus's hide-timeout (number).

persistent — When menu inactive and persistent is false , dropdown menu will be destroyed. Element Plus's persistent (boolean).

popper_class — Custom class name for all popup menus and titles' tooltips. Element Plus's popper-class (string).

popper_effect — Tooltip theme, built-in theme: dark / light when menu is collapsed. Element Plus's popper-effect ('dark' | 'light' / string).

popper_offset — Offset of the popper (effective for all submenus). Element Plus's popper-offset (number).

popper_style — Custom style for all popup menus and titles' tooltips. Element Plus's popper-style (string / object).

show_timeout	Control timeout for all menus before showing. Element Plus's show-timeout (number).
class, style	Extra classes and inline style on the menu, as Element passes them to its root: Element's examples style theirs by a class of their own.
width	Component width, as a CSS unit – "200px", "50%", or a number taken as pixels. Element's own markup carries it, so it behaves like the width argument of a Shiny input.
slots	Named list of Element slot contents, such as list(title = shiny::tags$b("Bold")). A shiny.element component given here is absorbed rather than nested. For a scoped slot, write the template with template().
events	Element's events to report besides those reported unasked, by name: events = "node_drop" reports input$<id>_node_drop. The component's are listed under "Shiny inputs", and by el_events(); a name it does not have is an error.
on	Handlers of your own, for an event not reported or to send something else: a named list of JS() functions, one per event – Element's, or a DOM event with Vue's modifiers ("keyup.enter"). Each is called with report and the event's arguments; report(name, value) sets input$<id>_<name>. See el_widget().
session	In el_menu(), deprecated: inside a module, wrap id in ns(), as for any Shiny input; a session given here namespaces id once more, with a warning. In update_el_menu(), the Shiny session, the current one by default, as for shiny::updateTextInput().

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
input$<id>	unasked	the index of the selected item
input$<id>_path	unasked	the indexes down to it
input$<id>_open	events = "open"	list(index, path) of the submenu
input$<id>_close	events = "close"	list(index, path) of the submenu

The same list as el_events("el_menu"), which says how an event's arguments travel.

Element methods

Callable with call_el():

- close() – Close a specific sub-menu
- open() – Open a specific sub-menu

Updating from the server

`update_el_menu()` changes the component from the server.

Every other argument of `el_menu()` that can change once it is drawn is an argument here too, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_menu()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_menu(
  id = "nav",
  default_active = "home",
  items = list(
    list(index = "home", label = "Home", icon = "House"),
    list(
      index = "products",
      label = "Products",
      icon = "Goods",
      children = list(
        list(index = "products-all", label = "All"),
        list(index = "products-new", label = "New")
      )
    ),
    list(index = "help", label = "Help", disabled = TRUE)
  )
)

# Horizontal, as a top bar
el_menu(
  id = "topnav",
  mode = "horizontal",
  default_active = "a",
  items = list(
    list(index = "a", label = "One"),
    list(index = "b", label = "Two")
  )
)
if (interactive()) {
  # inside a server function
  observeEvent(input$go, {
    update_el_menu(session, "nav", default_active = "data")
  })
}
```

---

el_menu_item *Items of* `el_menu()`*: an entry, a submenu, a group*

Description

Element Plus's `el-menu-item`, `el-sub-menu` and `el-menu-item-group`, for `el_menu(items =)`. A submenu and a group hold items of their own in `...`, nested to any depth.

Usage

```
el_menu_item(label, index, icon = NULL, route = NULL, disabled = NULL)

el_sub_menu(
  label,
  index,
  ...,
  icon = NULL,
  disabled = NULL,
  popper_class = NULL,
  popper_style = NULL,
  show_timeout = NULL,
  hide_timeout = NULL,
  teleported = NULL,
  popper_offset = NULL,
  expand_close_icon = NULL,
  expand_open_icon = NULL,
  collapse_close_icon = NULL,
  collapse_open_icon = NULL
)

el_menu_item_group(title, ...)
```

Arguments

`label`	The text shown.
`index`	The item's value, `input$<id>` when it is selected.
`icon`	An icon, by name: `"House"`.
`route`	Where the item goes, with `el_menu(router = TRUE)`.
`disabled`	Whether the item or submenu is disabled.
`...`	The submenu's or the group's items.
`popper_class`, `popper_style`	Class and style of the submenu's popup.
`show_timeout`, `hide_timeout`	Delay before the submenu opens and closes, in milliseconds; the menu's by default.
`teleported`	Whether the popup is appended to `<body>`: by default for a top-level submenu, not for others.
`popper_offset`	Offset of the popup, over the menu's.
`expand_close_icon`, `expand_open_icon`	Icons of the submenu when the menu is expanded, closed and open; give both.
`collapse_close_icon`, `collapse_open_icon`	The same when the menu is collapsed.
`title`	The group's title.

Value

An item, for `el_menu(items =)`.

See Also

Other items: `el_anchor_link()`, `el_breadcrumb_item()`, `el_carousel_item()`, `el_collapse_item()`, `el_descriptions_item()`, `el_dropdown_item()`, `el_option()`, `el_skeleton_item()`, `el_step()`, `el_tab_pane()`, `el_table_column()`, `el_table_v2_column()`, `el_timeline_item()`, `el_tour_step()`

Examples

```
el_menu(
  "m",
  items = list(
    el_menu_item("Processing Center", "1"),
    el_sub_menu(
      "Workspace",
      "2",
      el_menu_item("item one", "2-1"),
      el_menu_item_group("Group", el_menu_item("item two", "2-2"))
    ),
    el_menu_item("Info", "3", disabled = TRUE)
  ),
  mode = "horizontal"
)
```

---

el_message *Show Element Plus Message*

Description

Server-side function to show a top-centre message toast. Requires `use_element()` or `el_page()` in the UI to load the JS handler.

Usage

```
el_message(
  session = shiny::getDefaultReactiveDomain(),
  message,
  type = "info",
  duration = 3000,
  show_close = FALSE,
  offset = 16,
  custom_class = NULL,
  dangerously_use_html_string = FALSE,
  id = NULL,
  icon = NULL,
  plain = NULL,
```

```
  placement = NULL,
  append_to = NULL,
  grouping = NULL,
  repeat_num = NULL
)
```

Arguments

session	Shiny session; the current one by default, as for shiny::updateTextInput().
message	Message text, or a VNode built in the browser: JS() code calling Vue.h().
type	Message type: "info", "success", "warning", or "error". Default "info".
duration	Auto-close delay in milliseconds. 0 disables auto-close. Default 3000.
show_close	Whether to show the close button. Default FALSE.
offset	Distance from the edge of the window, in pixels. Default 16.
custom_class	Extra class name.
dangerously_use_html_string	Whether message is inserted as HTML. Only pass TRUE for markup you control – it is not escaped.
id	Name for this message, so el_message_close() can close it and input$<id>_close reports when it closes.
icon	Icon to show instead of the one type implies, by name.
plain	Whether it is drawn plain, without a background colour.
placement	Where it appears: "top" (the default), "top-left", "top-right", "bottom", "bottom-left" or "bottom-right".
append_to	CSS selector of the element it is appended to. Default <body>.
grouping	Whether identical messages shown together are merged into one, with a count.
repeat_num	The count a grouped message starts from.

Value

A Shiny UI element.

Examples

```
if (interactive()) {
  library(shiny)
  library(shiny.element)
  ui <- el_page(
    el_button("btn", "Message", type = "primary")
  )
  server <- function(input, output, session) {
    observeEvent(input$btn, {
      el_message(
        session,
        message = "This is a message toast.",
        type = "warning"
```

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

---

el_message_box *Element Plus Message Box*

Description

A modal that asks something and waits for an answer: a confirmation, an acknowledgement, or a line of text.

Usage

```
el_message_box(
  session = shiny::getDefaultReactiveDomain(),
  id,
  message,
  title = NULL,
  type = NULL,
  box_type = c("confirm", "alert", "prompt"),
  confirm_button_text = NULL,
  cancel_button_text = NULL,
  show_cancel_button = NULL,
  show_close = NULL,
  center = FALSE,
  round_button = FALSE,
  dangerously_use_html_string = FALSE,
  custom_class = NULL,
  close_on_click_modal = NULL,
  close_on_press_escape = NULL,
  input_placeholder = NULL,
  input_value = NULL,
  input_pattern = NULL,
  input_error_message = NULL,
  input_type = NULL,
  input_validator = NULL,
  show_input = NULL,
  show_confirm_button = NULL,
  confirm_button_class = NULL,
  cancel_button_class = NULL,
  distinguish_cancel_and_close = NULL,
  lock_scroll = NULL,
  close_on_hash_change = NULL,
  before_close = NULL,
```

```
  custom_style = NULL,
  icon = NULL,
  close_icon = NULL,
  confirm_button_type = NULL,
  cancel_button_type = NULL,
  confirm_button_loading_icon = NULL,
  cancel_button_loading_icon = NULL,
  autofocus = NULL,
  modal = NULL,
  modal_class = NULL,
  draggable = NULL,
  overflow = NULL,
  button_size = NULL,
  append_to = NULL
)
```

Arguments

session	Shiny session; the current one by default, as for shiny::updateTextInput().
id	Input ID the answer is reported to.
message	The question or statement, or a VNode built in the browser: JS() code calling Vue.h(), or a function of the box's actions (confirm, cancel, close) returning one.
title	Title of the box.
type	Icon shown: "success", "info", "warning" or "error".
box_type	"confirm" (default) offers two buttons, "alert" one, and "prompt" asks for text.
confirm_button_text, cancel_button_text	Button labels.
show_cancel_button	Whether to offer a cancel button. Default TRUE for "confirm" and "prompt".
show_close	Whether to show the close cross. Default TRUE.
center	Whether to centre the content.
round_button	Whether the buttons are rounded.
dangerously_use_html_string	Whether message is rendered as HTML. Only pass TRUE for markup you control – it is inserted unescaped.
custom_class	Extra class name.
close_on_click_modal	Whether clicking the backdrop closes it.
close_on_press_escape	Whether Escape closes it.
input_placeholder, input_value	For box_type = "prompt": the placeholder and the initial text.
input_pattern	Regular expression the text must match, as a string.

`input_error_message`
: Message shown when it does not match.

`input_type`
: For a prompt, the input's type, such as `"password"`.

`input_validator`
: For a prompt, `JS()` function checking the text; return `true`, or an error message.

`show_input`
: Whether to show a text input, as a prompt does.

`show_confirm_button`
: Whether to show the confirm button.

`confirm_button_class`, `cancel_button_class`
: Extra class names for the buttons.

`distinguish_cancel_and_close`
: Whether closing by the cross or Escape reports `"close"` rather than `"cancel"`.

`lock_scroll`
: Whether the page stops scrolling while the box is open.

`close_on_hash_change`
: Whether a change of the URL hash closes it.

`before_close`
: `JS()` function `function(action, instance, done)`, called before the box closes; call `done()` to let it.

`custom_style`
: CSS for the box, a string or a named list.

`icon`, `close_icon`
: The icon shown instead of the one `type` implies, and the close button's, by name.

`confirm_button_type`, `cancel_button_type`
: The buttons' types, as `el_button()`'s `type`.

`confirm_button_loading_icon`, `cancel_button_loading_icon`
: The buttons' loading icons, by name.

`autofocus`
: Whether the box takes focus when it opens. Default `TRUE`.

`modal`
: Whether it has a backdrop. Default `TRUE`.

`modal_class`
: Extra class name for the backdrop.

`draggable`
: Whether it can be dragged by its header.

`overflow`
: With `draggable`, whether it can be dragged past the viewport.

`button_size`
: The buttons' size: `"small"`, `"default"` or `"large"`.

`append_to`
: CSS selector of the element it is appended to. Default `<body>`.

Details

Unlike `el_message()` and `el_notification()`, this one answers back. The reply arrives as `input$<id>`, so read it with `observeEvent()`: the input is set with event priority, so answering "confirm" twice in a row fires the observer twice, where an output reading the value would see no change.

Value

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

Shiny inputs

- input$<id> – "confirm", "cancel" or "close". For a prompt that was confirmed, a list of action and value, where value is the text the user typed.

Examples

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

  ui <- el_page(
    el_button("del", "Delete", type = "danger"),
    verbatimTextOutput("answer")
  )

  server <- function(input, output, session) {
    observeEvent(input$del, {
      el_message_box(
        session,
        "confirm_delete",
        "This cannot be undone.",
        title = "Delete the row?",
        type = "warning"
      )
    })
    observeEvent(input$confirm_delete, {
      output$answer <- renderPrint(input$confirm_delete)
    })
  }
  shinyApp(ui, server)
}
```

---

el_notification *Show Element Plus Notification*

Description

Server-side function to show a desktop-corner notification popup. Requires use_element() or el_page() in the UI to load the JS handler.

Usage

```
el_notification(
  session = shiny::getDefaultReactiveDomain(),
  message,
  title = "",
  type = "info",
  duration = 4500,
  position = "top-right",
```

```
  show_close = TRUE,
  offset = 0,
  custom_class = NULL,
  dangerously_use_html_string = FALSE,
  id = NULL,
  icon = NULL,
  append_to = NULL,
  z_index = NULL,
  close_icon = NULL,
  progress = NULL,
  pause_on_hover = NULL
)
```

Arguments

session	Shiny session; the current one by default, as for shiny::updateTextInput().
message	Notification body text, or a VNode built in the browser: JS() code calling Vue.h(), or a function returning one.
title	Notification title. Default "".
type	Notification type: "info", "success", "warning", or "error". Default "info".
duration	Auto-close delay in milliseconds. 0 disables auto-close. Default 4500.
position	Screen corner: "top-right", "top-left", "bottom-right", or "bottom-left". Default "top-right".
show_close	Whether to show the close button. Default TRUE.
offset	Distance from the corner edge in pixels. Default 0.
custom_class	Extra class name.
dangerously_use_html_string	Whether message is inserted as HTML. Only pass TRUE for markup you control – it is not escaped.
id	Name for this notification, so el_notification_close() can close it, input$<id>_close reports when it closes and input$<id>_click when it is clicked.
icon	Icon to show instead of the one type implies, by name.
append_to	CSS selector of the element it is appended to. Default <body>.
z_index	Its z-index, instead of the next one Element Plus hands out.
close_icon	The close button's icon, by name.
progress	Show a bar counting down duration: TRUE, or a list of Element Plus's progress options.
pause_on_hover	Whether the countdown stops while the pointer is over it. Default TRUE.

Value

A Shiny UI element.

Examples

```
if (interactive()) {
  library(shiny)
  library(shiny.element)
  ui <- el_page(
    el_button("btn", "Notify", type = "primary")
  )
  server <- function(input, output, session) {
    observeEvent(input$btn, {
      el_notification(
        session,
        message = "Operation successful!",
        title = "Success",
        type = "success"
      )
    })
  }
  shinyApp(ui, server)
}
```

`el_option` *A choice of a select, a radio group, a checkbox group*

Description

Element Plus's `el-option` and `el-option-group`, for the `choices` of `el_select()` and the other choice components – where a named vector cannot say that one choice is disabled, or that choices come in groups.

Usage

```
el_option(label, value = label, disabled = NULL)

el_option_group(label, ..., disabled = NULL)
```

Arguments

`label`	The text shown.
`value`	The value reported when it is chosen. Defaults to `label`.
`disabled`	Whether it can be chosen; for a group, whether any of its choices can.
`...`	The group's choices, each an `el_option()`.

Value

A choice, or a group of them, for `choices =`.

See Also

Other items: el_anchor_link(), el_breadcrumb_item(), el_carousel_item(), el_collapse_item(), el_descriptions_item(), el_dropdown_item(), el_menu_item(), el_skeleton_item(), el_step(), el_tab_pane(), el_table_column(), el_table_v2_column(), el_timeline_item(), el_tour_step()

Examples

```
el_select(
  "city",
  choices = list(
    el_option_group(
      "Popular cities",
      el_option("Shanghai"),
      el_option("Beijing", disabled = TRUE)
    ),
    el_option_group("City name", el_option("Chengdu"), el_option("Dalian"))
  )
)
```

---

el_page *Element Plus Page Wrapper with Theme Support*

Description

Top-level page constructor that loads Element-UI, Vue, and layout CSS dependencies, and supports both bslib/shiny themes and Element-UI layout CSS.

Usage

```
el_page(
  ...,
  title = NULL,
  theme = el_theme(),
  theme_css = el_layout_css_dependency(),
  offline = TRUE,
  locale = getOption("shiny.element.locale", "en"),
  dev = .vue_dev(),
  size = NULL,
  z_index = NULL
)
```

Arguments

...	UI elements to include in the page body.
title	Optional page title.

theme	Bootstrap theme for the rest of the page: a bslib::bs_theme(). The default, el_theme(), carries Element's own colours, font and sizes, so Shiny's inputs and outputs match the Element components beside them. NULL gives Shiny's plain Bootstrap 3.
theme_css	Element's layout CSS, el_layout_css_dependency(); NULL leaves it out.
offline	Serve Element Plus from the copy bundled with this package rather than the unpkg CDN. See element_plus_dependency().
locale	Language for Element Plus's built-in text – pagination summaries, date-picker buttons, select placeholders. English by default, or getOption("shiny.element.locale") when set; "zh-CN" gives Element's own Simplified Chinese. See el_locales() for the rest.
dev	Load Vue's development build (vue.global.js) instead of the production one. The production build strips every warning, which is why a template that fails to compile renders nothing and says nothing. Defaults to getOption("shiny.vue.dev") (or shiny.element.dev), FALSE unless set, so it can be turned on for a whole session without touching the UI code. A package installed from CRAN does not carry the development build and loads it from the unpkg CDN; one installed from its sources has it.
size, z_index	Element Plus's global config, as app.use() gives it: the size of every component not given one of its own ("large", "default" or "small"), and the z-index its popups start from (2000 by default). NULL leaves Element's default.

Details

Use this as the root UI function for your Shiny app. You can combine bslib layouts (such as page_sidebar, layout_columns) and Element components (such as el_button).

Element's layout (el_container(), el_row(), el_col()) and bslib's (page_sidebar(), layout_columns()) each work here; nest one inside a cell of the other rather than interleaving them.

A page returned by shiny::renderUI() works there too. Like Shiny's and bslib's page functions it is a whole page, not one tag, so shinyjs's hidden() cannot wrap it: hide its output instead – shinyjs::hidden(uiOutput("id")), or shinyjs::hide("id") later. Inside renderUI() the components alone are enough: each brings what it needs.

Value

A Shiny UI element.

Examples

```
el_page(
  title = "My app",
  el_input("name", value = "Ada"),
  el_button("go", "Submit", type = "primary")
)

# English component text, and Vue's development build for debugging
el_page(locale = "en", dev = TRUE, el_input("name"))
```

`el_page_header` *Element Plus Page Header*

Description

A page title with a back link.

Usage

```
el_page_header(
  id = NULL,
  title = NULL,
  content = NULL,
  icon = NULL,
  width = NULL,
  slots = NULL,
  events = NULL,
  on = NULL,
  session = NULL
)

update_el_page_header(
  session = shiny::getDefaultReactiveDomain(),
  id,
  title = NULL,
  content = NULL,
  icon = NULL
)
```

Arguments

`id`	Header ID. Auto-generated if `NULL`.
`title`	Text of the back link. Default `"Back"`.
`content`	The page's own title, shown after the separator.
`icon`	Icon component of page header. Element Plus's `icon` (string / Component). An icon's name, such as `"Search"`.
`width`	Component width, as a CSS unit.
`slots`	Named list of Element slot contents, such as `list(title = shiny::tags$b("Bold"))`. A shiny.element component given here is absorbed rather than nested. For a scoped slot, write the template with `template()`.
`events`	Element's events to report besides those reported unasked, by name: `events = "node_drop"` reports `input$<id>_node_drop`. The component's are listed under "Shiny inputs", and by `el_events()`; a name it does not have is an error.

on — Handlers of your own, for an event not reported or to send something else: a named list of `JS()` functions, one per event – Element's, or a DOM event with Vue's modifiers (`"keyup.enter"`). Each is called with `report` and the event's arguments; `report(name, value)` sets `input$<id>_<name>`. See `el_widget()`.

session — In `el_page_header()`, deprecated: inside a module, wrap `id` in `ns()`, as for any Shiny input; a session given here namespaces `id` once more, with a warning. In `update_el_page_header()`, the Shiny session, the current one by default, as for `shiny::updateTextInput()`.

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
`input$<id>_back`	unasked	triggers when right side is clicked

The same list as `el_events("el_page_header")`, which says how an event's arguments travel.

Updating from the server

Server-side update for `el_page_header()`.

Every other argument of `el_page_header()` that can change once it is drawn is an argument here too, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_page_header()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_page_header("hdr", content = "Sales for March")
el_page_header("hdr", title = "All reports", content = "Sales for March")

if (interactive()) {
  library(shiny)
  ui <- el_page(
    el_page_header("hdr", content = "Detail"),
    verbatimTextOutput("where")
  )
  server <- function(input, output, session) {
    output$where <- renderPrint(input$hdr_back)
  }
  shinyApp(ui, server)
}
if (interactive()) {
  # inside a server function
  observeEvent(input$row_click, {
    update_el_page_header(session, "hdr", content = selected_name())
  })
}
```

el_pagination *Element Plus Pagination Component*

Description

Creates an Element Plus pagination bar with page navigation, size selector, jump-to-page input, and total count display.

Usage

```
el_pagination(
  id = NULL,
  total,
  page_size = 10,
  current_page = 1,
  page_sizes = c(10, 20, 30, 50),
  layout = "total, sizes, prev, pager, next, jumper",
  background = FALSE,
  small = FALSE,
  disabled = FALSE,
  pager_count = 7,
  prev_text = NULL,
  next_text = NULL,
  hide_on_single_page = NULL,
  page_count = NULL,
  popper_class = NULL,
  append_size_to = NULL,
  default_current_page = NULL,
  default_page_size = NULL,
  next_icon = NULL,
  popper_style = NULL,
  prev_icon = NULL,
  size = NULL,
  teleported = NULL,
  width = NULL,
  slots = NULL,
  events = NULL,
  on = NULL,
  session = NULL
)

update_el_pagination(
  session = shiny::getDefaultReactiveDomain(),
  id,
  total = NULL,
  current_page = NULL,
  page_size = NULL,
```

```
  disabled = NULL,
  page_sizes = NULL,
  layout = NULL,
  background = NULL,
  small = NULL,
  pager_count = NULL,
  prev_text = NULL,
  next_text = NULL,
  hide_on_single_page = NULL,
  page_count = NULL,
  popper_class = NULL,
  append_size_to = NULL,
  next_icon = NULL,
  popper_style = NULL,
  prev_icon = NULL,
  size = NULL,
  teleported = NULL
)
```

Arguments

`id`	Pagination ID. Auto-generated UUID if `NULL`.
`total`	Total item count (required).
`page_size`	Items per page. Default `10`.
`current_page`	Current page number (1-based). Default `1`.
`page_sizes`	Vector of page-size options. Default `c(10, 20, 30, 50)`.
`layout`	Comma-separated list of layout elements. Default `"total, sizes, prev, pager, next, jumper"`.
`background`	Whether to use background colour on page buttons. Default `FALSE`.
`small`	Whether to use compact (small) mode. Default `FALSE`.
`disabled`	Whether the pagination is disabled. Default `FALSE`.
`pager_count`	Number of pager buttons to show. Default `7`.
`prev_text`	Text of the previous-page button, in place of the arrow icon.
`next_text`	Text of the next-page button, in place of the arrow icon.
`hide_on_single_page`	Whether to hide the pager when there is only one page.
`page_count`	Total page count. Set either this or `total`.
`popper_class`	Extra class name for the page-size dropdown.
`append_size_to`	Which element the size dropdown appends to. Element Plus's `append-size-to` (string).
`default_current_page`	Default initial value of current-page, not setting is the same as setting 1. Element Plus's `default-current-page` (number).

`default_page_size`
: Default initial value of page size, not setting is the same as setting 10. Element Plus's `default-page-size` (number).

`next_icon`
: Icon for the next button, has a lower priority than `next-text`. Element Plus's `next-icon` (string / Component). An icon's name, such as `"Search"`.

`popper_style`
: Custom style for the page size Select's dropdown. Element Plus's `popper-style` (string / object).

`prev_icon`
: Icon for the prev button, has a lower priority than `prev-text`. Element Plus's `prev-icon` (string / Component). An icon's name, such as `"Search"`.

`size`
: Pagination size. Element Plus's `size` ('large' | 'default' | 'small').

`teleported`
: Whether Pagination select dropdown is teleported to the body. Element Plus's `teleported` (boolean).

`width`
: Component width, as a CSS unit – `"200px"`, `"50%"`, or a number taken as pixels. Element's own markup carries it, so it behaves like the `width` argument of a Shiny input.

`slots`
: Named list of Element slot contents, such as `list(title = shiny::tags$b("Bold"))`. A shiny.element component given here is absorbed rather than nested. For a scoped slot, write the template with `template()`.

`events`
: Element's events to report besides those reported unasked, by name: `events = "node_drop"` reports `input$<id>_node_drop`. The component's are listed under "Shiny inputs", and by `el_events()`; a name it does not have is an error.

`on`
: Handlers of your own, for an event not reported or to send something else: a named list of `JS()` functions, one per event – Element's, or a DOM event with Vue's modifiers (`"keyup.enter"`). Each is called with `report` and the event's arguments; `report(name, value)` sets `input$<id>_<name>`. See `el_widget()`.

`session`
: In `el_pagination()`, deprecated: inside a module, wrap `id` in `ns()`, as for any Shiny input; a session given here namespaces `id` once more, with a warning. In `update_el_pagination()`, the Shiny session, the current one by default, as for `shiny::updateTextInput()`.

Value

An `htmltools` tagList with a Vue-managed pagination component.

Shiny inputs

Input	Reported	Value
`input$<id>`	unasked	the current page, from 1
`input$<id>_page_size`	unasked	the page size
`input$<id>_prev_click`	`events = "prev_click"`	triggers when the prev button is clicked and current page changes
`input$<id>_next_click`	`events = "next_click"`	triggers when the next button is clicked and current page changes
`input$<id>_change`	`events = "change"`	triggers when current-page or page-size changes

The same list as `el_events("el_pagination")`, which says how an event's arguments travel.

Updating from the server

Server-side update for `el_pagination()`.

Every other argument of `el_pagination()` that can change once it is drawn is an argument here too, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_pagination()` is called for its side effect and returns `NULL` invisibly.

Examples

```
# Basic usage
el_pagination("pg1", total = 100)

# With custom page sizes and layout
el_pagination(
  "pg2",
  total = 500,
  page_size = 20,
  page_sizes = c(10, 20, 50, 100),
  layout = "total, sizes, prev, pager, next"
)

# Shiny app example
if (interactive()) {
  library(shiny)
  library(shiny.element)
  ui <- el_page(
    el_pagination("pg1", total = 200),
    verbatimTextOutput("page_info")
  )
  server <- function(input, output, session) {
    output$page_info <- renderPrint({
      list(page = input$pg1, size = input$pg1_page_size)
    })
  }
  shinyApp(ui, server)
}
if (interactive()) {
  # inside a server function
  observeEvent(input$go, {
    update_el_pagination(session, "pager", current_page = 2)
  })
}
```

---

el_popconfirm *Element Plus Confirmation Bubble*

Description

A small confirmation prompt anchored to the element that triggers it, for actions that warrant a check but not a dialog.

Usage

```
el_popconfirm(
  id = NULL,
  reference = NULL,
  title = NULL,
  confirm_button_text = NULL,
  cancel_button_text = NULL,
  confirm_button_type = NULL,
  cancel_button_type = NULL,
  icon = NULL,
  icon_color = NULL,
  hide_icon = NULL,
  effect = NULL,
  hide_after = NULL,
  persistent = NULL,
  teleported = NULL,
  popconfirm_width = NULL,
  ...,
  width = NULL,
  slots = NULL,
  on = NULL,
  session = NULL,
  placement = NULL
)

update_el_popconfirm(
  session = shiny::getDefaultReactiveDomain(),
  id,
  title = NULL,
  confirm_button_text = NULL,
  cancel_button_text = NULL
)
```

Arguments

id	Popconfirm ID. Auto-generated if NULL.
reference	The element that opens the prompt. Any Shiny UI, including another shiny.element component – that component is folded into this one's Vue instance rather than nested inside it, so its inputs keep reporting. Its update_el_*() no longer reaches it, though.
title	The question.
confirm_button_text, cancel_button_text	Button labels.
confirm_button_type, cancel_button_type	Button types, as in el_button(). Default "primary" and "text".
icon	Icon class shown beside the question.
icon_color	Colour of that icon.

hide_icon	Whether to leave the icon out. Default FALSE.
effect	Tooltip theme, built-in theme: dark / light. Element Plus's effect ('dark' | 'light' / string).
hide_after	Delay of disappear, in millisecond. Element Plus's hide-after (number).
persistent	When popconfirm inactive and persistent is false , popconfirm will be destroyed. Element Plus's persistent (boolean).
teleported	Whether popconfirm is teleported to the body. Element Plus's teleported (boolean).
popconfirm_width	Width of the prompt: pixels, or a CSS width; at least 150px. Element Plus's width, named apart from width (the component's own box) as el_popover()'s popover_width is.
...	Any other attribute of Element Plus's tooltip, which the popconfirm inherits, in snake_case: enterable = FALSE, offset = 20.
width	Component width, as a CSS unit.
slots	Named list of Element slot contents, such as list(title = shiny::tags$b("Bold")). A shiny.element component given here is absorbed rather than nested. For a scoped slot, write the template with template().
on	Handlers of your own, for an event not reported or to send something else: a named list of JS() functions, one per event – Element's, or a DOM event with Vue's modifiers ("keyup.enter"). Each is called with report and the event's arguments; report(name, value) sets input$<id>_<name>. See el_widget().
session	In el_popconfirm(), deprecated: inside a module, wrap id in ns(), as for any Shiny input; a session given here namespaces id once more, with a warning. In update_el_popconfirm(), the Shiny session, the current one by default, as for shiny::updateTextInput().
placement	Where the popup goes: "top", "bottom-start" and the rest of the tooltip's placements. Passed through to its tooltip, as upstream passes it. Default "bottom".

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
input$<id>_confirm	unasked	fires when the user confirms
input$<id>_cancel	unasked	fires when the user backs out

The same list as el_events("el_popconfirm"), which says how an event's arguments travel.

Updating from the server

Server-side update for el_popconfirm().

update_el_popconfirm() is called for its side effect and returns NULL invisibly.

Examples

```
el_popconfirm(
  "del",
  reference = el$button(type = "danger", "Delete"),
  title = "Delete this row?"
)

if (interactive()) {
  library(shiny)
  ui <- el_page(
    el_popconfirm(
      "del",
      reference = el$button(type = "danger", "Delete"),
      title = "Delete this row?"
    )
  )
  server <- function(input, output, session) {
    observeEvent(input$del_confirm, {
      showNotification("Deleted")
    })
  }
  shinyApp(ui, server)
}
if (interactive()) {
  # inside a server function
  observeEvent(input$row_click, {
    update_el_popconfirm(
      session,
      "del",
      title = paste0("Delete ", selected_name(), "?")
    )
  })
}
```

---

el_popover *Element Plus Popover*

Description

A card shown on click or hover, with a title and body.

Usage

```
el_popover(
  id = NULL,
  reference = NULL,
  title = NULL,
  content = NULL,
  body = NULL,
```

```
  trigger = NULL,
  placement = NULL,
  popover_width = NULL,
  disabled = NULL,
  offset = NULL,
  transition = NULL,
  popper_class = NULL,
  popper_options = NULL,
  tabindex = NULL,
  append_to = NULL,
  auto_close = NULL,
  effect = NULL,
  hide_after = NULL,
  persistent = NULL,
  popper_style = NULL,
  show_after = NULL,
  show_arrow = NULL,
  teleported = NULL,
  trigger_keys = NULL,
  virtual_ref = NULL,
  virtual_triggering = NULL,
  visible = NULL,
  ...,
  width = NULL,
  slots = NULL,
  events = NULL,
  on = NULL,
  session = NULL
)

update_el_popover(
  session = shiny::getDefaultReactiveDomain(),
  id,
  title = NULL,
  content = NULL,
  disabled = NULL,
  visible = NULL
)
```

Arguments

id	Popover ID. Auto-generated if NULL.
reference	The element the popover hangs off. Any Shiny UI, including another shiny.element component – that component is folded into the popover's Vue instance rather than nested inside it, so its inputs keep reporting. Its update_el_*() no longer reaches it.
title	Title of the card.
content	Body text. For richer content pass body.

body	Body markup, used instead of content. Components are absorbed here too.
trigger	How it opens: "click", "focus", "hover" (default) or "contextmenu".
placement	Where it appears: "bottom" (default), "top", "left", "right", each also with -start and -end.
popover_width	Width of the card, in pixels. Element's own width prop, named apart from width so the two are not confused.
disabled	Whether the popover is suppressed.
offset	Offset from the reference, in pixels.
transition	Name of the transition to animate with.
popper_class	Extra class name for the card.
popper_options	Additional Popper.js options, as a named list.
tabindex	Tabindex of Popover. Element Plus's tabindex (number / string).
append_to	Which element the popover CONTENT appends to. Element Plus's append-to (CSSSelector / HTMLElement).
auto_close	Timeout in milliseconds to hide tooltip, not valid in controlled mode. Element Plus's auto-close (number).
effect	Tooltip theme, built-in theme: dark / light. Element Plus's effect ('dark' | 'light' / string).
hide_after	Delay of disappear, in millisecond, not valid in controlled mode. Element Plus's hide-after (number).
persistent	When popover inactive and persistent is false , popover will be destroyed. Element Plus's persistent (boolean).
popper_style	Custom style for popover. Element Plus's popper-style (string / object).
show_after	Delay of appearance, in millisecond, not valid in controlled mode. Element Plus's show-after (number).
show_arrow	Whether a tooltip arrow is displayed or not. For more info, please refer to ElPopper. Element Plus's show-arrow (boolean).
teleported	Whether popover dropdown is teleported to the body. Element Plus's teleported (boolean).
trigger_keys	When you click the mouse to focus on the trigger element, you can define a set of keyboard codes to control the display of popover through the keyboard, not valid in controlled mode. Element Plus's trigger-keys (Array).
virtual_ref	A CSS selector, "#help-icon", for an element elsewhere on the page that the popover is attached to, in place of a trigger of its own. Element Plus's virtual-ref takes the element itself; the selector is looked up in the browser.
virtual_triggering	Whether virtual triggering is enabled. Element Plus's virtual-triggering (boolean); TRUE when virtual_ref is given.
visible	Whether popover is visible. Element Plus's visible (boolean / null).
...	Any other attribute of Element Plus's tooltip, which the popover inherits, in snake_case: enterable = FALSE, offset = 20.

width	Component width, as a CSS unit.
slots	Named list of Element slot contents, such as `list(title = shiny::tags$b("Bold"))`. A shiny.element component given here is absorbed rather than nested. For a scoped slot, write the template with `template()`.
events	Element's events to report besides those reported unasked, by name: `events = "node_drop"` reports `input$<id>_node_drop`. The component's are listed under "Shiny inputs", and by `el_events()`; a name it does not have is an error.
on	Handlers of your own, for an event not reported or to send something else: a named list of `JS()` functions, one per event – Element's, or a DOM event with Vue's modifiers (`"keyup.enter"`). Each is called with `report` and the event's arguments; `report(name, value)` sets `input$<id>_<name>`. See `el_widget()`.
session	In `el_popover()`, deprecated: inside a module, wrap `id` in `ns()`, as for any Shiny input; a session given here namespaces `id` once more, with a warning. In `update_el_popover()`, the Shiny session, the current one by default, as for `shiny::updateTextInput()`.

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
`input$<id>_show`	`events = "show"`	triggers when popover shows
`input$<id>_hide`	`events = "hide"`	triggers when popover hides
`input$<id>_after_enter`	`events = "after_enter"`	triggers when the entering transition ends
`input$<id>_after_leave`	`events = "after_leave"`	triggers when the leaving transition ends
`input$<id>_before_enter`	`events = "before_enter"`	triggers when the entering transition before
`input$<id>_before_leave`	`events = "before_leave"`	triggers when the leaving transition before

The same list as `el_events("el_popover")`, which says how an event's arguments travel.

Updating from the server

Server-side update for `el_popover()`. Setting `visible` opens or closes the card, which then stays as set: Element Plus's popover is controlled by its `visible` once that is given.

`update_el_popover()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_popover(
  "info",
  reference = el$button("Details"),
  title = "March",
  content = "Revenue up 4% on February."
)
```

```
# Hover, with markup in the body
el_popover(
  "info",
  reference = el$button("Details"),
  body = shiny::tags$ul(shiny::tags$li("One"), shiny::tags$li("Two")),
  trigger = "hover",
  placement = "right"
)
if (interactive()) {
  # inside a server function
  observeEvent(input$explain, {
    update_el_popover(session, "info", content = summary_text(), visible = TRUE)
  })
}
```

---

el_progress *Element Plus Progress Component*

Description

Creates an Element Plus progress bar. This is a display-only component; update it from the server with update_el_progress().

Usage

```
el_progress(
  id = NULL,
  percentage = 0,
  type = "line",
  status = NULL,
  stroke_width = 6,
  text_inside = FALSE,
  show_text = TRUE,
  color = NULL,
  width = 126,
  stroke_linecap = NULL,
  slots = NULL,
  format = NULL,
  duration = NULL,
  indeterminate = NULL,
  striped = NULL,
  striped_flow = NULL,
  on = NULL,
  session = NULL
)

update_el_progress(
```

```
  session = shiny::getDefaultReactiveDomain(),
  id,
  percentage = NULL,
  type = NULL,
  status = NULL,
  color = NULL,
  stroke_width = NULL,
  show_text = NULL,
  text_inside = NULL,
  stroke_linecap = NULL,
  format = NULL,
  duration = NULL,
  indeterminate = NULL,
  striped = NULL,
  striped_flow = NULL
)
```

Arguments

id	Progress ID. Auto-generated UUID if NULL.
percentage	Progress percentage, 0 to 100. Default 0.
type	Progress bar type: "line", "circle", or "dashboard". Default "line".
status	Status theme: NULL, "success", "exception", or "warning". NULL means no status colour. Default NULL.
stroke_width	Stroke width in pixels. Default 6.
text_inside	Whether to display the percentage text inside the bar (only applies to type = "line"). Default FALSE.
show_text	Whether to show the progress text. Default TRUE.
color	Custom colour string (e.g. "#409EFF"). Overrides status colour when set. Default NULL.
width	Width in pixels for "circle" and "dashboard" types. Default 126.
stroke_linecap	Shape of the bar's ends: "round" (default), "butt" or "square".
slots	Named list of Element slot contents, such as list(title = shiny::tags$b("Bold")). A shiny.element component given here is absorbed rather than nested. For a scoped slot, write the template with template().
format	JS() function function(percentage) returning the text shown.
duration	Control the animation duration of indeterminate progress or striped flow progress. Element Plus's duration (number).
indeterminate	Set indeterminate progress. Element Plus's indeterminate (boolean).
striped	Stripe over the progress bar's color. Element Plus's striped (boolean).
striped_flow	Get the stripes to flow. Element Plus's striped-flow (boolean).
on	Handlers of your own, for an event not reported or to send something else: a named list of JS() functions, one per event – Element's, or a DOM event with Vue's modifiers ("keyup.enter"). Each is called with report and the event's arguments; report(name, value) sets input$<id>_<name>. See el_widget().

session — In el_progress(), deprecated: inside a module, wrap id in ns(), as for any Shiny input; a session given here namespaces id once more, with a warning. In update_el_progress(), the Shiny session, the current one by default, as for shiny::updateTextInput().

Value

An htmltools tagList with a Vue-managed progress component.

Updating from the server

Server-side update for el_progress().

Every other argument of el_progress() that can change once it is drawn is an argument here too, under the same name. One left NULL stays as it is; NA returns it to Element's default.

Unlike the other updates, which go with the flush (flush_vue()), it is sent at once, as shiny::withProgress() reports: a loop updating it shows every step.

update_el_progress() is called for its side effect and returns NULL invisibly.

Examples

```
# Basic line progress
el_progress("prog1", percentage = 60)

# Circle progress with success status
el_progress("prog2", percentage = 100, type = "circle", status = "success")

# Dashboard style with custom colour
el_progress("prog3", percentage = 75, type = "dashboard", color = "#67C23A")

# Shiny app example
if (interactive()) {
  library(shiny)
  library(shiny.element)
  ui <- el_page(
    el_progress("prog1", percentage = 0),
    actionButton("go", "Advance")
  )
  server <- function(input, output, session) {
    observeEvent(input$go, {
      update_el_progress(
        session,
        "prog1",
        percentage = min(100, (input$go * 10))
      )
    })
  }
  shinyApp(ui, server)
}
if (interactive()) {
  # inside a server function
  observeEvent(input$go, {
```

```
    update_el_progress(session, "pct", percentage = 100)
  })
}
```

---

el_radio_group *Element Plus Radio Group Component*

Description

Creates an Element Plus <el-radio-group> component backed by a Vue instance. Supports both standard radio buttons (<el-radio>) and button-style radios (<el-radio-button>).

Usage

```
el_radio_group(
  id = NULL,
  choices = NULL,
  selected = NULL,
  disabled = FALSE,
  size = NULL,
  button = FALSE,
  fill = NULL,
  text_color = NULL,
  label = NULL,
  label_position = c("top", "left", "right"),
  label_width = NULL,
  label_suffix = NULL,
  required = FALSE,
  error = NULL,
  show_message = TRUE,
  inline_message = FALSE,
  width = NULL,
  slots = NULL,
  value = NULL,
  options = NULL,
  aria_label = NULL,
  props = NULL,
  type = NULL,
  validate_event = NULL,
  on = NULL,
  session = NULL
)

update_el_radio_group(
  session = shiny::getDefaultReactiveDomain(),
  id,
  selected = NULL,
```

```
  choices = NULL,
  disabled = NULL,
  value = NULL,
  options = NULL,
  label = NULL,
  error = NULL,
  size = NULL,
  fill = NULL,
  text_color = NULL,
  aria_label = NULL,
  props = NULL,
  type = NULL,
  validate_event = NULL
)
```

Arguments

id	Input ID. Auto-generated UUID if NULL.
choices, options	The choices: a named character vector (c(Label = value)) or a list of el_option()s – which can be disabled – or of list(value = ..., label = ...). Unnamed vectors are allowed; the element is used as both value and label. choices is Shiny's name for it, options Element's; give either.
selected, value	Initially selected value; nothing by default. selected is Shiny's name, value Element's (its v-model); give either.
disabled	Whether the entire group is disabled. Default FALSE.
size	Size: "large", "default" or "small"; NULL follows the form or the page. Only affects button-style radios (button = TRUE).
button	Whether to render as <el-radio-button> (pill/button style) instead of standard <el-radio>. Default FALSE.
fill	Border and background colour of a checked radio button.
text_color	Text colour of a checked radio button.
label	A label shown with the component, as Shiny's inputs have: text or a tag. NULL, the default, shows none. It is the component's accessible name too – tied to it with for where the component has a native input that takes the id <id>-input, else with aria-labelledby.
label_position	Where the label sits, as el_form()'s label_position: "top" (the default, as Shiny's labels sit), or beside the component, its text aligned "left" or "right" – which shows once label_width gives the labels a common width.
label_width	Width of a label beside the component, as a CSS unit, so that several line up. Element's label-width.
label_suffix	Text after the label, such as ":". Element's label-suffix.
required	Draw Element's red asterisk before the label. It marks the field; it does not check it – shinyvalidate or el_form() does that.

`error` An error message shown under the component in Element's style, the field framed in red. Element's `error`.

`show_message, inline_message`
Whether `error`'s message is shown, and whether beside the component rather than under it. Element's `show-message` and `inline-message`.

`width` Component width, as a CSS unit – `"200px"`, `"50%"`, or a number taken as pixels. Element's own markup carries it, so it behaves like the `width` argument of a Shiny input.

`slots` Named list of Element slot contents, such as `list(title = shiny::tags$b("Bold"))`. A shiny.element component given here is absorbed rather than nested. For a scoped slot, write the template with `template()`.

`aria_label` Same as `aria-label` in RadioGroup. Element Plus's `aria-label` (string).

`props` Configuration options. Element Plus's `props` (`{ value?: string, label?: string, disabled?: stri`

`type` Component type to render options (e.g. `'button'`). Element Plus's `type` ('radio' | 'button').

`validate_event` Whether to trigger form validation. Element Plus's `validate-event` (boolean).

`on` Handlers of your own, for an event not reported or to send something else: a named list of `JS()` functions, one per event – Element's, or a DOM event with Vue's modifiers (`"keyup.enter"`). Each is called with `report` and the event's arguments; `report(name, value)` sets `input$<id>_<name>`. See `el_widget()`.

`session` In `el_radio_group()`, deprecated: inside a module, wrap `id` in `ns()`, as for any Shiny input; a session given here namespaces `id` once more, with a warning. In `update_el_radio_group()`, the Shiny session, the current one by default, as for `shiny::updateTextInput()`.

Value

An `htmltools` tagList containing the Vue-managed radio group.

Shiny inputs

Input	Reported	Value
`input$<id>`	unasked	the value picked

The same list as `el_events("el_radio_group")`, which says how an event's arguments travel.

Updating from the server

Server-side update for `el_radio_group()`. Sends a custom message to update reactive fields on the underlying Vue instance.

Every other argument of `el_radio_group()` that can change once it is drawn is an argument here too, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_radio_group()` is called for its side effect and returns `NULL` invisibly.

Examples

```
# Standard radio buttons from a named vector
el_radio_group(
  "size",
  choices = c(Small = "s", Medium = "m", Large = "l"),
  selected = "m"
)

# Button-style radio group
el_radio_group(
  "theme",
  choices = c(Light = "light", Dark = "dark"),
  button = TRUE,
  size = "small"
)

# Shiny app example
if (interactive()) {
  library(shiny)
  library(shiny.element)
  ui <- el_page(
    el_radio_group(
      "fruit",
      choices = c(Apple = "apple", Banana = "banana", Cherry = "cherry"),
      selected = "apple"
    ),
    verbatimTextOutput("selected")
  )
  server <- function(input, output, session) {
    output$selected <- renderPrint(input$fruit)
  }
  shinyApp(ui, server)
}
if (interactive()) {
  # inside a server function
  observeEvent(input$go, {
    update_el_radio_group(session, "plan", selected = "pro")
  })
}
```

---

el_rate *Element Plus Rate (Star Rating)*

Description

A star-rating input. Supports half-stars, custom icons, and read-only display.

Usage

```
el_rate(
  id = NULL,
  value = 0,
  max = 5L,
  disabled = FALSE,
  allow_half = FALSE,
  show_text = FALSE,
  show_score = FALSE,
  texts = NULL,
  text_color = NULL,
  score_template = "{value}",
  colors = NULL,
  void_color = NULL,
  disabled_void_color = NULL,
  low_threshold = NULL,
  high_threshold = NULL,
  label = NULL,
  label_position = c("top", "left", "right"),
  label_width = NULL,
  label_suffix = NULL,
  required = FALSE,
  error = NULL,
  show_message = TRUE,
  inline_message = FALSE,
  aria_label = NULL,
  clearable = NULL,
  disabled_void_icon = NULL,
  icons = NULL,
  size = NULL,
  void_icon = NULL,
  width = NULL,
  slots = NULL,
  on = NULL,
  session = NULL
)

update_el_rate(
  session = shiny::getDefaultReactiveDomain(),
  id,
  value = NULL,
  disabled = NULL,
  label = NULL,
  error = NULL,
  max = NULL,
  allow_half = NULL,
  show_text = NULL,
  show_score = NULL,
```

```
  texts = NULL,
  text_color = NULL,
  score_template = NULL,
  void_color = NULL,
  disabled_void_color = NULL,
  low_threshold = NULL,
  high_threshold = NULL,
  aria_label = NULL,
  clearable = NULL,
  disabled_void_icon = NULL,
  icons = NULL,
  size = NULL,
  void_icon = NULL
)
```

Arguments

id	Rate ID. Auto-generated UUID if NULL.
value	Initial rating value. Default 0.
max	Maximum number of stars. Default 5.
disabled	Read-only display mode. Default FALSE.
allow_half	Whether to allow half-star selection. Default FALSE.
show_text	Whether to show descriptive text beside the stars. Uses the texts vector. Default FALSE.
show_score	Whether to show the numeric score. Default FALSE.
texts	Character vector of length max used when show_text = TRUE. NULL takes Element Plus's: "Extremely bad", "Disappointed", "Fair", "Satisfied", "Surprise".
text_color	Colour of the text or score. NULL takes Element Plus's, from its CSS variables.
score_template	Template for score display. {value} is replaced. Default "{value}".
colors	Colours for the three score levels, or a named list keyed by threshold.
void_color	Colour of unselected icons.
disabled_void_color	Colour of unselected icons when disabled = TRUE.
low_threshold	Scores at or below this use the first colour and icon. Default 2.
high_threshold	Scores above this use the third colour and icon. Default 4.
label	A label shown with the component, as Shiny's inputs have: text or a tag. NULL, the default, shows none. It is the component's accessible name too – tied to it with for where the component has a native input that takes the id <id>-input, else with aria-labelledby.
label_position	Where the label sits, as el_form()'s label_position: "top" (the default, as Shiny's labels sit), or beside the component, its text aligned "left" or "right" – which shows once label_width gives the labels a common width.
label_width	Width of a label beside the component, as a CSS unit, so that several line up. Element's label-width.

label_suffix
: Text after the label, such as ":". Element's label-suffix.

required
: Draw Element's red asterisk before the label. It marks the field; it does not check it – shinyvalidate or el_form() does that.

error
: An error message shown under the component in Element's style, the field framed in red. Element's error.

show_message, inline_message
: Whether error's message is shown, and whether beside the component rather than under it. Element's show-message and inline-message.

aria_label
: Same as aria-label in Rate. Element Plus's aria-label (string).

clearable
: Whether value can be reset to 0. Element Plus's clearable (boolean).

disabled_void_icon
: Component of unselected read-only icons. Element Plus's disabled-void-icon (string / Component). An icon's name, such as "Search".

icons
: Icon components. If array, it should have 3 elements, each of which corresponds with a score level, else if object, the key should be threshold value between two levels, and the value should be corresponding icon component. Element Plus's icons (string[] | Component[] / Record<number, string | Component>). An icon's name, such as "Search".

size
: Size of Rate. Element Plus's size ('large' | 'default' | 'small').

void_icon
: Component of unselected icons. Element Plus's void-icon (string / Component). An icon's name, such as "Search".

width
: Component width, as a CSS unit – "200px", "50%", or a number taken as pixels. Element's own markup carries it, so it behaves like the width argument of a Shiny input.

slots
: Named list of Element slot contents, such as list(title = shiny::tags$b("Bold")). A shiny.element component given here is absorbed rather than nested. For a scoped slot, write the template with template().

on
: Handlers of your own, for an event not reported or to send something else: a named list of JS() functions, one per event – Element's, or a DOM event with Vue's modifiers ("keyup.enter"). Each is called with report and the event's arguments; report(name, value) sets input$<id>_<name>. See el_widget().

session
: In el_rate(), deprecated: inside a module, wrap id in ns(), as for any Shiny input; a session given here namespaces id once more, with a warning. In update_el_rate(), the Shiny session, the current one by default, as for shiny::updateTextInput().

Value

An htmltools tagList with a Vue-managed rate component.

Shiny inputs

Input	Reported	Value
input$<id>	unasked	the rating

The same list as el_events("el_rate"), which says how an event's arguments travel.

Updating from the server

Server-side update for `el_rate()`.

Every other argument of `el_rate()` that can change once it is drawn is an argument here too, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_rate()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_rate("rate1", value = 3)
el_rate("rate2", allow_half = TRUE, show_score = TRUE)
if (interactive()) {
  # inside a server function
  observeEvent(input$go, {
    update_el_rate(session, "stars", value = 5)
  })
}
```

---

el_result *Element Plus Result*

Description

The outcome of an operation: an icon, a title, a line of detail, and what to do next.

Usage

```
el_result(
  id = NULL,
  ...,
  icon = NULL,
  title = NULL,
  sub_title = NULL,
  width = NULL,
  slots = NULL,
  on = NULL,
  session = NULL
)

update_el_result(
  session = shiny::getDefaultReactiveDomain(),
  id,
  icon = NULL,
  title = NULL,
  sub_title = NULL
)
```

Arguments

id	Component ID. Auto-generated if NULL.
...	What to do next, shown under the text – usually buttons. A shiny.element component here is absorbed, not nested, and keeps reporting its inputs.
icon	"primary", "success", "warning", "info" or "error".
title	Headline.
sub_title	Detail under the headline.
width	Component width, as a CSS unit.
slots	Named list of Element slot contents: icon, title, sub-title, extra.
on	Handlers of your own, for an event not reported or to send something else: a named list of JS() functions, one per event – Element's, or a DOM event with Vue's modifiers ("keyup.enter"). Each is called with report and the event's arguments; report(name, value) sets input$<id>_<name>. See el_widget().
session	In el_result(), deprecated: inside a module, wrap id in ns(), as for any Shiny input; a session given here namespaces id once more, with a warning. In update_el_result(), the Shiny session, the current one by default, as for shiny::updateTextInput().

Value

A Shiny UI element.

Updating from the server

Server-side update for el_result().

update_el_result() is called for its side effect and returns NULL invisibly.

Examples

```
el_result(
  "done",
  icon = "success",
  title = "Report submitted",
  sub_title = "It will be reviewed within a day",
  el_button("back", "Back to the list", type = "primary")
)
if (interactive()) {
  # inside a server function
  observeEvent(input$submit, {
    ok <- tryCatch(
      {
        save()
        TRUE
      },
      error = function(e) FALSE
    )
    update_el_result(
```

```
      session,
      "outcome",
      icon = if (ok) "success" else "error",
      title = if (ok) "Saved" else "Could not save"
    )
  })
}
```

---

el_row *Element Plus Layout Row*

Description

Emits `<div class="el-row">` directly rather than an `<el-row>` custom tag. Nothing mounts a Vue instance over page-level markup, so a custom tag would never be compiled and would render as an unstyled inline element; the Element Plus stylesheet is already loaded, so the class name is all that is needed.

Usage

```
el_row(
  ...,
  gutter = NULL,
  type = NULL,
  justify = NULL,
  align = NULL,
  tag = "div",
  class = NULL,
  style = NULL
)
```

Arguments

`...`	Child columns (`el_col()`) or other content.
`gutter`	Spacing between columns, in pixels.
`type`	Element UI's `"flex"`, kept so old code runs: Element Plus's row is always a flex row, and `justify` and `align` apply without it.
`justify`	Flex horizontal alignment: `"start"` (default), `"center"`, `"end"`, `"space-between"` or `"space-around"`.
`align`	Flex vertical alignment: `"top"` (default), `"middle"` or `"bottom"`.
`tag`	HTML element to render, as Element's `tag`. Default `"div"`; `"ul"` and `"li"` suit a grid of list items.
`class`	Extra CSS classes.
`style`	Extra inline style.

Value

A Shiny UI element.

Examples

```
# Two equal columns with a 20px gutter
el_row(
  gutter = 20,
  el_col(span = 12, "left"),
  el_col(span = 12, "right")
)

# Centred row
el_row(
  justify = "center",
  align = "middle",
  el_col(span = 8, "centred")
)
```

---

el_rule *Declare a validation rule*

Description

Builds one async-validator rule, the format Element Plus's form expects.

Usage

```
el_rule(
  required = NULL,
  min = NULL,
  max = NULL,
  len = NULL,
  pattern = NULL,
  type = NULL,
  message = NULL,
  trigger = "blur",
  enum = NULL,
  whitespace = NULL,
  validator = NULL,
  transform = NULL
)
```

Arguments

required	Whether the field must be filled.
min, max	Minimum and maximum: length for strings, value for numbers.

`len`	Exact length.
`pattern`	A regular expression the value must match.
`type`	Value type to check: `"string"`, `"number"`, `"boolean"`, `"integer"`, `"float"`, `"array"`, `"object"`, `"enum"`, `"date"`, `"url"`, `"hex"`, `"email"` or `"any"`.
`message`	Text shown when the rule fails.
`trigger`	When to run the rule: `"blur"`, `"change"`, or both as `c("blur", "change")`.
`enum`	Allowed values, with `type = "enum"`.
`whitespace`	Whether a value of only whitespace counts as empty, with `required = TRUE`.
`validator`	A `JS()` function `function(rule, value, callback)` that calls `callback()` when the value is valid and `callback(new Error("message"))` when it is not – Element's custom rule. A check only the server can make goes through `update_el_form()`'s `errors` instead.
`transform`	A `JS()` function turning the value into what the rule checks, such as `function(v) { return v.trim(); }`

Value

A rule, for `el_form_field()`'s `rules` argument.

Examples

```
el_rule(required = TRUE, message = "Please enter your name")
el_rule(type = "email", message = "Not a valid email", trigger = "blur")
el_rule(type = "enum", enum = c("a", "b"), message = "a or b")
# Element's custom validator
el_rule(
  validator = JS(
    "function(rule, value, callback) {",
    "  value % 2 === 0 ? callback() : callback(new Error('An even number'));",
    "}"
  ),
  trigger = "change"
)
```

`el_scrollbar` *Element Plus Scrollbar*

Description

A scroll area with Element's own thin scrollbars.

Usage

```
el_scrollbar(
  ...,
  id = NULL,
  height = NULL,
  max_height = NULL,
  native = NULL,
  wrap_style = NULL,
  wrap_class = NULL,
  view_style = NULL,
  view_class = NULL,
  noresize = NULL,
  tag = NULL,
  always = NULL,
  min_size = NULL,
  role = NULL,
  aria_label = NULL,
  aria_orientation = NULL,
  tabindex = NULL,
  distance = NULL,
  width = NULL,
  slots = NULL,
  events = NULL,
  on = NULL
)

update_el_scrollbar(
  session = shiny::getDefaultReactiveDomain(),
  id,
  height = NULL,
  max_height = NULL,
  native = NULL,
  wrap_style = NULL,
  wrap_class = NULL,
  view_style = NULL,
  view_class = NULL,
  noresize = NULL,
  always = NULL,
  min_size = NULL,
  aria_label = NULL,
  tabindex = NULL,
  distance = NULL,
  tag = NULL,
  role = NULL,
  aria_orientation = NULL
)
```

Arguments

`...`	Its content: any Shiny UI. Components of this package are folded into this one's Vue instance, as `el_button_group()` folds its buttons.
`id`	Component ID. Auto-generated if `NULL`.
`height`	Height of scrollbar. Element Plus's `height` (string / number).
`max_height`	Max height of scrollbar. Element Plus's `max-height` (string / number).
`native`	Whether to use the native scrollbar style. Element Plus's `native` (boolean).
`wrap_style`	Style of wrap container. Element Plus's `wrap-style` (string / CSSProperties | CSSProperties[] | string[]).
`wrap_class`	Class of wrap container. Element Plus's `wrap-class` (string).
`view_style`	Style of view. Element Plus's `view-style` (`string / CSSProperties | CSSProperties[] | string[]`).
`view_class`	Class of view. Element Plus's `view-class` (string).
`noresize`	Do not respond to container size changes, if the container size does not change, it is better to set it to optimize performance. Element Plus's `noresize` (boolean).
`tag`	Element tag of the view. Element Plus's `tag` (string).
`always`	Always show scrollbar. Element Plus's `always` (boolean).
`min_size`	Minimum size of scrollbar. Element Plus's `min-size` (number).
`role`	Role of view. Element Plus's `role` (string).
`aria_label`	Aria-label of view. Element Plus's `aria-label` (string).
`aria_orientation`	Aria-orientation of view. Element Plus's `aria-orientation` ('horizontal' | 'vertical').
`tabindex`	Tabindex of wrap container. Element Plus's `tabindex` (number / string).
`distance`	Trigger end-reached event distance(px). Element Plus's `distance` (number).
`width`	Component width, as a CSS unit.
`slots`	Named list of Element slot contents. A scoped slot is written with `template()`.
`events`	Element's events to report besides those reported unasked, by name: `events = "node_drop"` reports `input$<id>_node_drop`. The component's are listed under "Shiny inputs", and by `el_events()`; a name it does not have is an error.
`on`	Handlers of your own, for an event not reported or to send something else: a named list of `JS()` functions, one per event – Element's, or a DOM event with Vue's modifiers (`"keyup.enter"`). Each is called with `report` and the event's arguments; `report(name, value)` sets `input$<id>_<name>`. See `el_widget()`.
`session`	Shiny session; the current one by default, as for `shiny::updateTextInput()`.

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
`input$<id>_scroll`	`events = "scroll"`	`list(scrollLeft, scrollTop)`, at most every 200 ms
`input$<id>_end_reached`	unasked	triggers when the end of a scroll is triggered

The same list as `el_events("el_scrollbar")`, which says how an event's arguments travel.

Element methods

Callable with `call_el()`: `handleScroll()`, `scrollTo()`, `setScrollTop()`, `setScrollLeft()`, `update()`.

Updating from the server

`update_el_scrollbar()` changes the component from the server: every argument of `el_scrollbar()` that can change once it is drawn, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_scrollbar()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_scrollbar(height = "200px", lapply(1:20, function(i) shiny::tags$p(i)))
```

---

el_segmented *Element Plus Segmented*

Description

A row of options, one of them selected: a compact radio group.

Usage

```
el_segmented(
  id = NULL,
  value = NULL,
  options = NULL,
  size = NULL,
  block = NULL,
  disabled = NULL,
  validate_event = NULL,
  aria_label = NULL,
  direction = NULL,
  props = NULL,
  label = NULL,
  label_position = c("top", "left", "right"),
```

```
  label_width = NULL,
  label_suffix = NULL,
  required = FALSE,
  error = NULL,
  show_message = TRUE,
  inline_message = FALSE,
  width = NULL,
  slots = NULL,
  on = NULL
)

update_el_segmented(
  session = shiny::getDefaultReactiveDomain(),
  id,
  value = NULL,
  disabled = NULL,
  label = NULL,
  error = NULL,
  size = NULL,
  block = NULL,
  validate_event = NULL,
  aria_label = NULL,
  direction = NULL,
  props = NULL
)
```

Arguments

id	Component ID. Auto-generated if NULL.
value	Binding value: Element Plus's model-value, reported as input$<id>.
options	The options: a named vector c(Label = value), a vector, or a list of list(value =, label =, disabled =), as Element Plus takes them.
size	Size of component. Element Plus's size (" | 'large' | 'default' | 'small').
block	Fit width of parent content. Element Plus's block (boolean).
disabled	Whether segmented is disabled. Element Plus's disabled (boolean).
validate_event	Whether to trigger form validation. Element Plus's validate-event (boolean).
aria_label	Native aria-label attribute. Element Plus's aria-label (string).
direction	Display direction. Element Plus's direction ('horizontal' | 'vertical').
props	Which field of an option holds what, when the options are records named otherwise: list(value =, label =, disabled =), Element Plus's props.
label	A label shown with the component, as Shiny's inputs have: text or a tag. NULL, the default, shows none. It is the component's accessible name too – tied to it with for where the component has a native input that takes the id <id>-input, else with aria-labelledby.

`label_position` Where the label sits, as `el_form()`'s `label_position`: `"top"` (the default, as Shiny's labels sit), or beside the component, its text aligned `"left"` or `"right"` – which shows once `label_width` gives the labels a common width.

`label_width` Width of a label beside the component, as a CSS unit, so that several line up. Element's `label-width`.

`label_suffix` Text after the label, such as `":"`. Element's `label-suffix`.

`required` Draw Element's red asterisk before the label. It marks the field; it does not check it – shinyvalidate or `el_form()` does that.

`error` An error message shown under the component in Element's style, the field framed in red. Element's `error`.

`show_message, inline_message`
Whether `error`'s message is shown, and whether beside the component rather than under it. Element's `show-message` and `inline-message`.

`width` Component width, as a CSS unit.

`slots` Named list of Element slot contents. A scoped slot is written with `template()`.

`on` Handlers of your own, for an event not reported or to send something else: a named list of `JS()` functions, one per event – Element's, or a DOM event with Vue's modifiers (`"keyup.enter"`). Each is called with `report` and the event's arguments; `report(name, value)` sets `input$<id>_<name>`. See `el_widget()`.

`session` Shiny session; the current one by default, as for `shiny::updateTextInput()`.

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
`input$<id>`	unasked	the value

The same list as `el_events("el_segmented")`, which says how an event's arguments travel.

Updating from the server

Server-side update for `el_segmented()`.

Every other argument of `el_segmented()` that can change once it is drawn is an argument here too, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_segmented()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_segmented(
  "period",
  options = c(Day = "d", Week = "w", Month = "m"),
  value = "w"
)
if (interactive()) {
  # inside a server function
  observeEvent(input$reset, update_el_segmented(session, "x", value = NULL))
}
```

---

el_select *Element Plus Select Component*

Description

Creates an Element Plus <el-select> component backed by a Vue instance. Supports single and multiple selection, filtering, and all standard Element Plus select props.

Usage

```
el_select(
  id = NULL,
  choices = NULL,
  selected = NULL,
  multiple = FALSE,
  placeholder = NULL,
  disabled = FALSE,
  clearable = FALSE,
  filterable = FALSE,
  size = NULL,
  multiple_limit = 0,
  collapse_tags = FALSE,
  value_key = NULL,
  name = NULL,
  autocomplete = NULL,
  automatic_dropdown = NULL,
  allow_create = NULL,
  loading = NULL,
  loading_text = NULL,
  no_match_text = NULL,
  no_data_text = NULL,
  popper_class = NULL,
  reserve_keyword = NULL,
  default_first_option = NULL,
  remote = NULL,
  filter_method = NULL,
```

```
  remote_method = NULL,
  label = NULL,
  label_position = c("top", "left", "right"),
  label_width = NULL,
  label_suffix = NULL,
  required = FALSE,
  error = NULL,
  show_message = TRUE,
  inline_message = FALSE,
  width = NULL,
  option_template = NULL,
  slots = NULL,
  value = NULL,
  options = NULL,
  append_to = NULL,
  aria_label = NULL,
  clear_icon = NULL,
  collapse_tags_tooltip = NULL,
  debounce = NULL,
  effect = NULL,
  empty_values = NULL,
  fallback_placements = NULL,
  fit_input_width = NULL,
  max_collapse_tags = NULL,
  offset = NULL,
  persistent = NULL,
  placement = NULL,
  popper_options = NULL,
  popper_style = NULL,
  remote_show_suffix = NULL,
  show_arrow = NULL,
  suffix_icon = NULL,
  suffix_transition = NULL,
  tabindex = NULL,
  tag_effect = NULL,
  tag_type = NULL,
  teleported = NULL,
  validate_event = NULL,
  value_on_clear = NULL,
  props = NULL,
  tag_tooltip = NULL,
  events = NULL,
  on = NULL,
  session = NULL
)

update_el_select(
  session = shiny::getDefaultReactiveDomain(),
```

```
  id,
  selected = NULL,
  choices = NULL,
  disabled = NULL,
  placeholder = NULL,
  clearable = NULL,
  filterable = NULL,
  multiple_limit = NULL,
  loading = NULL,
  loading_text = NULL,
  no_match_text = NULL,
  no_data_text = NULL,
  value = NULL,
  options = NULL,
  label = NULL,
  error = NULL,
  multiple = NULL,
  size = NULL,
  collapse_tags = NULL,
  value_key = NULL,
  name = NULL,
  autocomplete = NULL,
  automatic_dropdown = NULL,
  allow_create = NULL,
  popper_class = NULL,
  reserve_keyword = NULL,
  remote = NULL,
  filter_method = NULL,
  remote_method = NULL,
  append_to = NULL,
  aria_label = NULL,
  clear_icon = NULL,
  collapse_tags_tooltip = NULL,
  debounce = NULL,
  effect = NULL,
  empty_values = NULL,
  fallback_placements = NULL,
  fit_input_width = NULL,
  max_collapse_tags = NULL,
  offset = NULL,
  persistent = NULL,
  placement = NULL,
  popper_options = NULL,
  popper_style = NULL,
  remote_show_suffix = NULL,
  show_arrow = NULL,
  suffix_icon = NULL,
  suffix_transition = NULL,
```

```
  tabindex = NULL,
  tag_effect = NULL,
  tag_type = NULL,
  teleported = NULL,
  validate_event = NULL,
  value_on_clear = NULL,
  tag_tooltip = NULL
)
```

Arguments

id	Input ID. Auto-generated UUID if NULL.
choices, options	The choices: a named character vector (c(Label = value)), a list of el_option()s (or of list(value = ..., label = ...)), or a list of el_option_group()s (or a named list of options) for option groups. Unnamed vectors are allowed; the element is used as both value and label. choices is Shiny's name for it, options Element's; give either. Empty by default, for a select whose options arrive later (remote = TRUE).
selected, value	Initially selected value(s); a character vector for multiple selection. selected is Shiny's name, value Element's (its v-model); give either.
multiple	Whether multiple items can be selected. Default FALSE.
placeholder	Placeholder text shown when nothing is selected.
disabled	Whether the select is disabled. Default FALSE.
clearable	Whether to show a clear button. Default FALSE.
filterable	Whether typing filters the options. Default FALSE.
size	Size: "large", "default" or "small"; NULL follows the form or the page.
multiple_limit	Maximum number of items that can be selected when multiple = TRUE. 0 means unlimited. Default 0.
collapse_tags	Whether to collapse selected tags into a summary when multiple = TRUE. Default FALSE.
value_key	Key that identifies an option when values are objects. Default "value".
name	Native name attribute.
autocomplete	Native autocomplete attribute. Default "off".
automatic_dropdown	Whether a filterable select opens its menu on focus.
allow_create	Whether the user may create options not in the list. Needs filterable = TRUE.
loading	Whether to show the loading state while options are being fetched.
loading_text	Text shown while loading. Default "Loading".
no_match_text	Text shown when filtering matches nothing.
no_data_text	Text shown when there are no options at all.
popper_class	Extra class name for the dropdown panel.

`reserve_keyword`
: Whether a multiple filterable select keeps the search term after selecting.

`default_first_option`
: Whether Enter picks the first matching option.

`remote`
: Whether options are fetched from the server as the user types. Needs `filterable = TRUE`; see "Shiny inputs".

`filter_method`
: `JS()` function filtering the options as the user types.

`remote_method`
: `JS()` function fetching options in the browser instead of from the server. Needs `remote = TRUE`.

`label`
: A label shown with the component, as Shiny's inputs have: text or a tag. `NULL`, the default, shows none. It is the component's accessible name too – tied to it with `for` where the component has a native input that takes the id `<id>-input`, else with `aria-labelledby`.

`label_position`
: Where the label sits, as `el_form()`'s `label_position`: `"top"` (the default, as Shiny's labels sit), or beside the component, its text aligned `"left"` or `"right"` – which shows once `label_width` gives the labels a common width.

`label_width`
: Width of a label beside the component, as a CSS unit, so that several line up. Element's `label-width`.

`label_suffix`
: Text after the label, such as `":"`. Element's `label-suffix`.

`required`
: Draw Element's red asterisk before the label. It marks the field; it does not check it – shinyvalidate or `el_form()` does that.

`error`
: An error message shown under the component in Element's style, the field framed in red. Element's `error`.

`show_message, inline_message`
: Whether `error`'s message is shown, and whether beside the component rather than under it. Element's `show-message` and `inline-message`.

`width`
: Component width, as a CSS unit – `"200px"`, `"50%"`, or a number taken as pixels. Element's own markup carries it, so it behaves like the `width` argument of a Shiny input.

`option_template`
: Markup drawn inside each option, in place of its label – Element's "custom template". The option is in reach as `opt`, with any field its choice carries – `{{ opt.label }}`, `{{ opt.code }}` – for choices given as `list(value =, label =, code =)`. See the example.

`slots`
: Named list of Element slot contents, such as `list(title = shiny::tags$b("Bold"))`. A shiny.element component given here is absorbed rather than nested. For a scoped slot, write the template with `template()`.

`append_to`
: Which element the select dropdown appends to. Element Plus's `append-to` (CSSSelector / HTMLElement).

`aria_label`
: Same as `aria-label` in native input. Element Plus's `aria-label` (string).

`clear_icon`
: Custom clear icon component. Element Plus's `clear-icon` (string / Component). An icon's name, such as `"Search"`.

collapse_tags_tooltip
: Whether show all selected tags when mouse hover text of collapse-tags. To use this, collapse-tags must be true. Element Plus's collapse-tags-tooltip (boolean).

debounce
: Debounce delay during remote search, in milliseconds. Element Plus's debounce (number).

effect
: Tooltip theme, built-in theme: dark / light. Element Plus's effect ('dark' | 'light' / string).

empty_values
: Empty values of component, see config-provider. Element Plus's empty-values (array).

fallback_placements
: List of possible positions for dropdown popper.js. Element Plus's fallback-placements (Placement[]).

fit_input_width
: Whether the width of the dropdown is the same as the input. Element Plus's fit-input-width (boolean).

max_collapse_tags
: The max tags number to be shown. To use this, collapse-tags must be true. Element Plus's max-collapse-tags (number).

offset
: Offset of the dropdown. Element Plus's offset (number).

persistent
: When select dropdown is inactive and persistent is false, select dropdown will be destroyed. Element Plus's persistent (boolean).

placement
: Position of dropdown. Element Plus's placement (enum).

popper_options
: Popper.js parameters. Element Plus's popper-options (object).

popper_style
: Custom style for Select's dropdown and tags' tooltip. Element Plus's popper-style (string / object).

remote_show_suffix
: In remote search method show suffix icon. Element Plus's remote-show-suffix (boolean).

show_arrow
: Whether the dropdown has an arrow. Element Plus's show-arrow (boolean).

suffix_icon
: Custom suffix icon component. Element Plus's suffix-icon (string / Component). An icon's name, such as "Search".

suffix_transition
: Animation when dropdown appears/disappears icon. Element Plus's suffix-transition (boolean).

tabindex
: Tabindex for input. Element Plus's tabindex (string / number).

tag_effect
: Tag effect. Element Plus's tag-effect ('' | 'light' | 'dark' | 'plain').

tag_type
: Tag type. Element Plus's tag-type ('' | 'success' | 'info' | 'warning' | 'danger').

teleported
: Whether select dropdown is teleported, if true it will be teleported to where append-to sets. Element Plus's teleported (boolean).

validate_event
: Whether to trigger form validation. Element Plus's validate-event (boolean).

value_on_clear
: Clear return value, see config-provider. Element Plus's value-on-clear (string / number / boolean / Function). Give it as JS().

`props` Which field of an option holds what, when the options are records named otherwise: `list(value =, label =, disabled =, options =)`, Element Plus's `props`.

`tag_tooltip` Settings for the tooltip listing collapsed tags, with `collapse_tags` and `collapse_tags_tooltip`: a named list of tooltip attributes (`placement`, `effect`, ...). Element Plus's `tag-tooltip`.

`events` Element's events to report besides those reported unasked, by name: `events = "node_drop"` reports `input$<id>_node_drop`. The component's are listed under "Shiny inputs", and by `el_events()`; a name it does not have is an error.

`on` Handlers of your own, for an event not reported or to send something else: a named list of `JS()` functions, one per event – Element's, or a DOM event with Vue's modifiers (`"keyup.enter"`). Each is called with `report` and the event's arguments; `report(name, value)` sets `input$<id>_<name>`. See `el_widget()`.

`session` In `el_select()`, deprecated: inside a module, wrap `id` in `ns()`, as for any Shiny input; a session given here namespaces `id` once more, with a warning. In `update_el_select()`, the Shiny session, the current one by default, as for `shiny::updateTextInput()`.

Value

An `htmltools` tagList containing the Vue-managed select component.

Element methods

Callable with `call_el()`:

- `blur()` – Blur the Input component, and hide the dropdown
- `focus()` – Focus the Input component

Shiny inputs

Input	Reported	Value
`input$<id>`	unasked	the value, several with `multiple`
`input$<id>_query`	unasked	with `remote = TRUE`, the text typed; answer with `update_e`
`input$<id>_visible_change`	`events = "visible_change"`	triggers when the dropdown appears/disappears
`input$<id>_remove_tag`	`events = "remove_tag"`	triggers when a tag is removed in multiple mode
`input$<id>_clear`	`events = "clear"`	triggers when the clear icon is clicked in a clearable Select
`input$<id>_blur`	`events = "blur"`	triggers when Input blurs
`input$<id>_focus`	`events = "focus"`	triggers when Input focuses
`input$<id>_end_reached`	unasked	triggers when dropdown scroll reaches an end
`input$<id>_popup_scroll`	`events = "popup_scroll"`	`list(scroll_left, scroll_top)`, at most every 200 ms

The same list as `el_events("el_select")`, which says how an event's arguments travel.

With `remote = TRUE`, `filterable = TRUE` and no `remote_method` of your own, the server does the search, as `selectizeInput()`'s server mode does: `input$<id>_query` is the text typed, and `update_el_select()` with the matching `choices` answers it – the select shows Element's loading text until then, or for 30 seconds at most.

Updating from the server

Server-side update for `el_select()`. Sends a custom message to update reactive fields on the underlying Vue instance.

Every other argument of `el_select()` that can change once it is drawn is an argument here too, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_select()` is called for its side effect and returns `NULL` invisibly.

Examples

```
# Each option drawn with a second field beside its label
el_select(
  "city",
  choices = list(
    list(value = "bj", label = "Beijing", code = "PEK"),
    list(value = "sh", label = "Shanghai", code = "SHA")
  ),
  option_template = htmltools::tagList(
    htmltools::tags$span(style = "float: left", "{{ opt.label }}"),
    htmltools::tags$span(
      style = "float: right; color: #8492a6",
      "{{ opt.code }}"
    )
  )
)

# Single-select from a named vector
el_select(
  "sel1",
  choices = c(Apple = "apple", Banana = "banana", Cherry = "cherry"),
  selected = "banana"
)

# Shiny app example
if (interactive()) {
  library(shiny)
  library(shiny.element)
  ui <- el_page(
    el_select(
      "fruit",
      choices = c(Apple = "apple", Banana = "banana", Cherry = "cherry"),
      selected = "apple",
      clearable = TRUE
    ),
    verbatimTextOutput("selected")
  )
  server <- function(input, output, session) {
    output$selected <- renderPrint(input$fruit)
  }
  shinyApp(ui, server)
}
if (interactive()) {
```

```
  # inside a server function
  observeEvent(input$go, {
    update_el_select(session, "city", selected = "sh")
  })
}
```

---

el_select_v2 *Element Plus Virtualized Select*

Description

A select drawing only the options in view, for lists of thousands.

Usage

```
el_select_v2(
  id = NULL,
  value = NULL,
  options = NULL,
  multiple = NULL,
  disabled = NULL,
  value_key = NULL,
  size = NULL,
  clearable = NULL,
  clear_icon = NULL,
  collapse_tags = NULL,
  multiple_limit = NULL,
  effect = NULL,
  autocomplete = NULL,
  placeholder = NULL,
  filterable = NULL,
  allow_create = NULL,
  filter_method = NULL,
  loading = NULL,
  loading_text = NULL,
  reserve_keyword = NULL,
  default_first_option = NULL,
  no_match_text = NULL,
  no_data_text = NULL,
  popper_class = NULL,
  popper_style = NULL,
  teleported = NULL,
  append_to = NULL,
  persistent = NULL,
  popper_options = NULL,
  automatic_dropdown = NULL,
  fit_input_width = NULL,
```

```
  suffix_icon = NULL,
  height = NULL,
  item_height = NULL,
  estimated_option_height = NULL,
  scrollbar_always_on = NULL,
  remote = NULL,
  debounce = NULL,
  remote_method = NULL,
  remote_show_suffix = NULL,
  validate_event = NULL,
  offset = NULL,
  show_arrow = NULL,
  placement = NULL,
  fallback_placements = NULL,
  collapse_tags_tooltip = NULL,
  max_collapse_tags = NULL,
  tag_type = NULL,
  tag_effect = NULL,
  aria_label = NULL,
  empty_values = NULL,
  value_on_clear = NULL,
  popper_append_to_body = NULL,
  tabindex = NULL,
  props = NULL,
  tag_tooltip = NULL,
  label = NULL,
  label_position = c("top", "left", "right"),
  label_width = NULL,
  label_suffix = NULL,
  required = FALSE,
  error = NULL,
  show_message = TRUE,
  inline_message = FALSE,
  width = NULL,
  slots = NULL,
  events = NULL,
  on = NULL
)

update_el_select_v2(
  session = shiny::getDefaultReactiveDomain(),
  id,
  value = NULL,
  disabled = NULL,
  label = NULL,
  error = NULL,
  multiple = NULL,
  value_key = NULL,
```

```
  size = NULL,
  clearable = NULL,
  clear_icon = NULL,
  collapse_tags = NULL,
  multiple_limit = NULL,
  effect = NULL,
  autocomplete = NULL,
  placeholder = NULL,
  filterable = NULL,
  allow_create = NULL,
  filter_method = NULL,
  loading = NULL,
  loading_text = NULL,
  reserve_keyword = NULL,
  no_match_text = NULL,
  no_data_text = NULL,
  popper_class = NULL,
  popper_style = NULL,
  teleported = NULL,
  append_to = NULL,
  persistent = NULL,
  popper_options = NULL,
  automatic_dropdown = NULL,
  fit_input_width = NULL,
  suffix_icon = NULL,
  height = NULL,
  item_height = NULL,
  estimated_option_height = NULL,
  scrollbar_always_on = NULL,
  remote = NULL,
  debounce = NULL,
  remote_method = NULL,
  remote_show_suffix = NULL,
  validate_event = NULL,
  offset = NULL,
  show_arrow = NULL,
  placement = NULL,
  fallback_placements = NULL,
  collapse_tags_tooltip = NULL,
  max_collapse_tags = NULL,
  tag_type = NULL,
  tag_effect = NULL,
  aria_label = NULL,
  empty_values = NULL,
  value_on_clear = NULL,
  popper_append_to_body = NULL,
  tabindex = NULL,
  props = NULL,
```

```
  tag_tooltip = NULL
)
```

Arguments

id	Component ID. Auto-generated if NULL.
value	Binding value: Element Plus's model-value, reported as input$<id>.
options	The options: a named vector c(Label = value), a vector, or a list of list(value =, label =, disabled =), as Element Plus takes them.
multiple	Is multiple. Element Plus's multiple (boolean).
disabled	Is disabled. Element Plus's disabled (boolean).
value_key	Unique identity key name for value, required when value is an object. Element Plus's value-key (string).
size	Size of component. Element Plus's size ('' | 'large' | 'default' | 'small').
clearable	Whether select can be cleared. Element Plus's clearable (boolean).
clear_icon	Custom clear icon. Element Plus's clear-icon (string / Component). An icon's name, such as "Search".
collapse_tags	Whether to collapse tags to a text when multiple selecting. Element Plus's collapse-tags (boolean).
multiple_limit	Maximum number of options user can select when multiple is true. No limit when set to 0. Element Plus's multiple-limit (number).
effect	Tooltip theme, built-in theme: dark / light. Element Plus's effect ('dark' | 'light' / string).
autocomplete	Autocomplete of select input. Element Plus's autocomplete (string).
placeholder	Placeholder. Element Plus's placeholder (string).
filterable	Whether Select is filterable. Element Plus's filterable (boolean).
allow_create	Whether creating new items is allowed. To use this, filterable must be true. Element Plus's allow-create (boolean).
filter_method	Custom filter method, the first parameter is the current input value. To use this, filterable must be true method. Element Plus's filter-method ((query: string) => void). Give it as JS().
loading	Whether Select is loading data from server. Element Plus's loading (boolean).
loading_text	Displayed text while loading data from server, default is 'Loading'. Element Plus's loading-text (string).
reserve_keyword	Whether reserve the keyword after select filtered option. Element Plus's reserve-keyword (boolean).
default_first_option	Select first matching option on enter key. Use with filterable or remote. Element Plus's default-first-option (boolean).
no_match_text	Displayed text when no data matches the filtering query, you can also use slot empty, default is 'No matching data'. Element Plus's no-match-text (string).

`no_data_text` Displayed text when there is no options, you can also use slot empty. Element Plus's `no-data-text` (string).

`popper_class` Custom class name for Select's dropdown and tags' tooltip. Element Plus's `popper-class` (string / object).

`popper_style` Custom style for Select's dropdown and tags' tooltip. Element Plus's `popper-style` (string / object).

`teleported` Whether select dropdown is teleported, if `true` it will be teleported to where `append-to` sets. Element Plus's `teleported` (boolean).

`append_to` Which element the select dropdown appends to. Element Plus's `append-to` (CSSSelector / HTMLElement).

`persistent` When select dropdown is inactive and `persistent` is `false`, select dropdown will be destroyed. Element Plus's `persistent` (boolean).

`popper_options` Popper.js parameters. Element Plus's `popper-options` (object).

`automatic_dropdown`
For non-filterable Select, this prop decides if the option menu pops up when the input is focused. Element Plus's `automatic-dropdown` (boolean).

`fit_input_width`
Whether the width of the dropdown is the same as the input, if the value is `number`, then the width is fixed. Element Plus's `fit-input-width` (boolean / number).

`suffix_icon` Custom suffix icon component. Element Plus's `suffix-icon` (string / Component). An icon's name, such as `"Search"`.

`height` The height of the dropdown panel, 34px for each item. Element Plus's `height` (number).

`item_height` The height of the dropdown item. Element Plus's `item-height` (number).

`estimated_option_height`
Controls virtual-list sizing mode: if undefined, the list uses fixed item height from `item-height`; if provided, the list uses dynamic item sizing and this value as the estimated item height. Element Plus's `estimated-option-height` (number).

`scrollbar_always_on`
Controls whether the scrollbar is always displayed. Element Plus's `scrollbar-always-on` (boolean).

`remote` Whether search data from server. Element Plus's `remote` (boolean).

`debounce` Debounce delay during remote search, in milliseconds. Element Plus's `debounce` (number).

`remote_method` Function that gets called when the input value changes. Its parameter is the current input value. To use this, `filterable` must be true. Element Plus's `remote-method` ((query: string) => void). Give it as `JS()`.

`remote_show_suffix`
In remote search method show suffix icon. Element Plus's `remote-show-suffix` (boolean).

`validate_event` Whether to trigger form validation. Element Plus's `validate-event` (boolean).

`offset` Offset of the dropdown. Element Plus's `offset` (number).

`show_arrow` Whether the dropdown has an arrow. Element Plus's `show-arrow` (boolean).

`placement` Position of dropdown. Element Plus's `placement` (enum).

`fallback_placements`
List of possible positions for dropdown popper.js. Element Plus's `fallback-placements` (`Placement[]`).

`collapse_tags_tooltip`
Whether show all selected tags when mouse hover text of collapse-tags. To use this, `collapse-tags` must be true. Element Plus's `collapse-tags-tooltip` (boolean).

`max_collapse_tags`
The max tags number to be shown. To use this, `collapse-tags` must be true. Element Plus's `max-collapse-tags` (number).

`tag_type` Tag type. Element Plus's `tag-type` ('' | 'success' | 'info' | 'warning' | 'danger').

`tag_effect` Tag effect. Element Plus's `tag-effect` ('' | 'light' | 'dark' | 'plain').

`aria_label` Same as `aria-label` in native input. Element Plus's `aria-label` (string).

`empty_values` Empty values of component, see config-provider. Element Plus's `empty-values` (array).

`value_on_clear` Clear return value, see config-provider. Element Plus's `value-on-clear` (string / number / boolean / Function). Give it as `JS()`.

`popper_append_to_body`
Whether to append the popper menu to body. If the positioning of the popper is wrong, you can try to set this prop to false. Element Plus's `popper-append-to-body` (boolean).

`tabindex` Tabindex for input. Element Plus's `tabindex` (string / number).

`props` Which field of an option holds what, when the options are records named otherwise: `list(value =, label =, disabled =, options =)`, Element Plus's `props`.

`tag_tooltip` Settings for the tooltip listing collapsed tags, with `collapse_tags` and `collapse_tags_tooltip`: a named list of tooltip attributes (`placement`, `effect`, ...). Element Plus's `tag-tooltip`.

`label` A label shown with the component, as Shiny's inputs have: text or a tag. `NULL`, the default, shows none. It is the component's accessible name too – tied to it with `for` where the component has a native input that takes the id `<id>-input`, else with `aria-labelledby`.

`label_position` Where the label sits, as `el_form()`'s `label_position`: `"top"` (the default, as Shiny's labels sit), or beside the component, its text aligned `"left"` or `"right"` – which shows once `label_width` gives the labels a common width.

`label_width` Width of a label beside the component, as a CSS unit, so that several line up. Element's `label-width`.

`label_suffix` Text after the label, such as `":"`. Element's `label-suffix`.

`required` Draw Element's red asterisk before the label. It marks the field; it does not check it – shinyvalidate or `el_form()` does that.

`error` An error message shown under the component in Element's style, the field framed in red. Element's `error`.

`show_message, inline_message` Whether `error`'s message is shown, and whether beside the component rather than under it. Element's `show-message` and `inline-message`.

`width` Component width, as a CSS unit.

`slots` Named list of Element slot contents: `header`, `footer`, `empty`, `prefix`, `tag`, `loading`, `label`. A scoped slot is written with `template()`.

`events` Element's events to report besides those reported unasked, by name: `events = "node_drop"` reports `input$<id>_node_drop`. The component's are listed under "Shiny inputs", and by `el_events()`; a name it does not have is an error.

`on` Handlers of your own, for an event not reported or to send something else: a named list of `JS()` functions, one per event – Element's, or a DOM event with Vue's modifiers (`"keyup.enter"`). Each is called with `report` and the event's arguments; `report(name, value)` sets `input$<id>_<name>`. See `el_widget()`.

`session` Shiny session; the current one by default, as for `shiny::updateTextInput()`.

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
`input$<id>`	unasked	the value, several with `multiple`
`input$<id>_visible_change`	`events = "visible_change"`	triggers when the dropdown appears/disappears, the param
`input$<id>_remove_tag`	`events = "remove_tag"`	triggers when a tag is removed in multiple mode, the param
`input$<id>_clear`	`events = "clear"`	triggers when the clear icon is clicked in a clearable Select
`input$<id>_blur`	`events = "blur"`	triggers when Input blurs
`input$<id>_focus`	`events = "focus"`	triggers when Input focuses
`input$<id>_end_reached`	unasked	triggers when dropdown scroll reaches an end

The same list as `el_events("el_select_v2")`, which says how an event's arguments travel.

Element methods

Callable with `call_el()`: `focus()`, `blur()`.

Updating from the server

Server-side update for `el_select_v2()`.

Every other argument of `el_select_v2()` that can change once it is drawn is an argument here too, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_select_v2()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_select_v2("city", options = paste("City", 1:10000), filterable = TRUE)
if (interactive()) {
  # inside a server function
  observeEvent(input$reset, update_el_select_v2(session, "x", value = NULL))
}
```

---

el_skeleton *Element Plus Skeleton*

Description

Grey placeholder shapes shown while content is on its way, then the content itself. In Shiny the usual pattern is to start with `loading = TRUE` and switch it off from the server once the work is done.

Usage

```
el_skeleton(
  id = NULL,
  ...,
  loading = TRUE,
  rows = NULL,
  animated = NULL,
  count = NULL,
  throttle = NULL,
  width = NULL,
  class = NULL,
  style = NULL,
  slots = NULL,
  on = NULL,
  session = NULL
)

update_el_skeleton(
  session = shiny::getDefaultReactiveDomain(),
  id,
  loading = NULL,
  rows = NULL,
  animated = NULL,
  count = NULL
)
```

Arguments

`id`	Component ID. Auto-generated if `NULL`.

...	The real content, shown once loading is FALSE. A shiny.element component here is absorbed, not nested.
loading	Whether to show the placeholder. Default TRUE.
rows	Number of placeholder lines. Default 3.
animated	Whether the placeholder shimmers.
count	How many copies of the placeholder to show.
throttle	Delay in milliseconds before the placeholder appears, so a fast load does not flash it.
width	Component width, as a CSS unit.
class, style	Extra classes and inline style on the skeleton, as Element passes them to its root: style = "display: flex; gap: 8px" lays several copies of the placeholder side by side.
slots	Named list of Element slot contents. template replaces the placeholder's shape; build it from el_skeleton_item().
on	Handlers of your own, for an event not reported or to send something else: a named list of JS() functions, one per event – Element's, or a DOM event with Vue's modifiers ("keyup.enter"). Each is called with report and the event's arguments; report(name, value) sets input$<id>_<name>. See el_widget().
session	In el_skeleton(), deprecated: inside a module, wrap id in ns(), as for any Shiny input; a session given here namespaces id once more, with a warning. In update_el_skeleton(), the Shiny session, the current one by default, as for shiny::updateTextInput().

Value

A Shiny UI element.

Updating from the server

Server-side update for el_skeleton(). loading = FALSE swaps the placeholder for the real content.

update_el_skeleton() is called for its side effect and returns NULL invisibly.

Examples

```
el_skeleton("report", rows = 4, animated = TRUE, shiny::tableOutput("summary"))

if (interactive()) {
  library(shiny)
  ui <- el_page(el_skeleton("report", animated = TRUE, tableOutput("summary")))
  server <- function(input, output, session) {
    output$summary <- renderTable({
      Sys.sleep(2)
      on.exit(update_el_skeleton(session, "report", loading = FALSE))
      head(mtcars)
    })
  }
```

```
  shinyApp(ui, server)
}
if (interactive()) {
  # inside a server function
  observeEvent(result(), {
    update_el_skeleton(session, "report", loading = FALSE)
  })
}
```

`el_skeleton_item` *A placeholder shape of* `el_skeleton()`*'s template*

Description

Element Plus's `el-skeleton-item`, for the `template` slot of `el_skeleton()`, which draws a custom placeholder while loading.

Usage

```
el_skeleton_item(variant = NULL, ...)
```

Arguments

`variant`	The shape: `"p"`, `"text"` (Element's default), `"h1"`, `"h3"`, `"caption"`, `"button"`, `"image"`, `"circle"` or `"rect"`.
`...`	Attributes for the tag, such as `style`.

Value

A tag.

See Also

Other items: `el_anchor_link()`, `el_breadcrumb_item()`, `el_carousel_item()`, `el_collapse_item()`, `el_descriptions_item()`, `el_dropdown_item()`, `el_menu_item()`, `el_option()`, `el_step()`, `el_tab_pane()`, `el_table_column()`, `el_table_v2_column()`, `el_timeline_item()`, `el_tour_step()`

Examples

```
el_skeleton(
  slots = list(
    template = htmltools::tagList(
      el_skeleton_item("image", style = "width: 240px; height: 240px"),
      el_skeleton_item("p", style = "width: 50%")
    )
  )
)
```

el_slider *Element Plus Slider Component*

Description

Creates an Element Plus slider with Vue instance, supporting single value and range modes, marks, vertical orientation, and optional numeric input box.

Usage

```
el_slider(
  id = NULL,
  value = 0,
  min = 0,
  max = 100,
  step = 1,
  range = FALSE,
  disabled = FALSE,
  show_input = FALSE,
  show_stops = FALSE,
  show_tooltip = TRUE,
  vertical = FALSE,
  height = NULL,
  marks = NULL,
  label = NULL,
  input_size = NULL,
  show_input_controls = NULL,
  tooltip_class = NULL,
  format_tooltip = NULL,
  label_position = c("top", "left", "right"),
  label_width = NULL,
  label_suffix = NULL,
  required = FALSE,
  error = NULL,
  show_message = TRUE,
  inline_message = FALSE,
  aria_label = NULL,
  format_value_text = NULL,
  persistent = NULL,
  placement = NULL,
  range_end_label = NULL,
  range_start_label = NULL,
  size = NULL,
  validate_event = NULL,
  width = NULL,
  slots = NULL,
  events = NULL,
```

```
  on = NULL,
  session = NULL
)

update_el_slider(
  session = shiny::getDefaultReactiveDomain(),
  id,
  value = NULL,
  min = NULL,
  max = NULL,
  step = NULL,
  disabled = NULL,
  label = NULL,
  error = NULL,
  show_input = NULL,
  show_stops = NULL,
  show_tooltip = NULL,
  vertical = NULL,
  height = NULL,
  marks = NULL,
  input_size = NULL,
  show_input_controls = NULL,
  tooltip_class = NULL,
  format_tooltip = NULL,
  aria_label = NULL,
  format_value_text = NULL,
  persistent = NULL,
  placement = NULL,
  range_end_label = NULL,
  range_start_label = NULL,
  size = NULL,
  validate_event = NULL
)
```

Arguments

id	Slider ID. Auto-generated UUID if NULL.
value	Initial value. A single number, or c(low, high) when range = TRUE. When range = TRUE and a scalar is supplied, the upper bound is set to max.
min	Minimum value. Default 0.
max	Maximum value. Default 100.
step	Step size. Default 1.
range	Whether to enable range selection. Default FALSE.
disabled	Whether the slider is disabled. Default FALSE.
show_input	Whether to display a numeric input box beside the slider (non-range mode only). Default FALSE.

show_stops Whether to display stop markers at each step. Default FALSE.

show_tooltip Whether to display the tooltip when dragging. Default TRUE.

vertical Whether to display in vertical orientation. Default FALSE.

height Height of the slider in vertical mode (e.g., "200px"). Defaults to "200px" when vertical = TRUE and not explicitly provided.

marks Named list of mark labels, e.g., list("0" = "0km", "50" = "50km"). Default NULL (no marks).

label A label shown with the component, as Shiny's inputs have: text or a tag. NULL, the default, shows none. It is the component's accessible name too – tied to it with for where the component has a native input that takes the id <id>-input, else with aria-labelledby.

input_size Size of the companion input when show_input = TRUE.

show_input_controls
Whether the companion input shows its spinner buttons.

tooltip_class Extra class name for the value tooltip.

format_tooltip JS() function formatting the value shown in the tooltip.

label_position Where the label sits, as el_form()'s label_position: "top" (the default, as Shiny's labels sit), or beside the component, its text aligned "left" or "right" – which shows once label_width gives the labels a common width.

label_width Width of a label beside the component, as a CSS unit, so that several line up. Element's label-width.

label_suffix Text after the label, such as ":". Element's label-suffix.

required Draw Element's red asterisk before the label. It marks the field; it does not check it – shinyvalidate or el_form() does that.

error An error message shown under the component in Element's style, the field framed in red. Element's error.

show_message, inline_message
Whether error's message is shown, and whether beside the component rather than under it. Element's show-message and inline-message.

aria_label Native aria-label attribute. Element Plus's aria-label (string).

format_value_text
Format to display the aria-valuenow attribute for screen readers. Element Plus's format-value-text ((value: number) => string).

persistent When slider tooltip inactive and persistent is false , tooltip will be destroyed. persistent always be false when show-tooltip is false. Element Plus's persistent (boolean).

placement Position of Tooltip. Element Plus's placement (enum).

range_end_label
When range is true, screen reader label for the end of the range. Element Plus's range-end-label (string).

range_start_label
When range is true, screen reader label for the start of the range. Element Plus's range-start-label (string).

size	Size of the slider wrapper, will not work in vertical mode. Element Plus's size (" | 'large' | 'default' | 'small').
validate_event	Whether to trigger form validation. Element Plus's validate-event (boolean).
width	Component width, as a CSS unit – "200px", "50%", or a number taken as pixels. Element's own markup carries it, so it behaves like the width argument of a Shiny input.
slots	Named list of Element slot contents, such as list(title = shiny::tags$b("Bold")). A shiny.element component given here is absorbed rather than nested. For a scoped slot, write the template with template().
events	Element's events to report besides those reported unasked, by name: events = "node_drop" reports input$<id>_node_drop. The component's are listed under "Shiny inputs", and by el_events(); a name it does not have is an error.
on	Handlers of your own, for an event not reported or to send something else: a named list of JS() functions, one per event – Element's, or a DOM event with Vue's modifiers ("keyup.enter"). Each is called with report and the event's arguments; report(name, value) sets input$<id>_<name>. See el_widget().
session	In el_slider(), deprecated: inside a module, wrap id in ns(), as for any Shiny input; a session given here namespaces id once more, with a warning. In update_el_slider(), the Shiny session, the current one by default, as for shiny::updateTextInput().

Value

An htmltools tagList with a Vue-managed slider component.

Shiny inputs

Input	Reported	Value
input$<id>	unasked	the value, two for a range, once the drag ends
input$<id>_input	events = "input"	the value while it is dragged, at most every 200 ms

The same list as el_events("el_slider"), which says how an event's arguments travel.

Updating from the server

Server-side update for el_slider(). Supports updating value, range bounds, step size, and disabled state.

Every other argument of el_slider() that can change once it is drawn is an argument here too, under the same name. One left NULL stays as it is; NA returns it to Element's default.

update_el_slider() is called for its side effect and returns NULL invisibly.

Examples

```
# Basic usage
el_slider("slider1", value = 30, min = 0, max = 100)

# Range slider
el_slider("slider2", value = c(20, 80), range = TRUE)

# Vertical slider with marks
el_slider(
  "slider3",
  value = 50,
  vertical = TRUE,
  height = "200px",
  marks = list("0" = "0km", "50" = "50km", "100" = "100km")
)

# Shiny app example
if (interactive()) {
  library(shiny)
  library(shiny.element)
  ui <- el_page(
    el_slider("slider1", value = 50, min = 0, max = 100),
    verbatimTextOutput("val")
  )
  server <- function(input, output, session) {
    output$val <- renderPrint(input$slider1)
  }
  shinyApp(ui, server)
}
if (interactive()) {
  # inside a server function
  observeEvent(input$go, {
    update_el_slider(session, "score", value = 80)
  })
}
```

---

el_space *Element Plus Space*

Description

Even spacing between its items, in a row or a column, wrapping if asked.

Usage

```
el_space(
  ...,
  id = NULL,
  alignment = NULL,
```

```
  direction = NULL,
  spacer = NULL,
  size = NULL,
  wrap = NULL,
  fill = NULL,
  fill_ratio = NULL,
  width = NULL,
  slots = NULL,
  on = NULL
)

update_el_space(
  session = shiny::getDefaultReactiveDomain(),
  id,
  alignment = NULL,
  direction = NULL,
  spacer = NULL,
  size = NULL,
  wrap = NULL,
  fill = NULL,
  fill_ratio = NULL
)
```

Arguments

`...`	Its content: any Shiny UI. Components of this package are folded into this one's Vue instance, as `el_button_group()` folds its buttons.
`id`	Component ID. Auto-generated if `NULL`.
`alignment`	Controls the alignment of items. Element Plus's `alignment` ('center' | 'normal' | 'stretch' | ...).
`direction`	Placement direction. Element Plus's `direction` ('vertical' | 'horizontal').
`spacer`	Spacer. Element Plus's `spacer` (string / number / VNode).
`size`	Spacing size. Element Plus's `size` (`'default' | 'small' | 'large'` / `number` / `[number, number]`).
`wrap`	Auto wrapping. Element Plus's `wrap` (boolean).
`fill`	Whether to fill the container. Element Plus's `fill` (boolean).
`fill_ratio`	Ratio of fill. Element Plus's `fill-ratio` (number).
`width`	Component width, as a CSS unit.
`slots`	Named list of Element slot contents. A scoped slot is written with `template()`.
`on`	Handlers of your own, for an event not reported or to send something else: a named list of `JS()` functions, one per event – Element's, or a DOM event with Vue's modifiers (`"keyup.enter"`). Each is called with `report` and the event's arguments; `report(name, value)` sets `input$<id>_<name>`. See `el_widget()`.
`session`	Shiny session; the current one by default, as for `shiny::updateTextInput()`.

Value

A Shiny UI element.

Shiny inputs

None: it reports nothing.

Updating from the server

`update_el_space()` changes the component from the server: every argument of `el_space()` that can change once it is drawn, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_space()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_space(
  el_button("a", "One"),
  el_button("b", "Two"),
  el_button("c", "Three"),
  size = 20
)
```

---

el_splitter *Element Plus Splitter*

Description

Panels side by side, or one above the other, resized by dragging the bars between them. Give it `el_splitter_panel()`s.

Usage

```
el_splitter(
  ...,
  id = NULL,
  layout = NULL,
  lazy = NULL,
  width = NULL,
  slots = NULL,
  events = NULL,
  on = NULL
)

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

```
  layout = NULL,
  lazy = NULL
)
```

Arguments

...	Its content: any Shiny UI. Components of this package are folded into this one's Vue instance, as `el_button_group()` folds its buttons.
id	Component ID. Auto-generated if `NULL`.
layout	Layout direction of the splitter. Element Plus's `layout` ('horizontal' | 'vertical').
lazy	Whether to enable lazy mode. Element Plus's `lazy` (boolean).
width	Component width, as a CSS unit.
slots	Named list of Element slot contents. A scoped slot is written with `template()`.
events	Element's events to report besides those reported unasked, by name: `events = "node_drop"` reports `input$<id>_node_drop`. The component's are listed under "Shiny inputs", and by `el_events()`; a name it does not have is an error.
on	Handlers of your own, for an event not reported or to send something else: a named list of `JS()` functions, one per event – Element's, or a DOM event with Vue's modifiers (`"keyup.enter"`). Each is called with `report` and the event's arguments; `report(name, value)` sets `input$<id>_<name>`. See `el_widget()`.
session	Shiny session; the current one by default, as for `shiny::updateTextInput()`.

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
`input$<id>_resize_start`	`events = "resize_start"`	Triggered when starting to resize a panel, index is the drag bar i
`input$<id>_resize`	`events = "resize"`	the sizes, at most every 200 ms
`input$<id>_resize_end`	unasked	Triggered when panel resizing ends, index is the drag bar index
`input$<id>_collapse`	unasked	Triggered when a panel is collapsed, index is the drag bar index

The same list as `el_events("el_splitter")`, which says how an event's arguments travel.

Updating from the server

`update_el_splitter()` changes the component from the server: every argument of `el_splitter()` that can change once it is drawn, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_splitter()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_splitter(el_splitter_panel("Left", size = "30%"), el_splitter_panel("Right"))
```

el_splitter_panel *Element Plus Splitter Panel*

Description

One panel of an el_splitter().

Usage

```
el_splitter_panel(
  ...,
  id = NULL,
  size = NULL,
  min = NULL,
  max = NULL,
  resizable = NULL,
  collapsible = NULL,
  width = NULL,
  slots = NULL,
  on = NULL
)

update_el_splitter_panel(
  session = shiny::getDefaultReactiveDomain(),
  id,
  size = NULL,
  min = NULL,
  max = NULL,
  resizable = NULL,
  collapsible = NULL
)
```

Arguments

...	Its content: any Shiny UI. Components of this package are folded into this one's Vue instance, as el_button_group() folds its buttons.
id	Component ID. Auto-generated if NULL.
size	Size of the panel (in pixels or percentage). Element Plus's size (string / number).
min	Minimum size of the panel (in pixels or percentage). Element Plus's min (string / number).
max	Maximum size of the panel (in pixels or percentage). Element Plus's max (string / number).
resizable	Whether the panel can be resized. Element Plus's resizable (boolean).
collapsible	Whether the panel can be collapsed. Element Plus's collapsible (boolean).

width	Component width, as a CSS unit.
slots	Named list of Element slot contents: start-collapsible, end-collapsible. A scoped slot is written with template().
on	Handlers of your own, for an event not reported or to send something else: a named list of JS() functions, one per event – Element's, or a DOM event with Vue's modifiers ("keyup.enter"). Each is called with report and the event's arguments; report(name, value) sets input$<id>_<name>. See el_widget().
session	Shiny session; the current one by default, as for shiny::updateTextInput().

Value

A Shiny UI element.

Shiny inputs

None: it reports nothing.

Updating from the server

update_el_splitter_panel() changes a panel's settings: one inside an el_splitter() is folded into the splitter's instance and still answers to its own id. One left NULL stays as it is; NA returns it to Element's default.

update_el_splitter_panel() is called for its side effect and returns NULL invisibly.

Examples

```
if (interactive()) {
  # inside a server function
  observeEvent(input$lock, {
    update_el_splitter_panel(session, "side", resizable = !input$lock)
  })
}
```

---

el_statistic *Element Plus Statistic*

Description

A headline number with a title, and an optional prefix and suffix. el_countdown() counts down to a moment instead.

Usage

```
el_statistic(
  id = NULL,
  value = 0,
  title = NULL,
  prefix = NULL,
  suffix = NULL,
  precision = NULL,
  decimal_separator = NULL,
  group_separator = NULL,
  value_style = NULL,
  formatter = NULL,
  width = NULL,
  slots = NULL,
  session = NULL
)

el_countdown(
  id = NULL,
  value = 0,
  title = NULL,
  prefix = NULL,
  suffix = NULL,
  format = NULL,
  value_style = NULL,
  width = NULL,
  slots = NULL,
  events = NULL,
  on = NULL,
  session = NULL
)

update_el_statistic(
  session = shiny::getDefaultReactiveDomain(),
  id,
  value = NULL,
  title = NULL,
  prefix = NULL,
  suffix = NULL,
  precision = NULL,
  decimal_separator = NULL,
  group_separator = NULL,
  value_style = NULL,
  formatter = NULL
)

update_el_countdown(
  session = shiny::getDefaultReactiveDomain(),
```

```
  id,
  value = NULL,
  title = NULL,
  prefix = NULL,
  suffix = NULL,
  format = NULL,
  value_style = NULL
)
```

Arguments

id	Component ID. Auto-generated if NULL.
value	The number; for el_countdown(), the moment to count down to, a POSIXct or milliseconds since the epoch.
title	Label above it.
prefix, suffix	Text before and after the number, such as a currency symbol or a unit.
precision	Decimal places to show.
decimal_separator	Decimal point. Default ".".
group_separator	Separator between digit groups. Default ",".
value_style	CSS for the number, as a string or a named list.
formatter	JS() function function(value) returning the text to show, in place of Element's formatting.
width	Component width, as a CSS unit.
slots	Named list of Element slot contents: prefix, suffix, title.
session	In el_statistic(), deprecated: inside a module, wrap id in ns(), as for any Shiny input; a session given here namespaces id once more, with a warning. In update_el_statistic(), the Shiny session, the current one by default, as for shiny::updateTextInput().
format	How a countdown is shown, in day.js's tokens, such as "HH:mm:ss" (the default) or "DD [days] HH:mm:ss".
events	Element's events to report besides those reported unasked, by name: events = "node_drop" reports input$<id>_node_drop. The component's are listed under "Shiny inputs", and by el_events(); a name it does not have is an error.
on	Handlers of your own, for an event not reported or to send something else: a named list of JS() functions, one per event – Element's, or a DOM event with Vue's modifiers ("keyup.enter"). Each is called with report and the event's arguments; report(name, value) sets input$<id>_<name>. See el_widget().

Value

A Shiny UI element.

Shiny inputs

`el_statistic()` reports nothing. `el_countdown()`:

Input	Reported	Value
`input$<id>_finish`	unasked	countdown end event
`input$<id>_change`	`events = "change"`	the milliseconds left, at most once a second

The same list as `el_events("el_countdown")`, which says how an event's arguments travel.

Updating from the server

Server-side update for `el_statistic()` and `el_countdown()`.

Every other argument of `el_statistic()` that can change once it is drawn is an argument here too, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_statistic()` is called for its side effect and returns `NULL` invisibly.

Updating a countdown

`update_el_countdown()` changes the countdown from the server: every argument of `el_countdown()` that can change once it is drawn, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_countdown()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_statistic("users", value = 26048, title = "Active users")
el_statistic(
  "revenue",
  value = 1318.5,
  title = "Revenue",
  prefix = "$",
  precision = 2
)

# A countdown to an hour from now
el_countdown(
  "sale",
  title = "Sale ends in",
  value = Sys.time() + 3600,
  format = "HH:mm:ss"
)
if (interactive()) {
  # inside a server function
  observe(update_el_statistic(session, "users", value = n_users()))
}
```

el_step *A step of* el_steps()

Description

Element Plus's el-step, for el_steps(steps =).

Usage

```
el_step(title = NULL, description = NULL, icon = NULL, status = NULL)
```

Arguments

title	Step title: text, or markup for the step's title slot.
description	Step description: text, or markup.
icon	The step's icon, by name, or markup.
status	The step's status, set by the steps when not given: "", "wait", "process", "finish", "error" or "success".

Value

A step, for el_steps(steps =).

See Also

Other items: el_anchor_link(), el_breadcrumb_item(), el_carousel_item(), el_collapse_item(), el_descriptions_item(), el_dropdown_item(), el_menu_item(), el_option(), el_skeleton_item(), el_tab_pane(), el_table_column(), el_table_v2_column(), el_timeline_item(), el_tour_step()

Examples

```
el_steps(
  "s",
  steps = list(
    el_step("Step 1", "Some description"),
    el_step("Step 2", icon = "Upload"),
    el_step("Step 3", status = "error")
  ),
  active = 1
)
```

el_steps *Element Plus Steps Component*

Description

Element Plus Steps Component

Usage

```
el_steps(
  id = NULL,
  steps = list(),
  active = 0,
  space = NULL,
  direction = "horizontal",
  process_status = "process",
  finish_status = "finish",
  align_center = FALSE,
  simple = FALSE,
  width = NULL,
  slots = NULL,
  events = NULL,
  on = NULL,
  session = NULL
)

update_el_steps(
  session = shiny::getDefaultReactiveDomain(),
  id,
  active = NULL,
  process_status = NULL,
  finish_status = NULL,
  space = NULL,
  direction = NULL,
  align_center = NULL,
  simple = NULL
)
```

Arguments

id	Steps ID (auto-generated if NULL)
steps	A list of steps, each an el_step() – or a list with title, description, icon and status. title, description and icon may be markup rather than text, which fills the step's slot of that name.
active	Current active step index (0-based)
space	Step spacing (number or percentage string)

direction	Display direction ("horizontal" or "vertical")
process_status	Status of current step
finish_status	Status of finished steps
align_center	Center align title and description
simple	Apply simple style
width	Component width, as a CSS unit – "200px", "50%", or a number taken as pixels. Element's own markup carries it, so it behaves like the width argument of a Shiny input.
slots	Named list of Element slot contents, such as list(title = shiny::tags$b("Bold")). A shiny.element component given here is absorbed rather than nested. For a scoped slot, write the template with template().
events	Element's events to report besides those reported unasked, by name: events = "node_drop" reports input$<id>_node_drop. The component's are listed under "Shiny inputs", and by el_events(); a name it does not have is an error.
on	Handlers of your own, for an event not reported or to send something else: a named list of JS() functions, one per event – Element's, or a DOM event with Vue's modifiers ("keyup.enter"). Each is called with report and the event's arguments; report(name, value) sets input$<id>_<name>. See el_widget().
session	In el_steps(), deprecated: inside a module, wrap id in ns(), as for any Shiny input; a session given here namespaces id once more, with a warning. In update_el_steps(), the Shiny session, the current one by default, as for shiny::updateTextInput().

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
input$<id>_change	events = "change"	triggers when the active step changes

The same list as el_events("el_steps"), which says how an event's arguments travel.

Updating from the server

update_el_steps() changes the component from the server.

Every other argument of el_steps() that can change once it is drawn is an argument here too, under the same name. One left NULL stays as it is; NA returns it to Element's default.

update_el_steps() is called for its side effect and returns NULL invisibly.

Examples

```
# Basic usage
el_steps(
  id = "my_steps",
  steps = list(
    list(title = "Step 1"),
    list(title = "Step 2"),
    list(title = "Step 3")
  )
)

# With descriptions and icons
el_steps(
  id = "my_steps",
  active = 1,
  finish_status = "success",
  steps = list(
    list(
      title = "Step 1",
      description = "Complete registration",
      icon = "el-icon-edit"
    ),
    list(
      title = "Step 2",
      description = "Upload documents",
      icon = "el-icon-upload"
    ),
    list(title = "Step 3", description = "Finish", icon = "el-icon-picture")
  )
)
if (interactive()) {
  # inside a server function
  observeEvent(input$go, {
    update_el_steps(session, "wizard", active = 2)
  })
}
```

---

el_switch *Element Plus Switch*

Description

Creates an Element Plus switch with Vue instance.

Usage

```
el_switch(
  id = NULL,
  value = FALSE,
```

```
  disabled = FALSE,
  label = NULL,
  label_position = c("top", "left", "right"),
  label_width = NULL,
  label_suffix = NULL,
  required = FALSE,
  error = NULL,
  show_message = TRUE,
  inline_message = FALSE,
  width = NULL,
  active_text = NULL,
  inactive_text = NULL,
  active_color = NULL,
  inactive_color = NULL,
  active_value = TRUE,
  inactive_value = FALSE,
  name = NULL,
  validate_event = NULL,
  active_action_icon = NULL,
  active_icon = NULL,
  aria_label = NULL,
  before_change = NULL,
  border_color = NULL,
  inactive_action_icon = NULL,
  inactive_icon = NULL,
  inline_prompt = NULL,
  loading = NULL,
  size = NULL,
  tabindex = NULL,
  slots = NULL,
  on = NULL,
  session = NULL
)

update_el_switch(
  session = shiny::getDefaultReactiveDomain(),
  id,
  value = NULL,
  disabled = NULL,
  active_text = NULL,
  inactive_text = NULL,
  active_color = NULL,
  inactive_color = NULL,
  label = NULL,
  error = NULL,
  active_value = NULL,
  inactive_value = NULL,
  name = NULL,
```

```
  validate_event = NULL,
  active_action_icon = NULL,
  active_icon = NULL,
  aria_label = NULL,
  before_change = NULL,
  border_color = NULL,
  inactive_action_icon = NULL,
  inactive_icon = NULL,
  inline_prompt = NULL,
  loading = NULL,
  size = NULL,
  tabindex = NULL
)
```

Arguments

id	Switch ID. Auto-generated UUID if NULL.
value	Initial switch state. Default FALSE.
disabled	Whether the switch is disabled. Default FALSE.
label	A label shown with the component, as Shiny's inputs have: text or a tag. NULL, the default, shows none. It is the component's accessible name too – tied to it with for where the component has a native input that takes the id <id>-input, else with aria-labelledby.
label_position	Where the label sits, as el_form()'s label_position: "top" (the default, as Shiny's labels sit), or beside the component, its text aligned "left" or "right" – which shows once label_width gives the labels a common width.
label_width	Width of a label beside the component, as a CSS unit, so that several line up. Element's label-width.
label_suffix	Text after the label, such as ":". Element's label-suffix.
required	Draw Element's red asterisk before the label. It marks the field; it does not check it – shinyvalidate or el_form() does that.
error	An error message shown under the component in Element's style, the field framed in red. Element's error.
show_message, inline_message	Whether error's message is shown, and whether beside the component rather than under it. Element's show-message and inline-message.
width	Switch width in pixels (integer).
active_text	Text displayed when switch is on.
inactive_text	Text displayed when switch is off.
active_color	Background color when switch is on (e.g. "#409EFF"): Element Plus's CSS variable --el-switch-on-color, which replaced its deprecated active-color.
inactive_color	Background color when switch is off: the CSS variable --el-switch-off-color, which replaced the deprecated inactive-color.
active_value	Value reported to Shiny when switch is on. Default TRUE.

inactive_value
: Value reported to Shiny when switch is off. Default FALSE.

name
: Native name attribute of the inner checkbox.

validate_event
: Whether a change triggers form validation. Default TRUE.

active_action_icon
: Component of the icon displayed in action when in on state. Element Plus's active-action-icon (string / Component). An icon's name, such as "Search".

active_icon
: Component of the icon displayed when in on state, overrides active-text. Element Plus's active-icon (string / Component). An icon's name, such as "Search".

aria_label
: Same as aria-label in native input. Element Plus's aria-label (string).

before_change
: Before-change hook before the switch state changes. If false is returned or a Promise is returned and then is rejected, will stop switching. Element Plus's before-change (() => Promise<boolean> | boolean).

border_color
: Border color of the switch: the CSS variable --el-switch-border-color, which replaced Element Plus's deprecated border-color.

inactive_action_icon
: Component of the icon displayed in action when in off state. Element Plus's inactive-action-icon (string / Component). An icon's name, such as "Search".

inactive_icon
: Component of the icon displayed when in off state, overrides inactive-text. Element Plus's inactive-icon (string / Component). An icon's name, such as "Search".

inline_prompt
: Whether icon or text is displayed inside dot, only the first character will be rendered for text. Element Plus's inline-prompt (boolean).

loading
: Whether Switch is in loading state. Element Plus's loading (boolean).

size
: Size of Switch. Element Plus's size ('' | 'large' | 'default' | 'small').

tabindex
: Tabindex for input. Element Plus's tabindex (string / number).

slots
: Named list of Element slot contents, such as list(title = shiny::tags$b("Bold")). A shiny.element component given here is absorbed rather than nested. For a scoped slot, write the template with template().

on
: Handlers of your own, for an event not reported or to send something else: a named list of JS() functions, one per event – Element's, or a DOM event with Vue's modifiers ("keyup.enter"). Each is called with report and the event's arguments; report(name, value) sets input$<id>_<name>. See el_widget().

session
: In el_switch(), deprecated: inside a module, wrap id in ns(), as for any Shiny input; a session given here namespaces id once more, with a warning. In update_el_switch(), the Shiny session, the current one by default, as for shiny::updateTextInput().

Value

An htmltools tagList with a Vue-managed switch component.

Element methods

Callable with call_el():

- focus() – Focus the Switch component

Shiny inputs

Input	Reported	Value
`input$<id>`	unasked	`active_value` when on, `inactive_value` when off

The same list as `el_events("el_switch")`, which says how an event's arguments travel.

Updating from the server

Server-side update for `el_switch()`. Pass only the fields to change; `NULL` fields are excluded from the update message.

Every other argument of `el_switch()` that can change once it is drawn is an argument here too, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_switch()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_switch("sw1", value = TRUE)

if (interactive()) {
  library(shiny)
  library(shiny.element)
  ui <- el_page(
    el_switch("sw1", active_text = "On", inactive_text = "Off"),
    verbatimTextOutput("state")
  )
  server <- function(input, output, session) {
    output$state <- renderPrint(input$sw1)
  }
  shinyApp(ui, server)
}
if (interactive()) {
  # inside a server function
  observeEvent(input$go, {
    update_el_switch(session, "live", value = TRUE)
  })
}
```

---

el_table *Element Plus Table Component*

Description

Rows of data, with sorting, selection, fixed columns, cell templates and row actions that report to the server.

Usage

```
el_table(
  id = NULL,
  data = list(),
  columns = list(),
  selection = FALSE,
  rownames = NULL,
  border = FALSE,
  stripe = NULL,
  size = NULL,
  height = NULL,
  max_height = NULL,
  fit = NULL,
  show_header = NULL,
  highlight_current_row = NULL,
  current_row_key = NULL,
  row_key = NULL,
  empty_text = NULL,
  default_expand_all = NULL,
  expand_row_keys = NULL,
  default_sort = NULL,
  tooltip_effect = NULL,
  show_summary = NULL,
  sum_text = NULL,
  select_on_indeterminate = NULL,
  indent = NULL,
  lazy = NULL,
  tree_props = NULL,
  row_class_name = NULL,
  row_style = NULL,
  cell_class_name = NULL,
  cell_style = NULL,
  header_row_class_name = NULL,
  header_row_style = NULL,
  header_cell_class_name = NULL,
  header_cell_style = NULL,
  span_method = NULL,
  summary_method = NULL,
  load = NULL,
  width = NULL,
  slots = NULL,
  loading = FALSE,
  loading_options = NULL,
  allow_drag_last_column = NULL,
  append_filter_panel_to = NULL,
  flexible = NULL,
  native_scrollbar = NULL,
  preserve_expanded_content = NULL,
```

```
  row_expandable = NULL,
  scrollbar_always_on = NULL,
  scrollbar_tabindex = NULL,
  show_overflow_tooltip = NULL,
  table_layout = NULL,
  tooltip_formatter = NULL,
  tooltip_options = NULL,
  session = NULL,
  events = NULL,
  on = NULL
)

update_el_table(
  session = shiny::getDefaultReactiveDomain(),
  id,
  data = NULL,
  insert = NULL,
  replace = NULL,
  delete = NULL,
  at = NULL,
  columns = NULL,
  border = NULL,
  selection = NULL,
  loading = NULL,
  loading_options = NULL,
  stripe = NULL,
  size = NULL,
  height = NULL,
  max_height = NULL,
  fit = NULL,
  show_header = NULL,
  highlight_current_row = NULL,
  current_row_key = NULL,
  row_key = NULL,
  empty_text = NULL,
  expand_row_keys = NULL,
  tooltip_effect = NULL,
  show_summary = NULL,
  sum_text = NULL,
  select_on_indeterminate = NULL,
  indent = NULL,
  lazy = NULL,
  tree_props = NULL,
  row_class_name = NULL,
  row_style = NULL,
  cell_class_name = NULL,
  cell_style = NULL,
  header_row_class_name = NULL,
```

```
  header_row_style = NULL,
  header_cell_class_name = NULL,
  header_cell_style = NULL,
  span_method = NULL,
  summary_method = NULL,
  load = NULL,
  allow_drag_last_column = NULL,
  append_filter_panel_to = NULL,
  flexible = NULL,
  native_scrollbar = NULL,
  preserve_expanded_content = NULL,
  row_expandable = NULL,
  scrollbar_always_on = NULL,
  scrollbar_tabindex = NULL,
  show_overflow_tooltip = NULL,
  table_layout = NULL,
  tooltip_formatter = NULL,
  tooltip_options = NULL
)
```

Arguments

id	The table's id. Leave it out: in an app the output's id is the table's. A table given an id in the UI still reports its inputs, but its data is then fixed in the page and `input$<id>_selection_change` gets its rows back through JSON rather than as R subsets them.
data	A data.frame, or a list of rows (each a named list). A data.frame is converted to rows automatically and its column names are sanitised (`.` becomes `_`) so `el-table`'s dotted `prop` lookup works. A data.frame inside the rows, or in a list column, is rows too. A vector of one element travels as a single value, as in Shiny; wrap it in `I()` to keep it an array: `tags = I("red")`.
columns	The columns, each an `el_table_column()` – or a list of the same fields, `list(prop =, label =, width =)`. Inferred from `data` when omitted. Beyond Element's column attributes, a column may carry: • `type` – `"index"` for row numbers, `"expand"` for a row that opens to show its `cell`, or `"selection"`. • `cell` – markup for each cell, rendered once per row. `scope.row`, `scope.column` and `scope.$index` are in reach, and raw Element tags (`el$tag()`, `el$button()`) work: `el$tag(":type" = "scope.row.ok ? 'success' : 'danger'", "{{ scope.row.status }}")`. A column without a `prop` – a column of buttons – is fine. See "Row actions" below. • `filter_icon` – the filter's icon, by name. • `header_html` – markup for the header cell, a string or htmltools tags, inserted unescaped, so pass only what you control. • `header` – a template for the header cell, as `cell` is for the others: components and all, `scope.column` and `scope.$index` in reach. `el$input(size = "small", ...)` puts a search box there.

- children – the columns under a group header, as Element nests el-table-column: list(label = "Address", children = list(...)), as deep as you nest them; each child column takes cell, header and header_html as a top-level one does.

selection — Enable row selection

rownames — Whether to show a data.frame's row names as the first column. NULL (the default) shows them when they carry something – mtcars' car names – and leaves out automatic ones, which only count rows.

border — Draw vertical borders between columns and a frame around the table. Default FALSE, as in Element.

stripe — Whether rows alternate background colour.

size — Size: "large", "default" or "small"; NULL follows the form or the page.

height — Table height. Fixes the header and scrolls the body.

max_height — Maximum table height, beyond which the body scrolls.

fit — Whether column widths stretch to fill the table. Default TRUE.

show_header — Whether the header row is shown. Default TRUE.

highlight_current_row — Whether the clicked row stays highlighted; the row clicked arrives as input$<id>_current_change.

current_row_key — Key of the row highlighted at start. Needs row_key.

row_key — Column whose value identifies a row. Needed for tree data and reserved selection.

empty_text — Text shown when there are no rows. Default "No Data".

default_expand_all — Whether expandable rows start expanded.

expand_row_keys — Keys of the rows that start expanded. Needs row_key.

default_sort — Initial sort, as list(prop =, order =).

tooltip_effect — Theme of overflow tooltips: "dark" (default) or "light".

show_summary — Whether to add a summary row at the bottom.

sum_text — Label of the summary row's first cell. Default "Sum".

select_on_indeterminate — What the header checkbox does when only some rows are selected. Default TRUE.

indent — Horizontal indent between tree levels, in pixels. Default 16.

lazy — Whether child rows of tree data are loaded on demand – from the server, unless load is given.

tree_props — Field names for tree data, as list(children =, hasChildren =).

row_class_name — Class name for every row, or a JS function returning one.

row_style — Inline style for every row, or a JS function returning one.

cell_class_name — Class name for every cell, or a JS function returning one.

`cell_style` Inline style for every cell, or a JS function returning one.

`header_row_class_name`
Class name for the header row, or a JS function returning one.

`header_row_style`
Inline style for the header row, or a JS function returning one.

`header_cell_class_name`
Class name for header cells, or a JS function returning one.

`header_cell_style`
Inline style for header cells, or a JS function returning one.

`span_method` `JS()` function deciding row/column spans for merged cells.

`summary_method` `JS()` function returning the summary row's cells.

`load` `JS()` function loading child rows in the browser instead of from the server. Needs `lazy = TRUE`.

`width` Component width, as a CSS unit. Replaces the table's default `width: 100%`. For a fixed header use `height` instead.

`slots` Named list of Element slot contents, such as `list(empty = shiny::tags$b("Nothing yet"))`. A shiny.element component given here is absorbed rather than nested. For a scoped slot, write the template with `template()`.

`loading` Whether to cover the table with Element's loading mask, as its `v-loading` does. `update_el_table()` turns it on and off around slow work.

`loading_options`
How the mask looks, as a list with the names of `el_loading()`'s arguments – `text`, `spinner`, `svg`, `svg_view_box`, `background`, `custom_class` – Element's `element-loading-*` attributes.

`allow_drag_last_column`
Whether to allow drag the last column. Element Plus's `allow-drag-last-column` (boolean).

`append_filter_panel_to`
Which element the filter panels appends to. Element Plus's `append-filter-panel-to` (string).

`flexible` Ensure main axis minimum-size doesn't follow the content. Element Plus's `flexible` (boolean).

`native_scrollbar`
Whether to use native scrollbars. Element Plus's `native-scrollbar` (boolean).

`preserve_expanded_content`
Whether to preserve expanded row content in DOM when collapsed. Element Plus's `preserve-expanded-content` (boolean).

`row_expandable` Enable expandable rows, works when the table has a column type="expand". Element Plus's `row-expandable` ((row: any, index: number) => boolean).

`scrollbar_always_on`
Always show scrollbar. Element Plus's `scrollbar-always-on` (boolean).

`scrollbar_tabindex`
Body scrollbar's wrap container tabindex. Element Plus's `scrollbar-tabindex` (string / number).

show_overflow_tooltip
: Whether to hide extra content and show them in a tooltip when hovering on the cell.It will affect all the table columns, refer to table tooltip-options. Element Plus's `show-overflow-tooltip` (boolean).

table_layout
: Sets the algorithm used to lay out table cells, rows, and columns. Element Plus's `table-layout` ('fixed' | 'auto').

tooltip_formatter
: Customize tooltip content when using `show-overflow-tooltip`. Element Plus's `tooltip-formatter` (Function). Give it as `JS()`.

tooltip_options
: The options for the overflow tooltip, see the following tooltip component. Element Plus's `tooltip-options` (object).

session
: In `el_table()`, deprecated: inside a module, wrap `id` in `ns()`, as for any Shiny input; a session given here namespaces `id` once more, with a warning. In `update_el_table()`, the Shiny session, the current one by default, as for `shiny::updateTextInput()`.

events
: Element's events to report besides those reported unasked, by name: `events = "node_drop"` reports `input$<id>_node_drop`. The component's are listed under "Shiny inputs", and by `el_events()`; a name it does not have is an error.

on
: Handlers of your own, for an event not reported or to send something else: a named list of `JS()` functions, one per event – Element's, or a DOM event with Vue's modifiers (`"keyup.enter"`). Each is called with `report` and the event's arguments; `report(name, value)` sets `input$<id>_<name>`. See `el_widget()`.

insert
: Rows to insert – a data.frame, or a list of rows – before row `at`, or at the end.

replace
: Rows to put in place of rows `at`, one for each.

delete
: Numbers of the rows to delete.

at
: Where `insert` goes, one row number; which rows `replace` replaces. Row numbers count the rows the table shows, as `input$<id>_selection_rows` does.

Details

In a Shiny app a table is an output, as DT's and reactable's are: the page holds `el_table_output()`, the server renders `el_table()` into it with `render_el_table()`, and the output's id names the table's inputs. Without Shiny – R Markdown, Quarto, a pkgdown page – `el_table()` is placed as it is, with no id.

`el_table()` returns the table's specification, drawn when it is placed (as an htmlwidget is).

Value

An `el_table` object, drawn as a table wherever UI goes.

Shiny inputs

Input	Reported	Value
`input$<id>_selection_rows`	unasked	the selected row numbers

input$<id>_selection_change	unasked	the selected rows, data[rows, , drop = FALSE]
input$<id>_cell_edit	unasked	an edited cell: list(row, column, value, old)
input$<id>_load	unasked	with lazy = TRUE, a row asking for its children; a
input$<id>_select	events = "select"	list(rows, row_index): the selected row numb
input$<id>_select_all	events = "select_all"	list(rows), the selected row numbers
input$<id>_cell_mouse_enter	events = "cell_mouse_enter"	list(row_index, row, column, value)
input$<id>_cell_mouse_leave	events = "cell_mouse_leave"	list(row_index, row, column, value)
input$<id>_cell_click	events = "cell_click"	list(row_index, row, column, value)
input$<id>_cell_dblclick	events = "cell_dblclick"	list(row_index, row, column, value)
input$<id>_cell_contextmenu	events = "cell_contextmenu"	list(row_index, row, column, value)
input$<id>_row_click	events = "row_click"	list(row_index, row, column)
input$<id>_row_contextmenu	events = "row_contextmenu"	list(row_index, row, column)
input$<id>_row_dblclick	events = "row_dblclick"	list(row_index, row, column)
input$<id>_header_click	events = "header_click"	list(column, label)
input$<id>_header_contextmenu	events = "header_contextmenu"	list(column, label)
input$<id>_sort_change	unasked	list(column, order)
input$<id>_filter_change	unasked	the filters, list(<column key> = values)
input$<id>_current_change	unasked	list(row_index, row, previous_index)
input$<id>_header_dragend	events = "header_dragend"	list(column, width, previous_width)
input$<id>_expand_change	unasked	list(row_index, expanded)
input$<id>_scroll	events = "scroll"	list(scroll_left, scroll_top), at most ever

The same list as el_events("el_table"), which says how an event's arguments travel.

Rendered into el_table_output("tbl"), the inputs are input$tbl_.... input$tbl itself is not used: Element's table has no value of its own (no v-model), and the name stays free.

An edit is shown at once and applied to the server's copy of the data, so el_table_data() and input$tbl_selection_change see it; an observer of input$tbl_cell_edit saves it, or refuses it by putting the old value back: update_el_table(session, "tbl", replace = row, at = inputtbl_cell_editrow).

Row numbers count rows of the data the table shows, whatever the user's sort – the data el_table_data() returns. Rendering the same data again keeps the user's ticks, sort and open rows; new data clears the selection, as Element does, unless rows carry a key: row_key and a selection column with reserve_selection = TRUE keep the ticked rows that are still there. Rows inserted, replaced or deleted with update_el_table() leave the other rows' ticks alone.

With tree data, lazy = TRUE and no load of your own, the server loads a row's children when it is opened: input$<id>_load asks, with key (the row's row_key field), row and request; answer with el_load_children(). A row with children to load carries hasChildren = TRUE.

Row actions

A button in a cell reports back with rowAction():

```
list(label = "", cell = el$button(size = "small",
  "@click" = "rowAction('edit', scope)", "Edit"))
```

sets input$<id>_edit to list(row_index =, row =) – the 1-based row number, to index your own data with, and the row as the table holds it. It is an event input: clicking the same row twice reports twice.

Element methods

Callable with `call_el()`:

- `clearFilter()` – Clear filters of the columns whose columnKey are passed in. If no params, clear all filters
- `clearSelection()` – Used in multiple selection Table, clear user selection
- `clearSort()` – Clear sorting, restore data to the original order
- `doLayout()` – Refresh the layout of Table. When the visibility of Table changes, you may need to call this method to get...
- `setCurrentRow()` – Used in single selection Table, set a certain row selected. If called without any parameter, it will clear...
- `sort()` – Sort Table manually. Property prop is used to set sort column, property order is used to set sort order
- `toggleAllSelection()` – Used in multiple selection Table, toggle the selected state of all rows
- `toggleRowExpansion()` – Used in expandable Table or tree Table, toggle if a certain row is expanded. With the second parameter,...
- `toggleRowSelection()` – Used in multiple selection Table, toggle if a certain row is selected. With the second parameter, you can...

Updating from the server

Changes a table from the server, as `shiny::updateSelectInput()` does a select: every argument of `el_table()` that can change once the table is drawn, under the same name. One left `NULL` stays as it is; `NA` returns a prop to Element's default. `rownames`, `slots`, `width`, `events` and the `default_*` arguments, which are read only when the table is created, are not here. A table in `el_table_output()` is reached by the output's id; rendering it again with `render_el_table()` does the same for any argument.

Rows. `data` replaces them all; `insert`, `replace` and `delete` change a few and send only those, one of them per call. The server's copy of the data (`el_table_data()`) is changed as R would change it – `rbind()` to insert, `data[at, ] <- replace`, `data[-delete, ]` – and the browser gets the rows as they then stand, so both hold the same data. Rows not touched keep their ticks and open state; the selection's row numbers are reported again, renumbered.

Data that comes from reactive expressions is best rendered: the table is patched in place either way, and one render is one source of the data. An update suits a change an observer makes – a row the user added, edited or deleted. A render after an update leaves the update's rows in place unless the render's own data changed.

A column's `cell` template is part of the table's markup, made when the table is. New columns given here keep the template of the column with the same `prop` (or label) and may drop it, but cannot bring a template the table was not created with.

`update_el_table()` is called for its side effect and returns `NULL` invisibly.

Examples

```
# A data.frame is enough -- columns are inferred
el_table(data = head(iris, 3))

# Explicit columns
el_table(
  data = data.frame(name = c("A", "B"), value = c(1, 2)),
  columns = list(
    list(prop = "name", label = "Name"),
    list(prop = "value", label = "Value", width = "100")
  )
)

# A Shiny app: the table is an output, its inputs named after it
if (interactive()) {
  library(shiny)
  library(shiny.element)
  ui <- el_page(
    el_input_number("n", value = 5, min = 1),
    el_table_output("flowers"),
    verbatimTextOutput("picked")
  )
  server <- function(input, output, session) {
    output$flowers <- render_el_table(
      el_table(
        data = head(iris, input$n),
        selection = TRUE,
        events = "row_dblclick"
      )
    )
    # the ticked rows, as R subsets them
    output$picked <- renderPrint(input$flowers_selection_change)
    observeEvent(input$flowers_row_dblclick, {
      el_message(message = paste("Row", input$flowers_row_dblclick$row_index))
    })
  }
  shinyApp(ui, server)
}
if (interactive()) {
  # inside a server function
  observeEvent(input$go, {
    update_el_table(session, "tbl", data = head(mtcars, 10))
  })
  # any other argument of el_table()
  update_el_table(session, "tbl", stripe = TRUE, table_layout = "auto")
  # back to Element's default
  update_el_table(session, "tbl", stripe = NA)
  # a few rows, leaving the rest as they are
  update_el_table(session, "tbl", insert = head(mtcars, 1), at = 1)
  update_el_table(session, "tbl", replace = mtcars[3, ], at = 3)
  update_el_table(session, "tbl", delete = c(2, 5))
}
```

el_table_column *A column of* el_table()

Description

Element Plus's el-table-column, for el_table(columns =) and update_el_table(). A column with columns of its own in ... is a group header, nested as deep as you like.

Usage

```
el_table_column(
  prop = NULL,
  label = NULL,
  ...,
  cell = NULL,
  editable = NULL,
  editor = NULL,
  header = NULL,
  header_html = NULL,
  filter_icon = NULL,
  type = NULL,
  index = NULL,
  column_key = NULL,
  width = NULL,
  min_width = NULL,
  fixed = NULL,
  render_header = NULL,
  sortable = NULL,
  sort_method = NULL,
  sort_by = NULL,
  sort_orders = NULL,
  resizable = NULL,
  formatter = NULL,
  show_overflow_tooltip = NULL,
  align = NULL,
  header_align = NULL,
  class_name = NULL,
  label_class_name = NULL,
  selectable = NULL,
  reserve_selection = NULL,
  filters = NULL,
  filter_placement = NULL,
  filter_class_name = NULL,
  filter_multiple = NULL,
  filter_method = NULL,
  filtered_value = NULL,
  tooltip_formatter = NULL
```

```
)
```

Arguments

prop	The field of each row the column shows.
label	The column's title.
...	Columns under this one, each an `el_table_column()`: the column is then a group header, `el_table_column(label = "Info", el_table_column( "name", "Name"), ...)`.
cell	A template for each cell, drawn once per row: tags or a string, with `scope.row`, `scope.column` and `scope.$index` in reach and raw Element tags (`el$tag()`) working. A button in it reports with `rowAction('edit', scope)`; see `el_table()`.
editable	Whether the column's cells are edited in place, and with what: `TRUE` or `"input"` for text, `"number"`, `"select"` or `"date"` – Element's input, input-number, select and date picker. A double click opens the editor; Enter or leaving it commits, Escape abandons, Tab commits and moves to the next editable cell. Each edit is shown at once, applied to the server's copy of the data (`el_table_data()`) and reported as `input$<id>_cell_edit`; see `el_table()`. Not with `cell`.
editor	The editor's props, under Element's names in snake_case: `list(min = 0, precision = 2)` for a number, `list(choices = c("a", "b"))` for a select, `list(placeholder = "...")`.
header	A template for the header cell, as `cell` is for the others: a search box, a button.
header_html	Markup for the header cell, inserted as it is: pass only what you control.
filter_icon	The filter's icon, by name.
type	`"selection"` (a checkbox), `"index"` (row numbers) or `"expand"` (an arrow opening the row to its `cell`); Element's default is `"default"`.
index	For `type = "index"`, a number to start from or a `JS()` function of the row's index.
column_key	The column's key, which `filter-change` names.
width, min_width	The column's width; `min_width` columns share what `width` columns leave, in proportion.
fixed	Fix the column at the `"left"` (or `TRUE`) or the `"right"`.
render_header	A `JS()` render function for the header.
sortable	Whether the column sorts; `"custom"` sorts on the server, through the table's `sort-change` input.
sort_method	A `JS()` comparison function.
sort_by	The field, fields or `JS()` function to sort by.
sort_orders	The orders a click cycles through: `list("ascending", "descending", NULL)`.
resizable	Whether the column can be resized, with `border = TRUE`.
formatter	A `JS()` function formatting the cell.
show_overflow_tooltip	Hide overflowing content behind a tooltip; `TRUE`, or the tooltip's options.

`align`, `header_align`
: Alignment of the cells and of the header: `"left"`, `"center"` or `"right"`.

`class_name`, `label_class_name`
: Class of the cells and of the header.

`selectable`
: For `type = "selection"`, a `JS()` function deciding whether a row can be ticked.

`reserve_selection`
: For `type = "selection"`, keep ticks when the data changes; needs the table's `row_key`.

`filters`
: The filter's choices: `list(list(text =, value =), ...)`.

`filter_placement`
: Where the filter's dropdown opens.

`filter_class_name`
: Class of the filter's dropdown.

`filter_multiple`
: Whether several filter choices can be ticked.

`filter_method`
: A `JS()` function `(value, row, column)` keeping a row.

`filtered_value`
: The filter's ticked values.

`tooltip_formatter`
: A `JS()` function of `{row, column, cellValue}` for the overflow tooltip's content.

Value

A column, for `el_table(columns =)`.

See Also

Other items: `el_anchor_link()`, `el_breadcrumb_item()`, `el_carousel_item()`, `el_collapse_item()`, `el_descriptions_item()`, `el_dropdown_item()`, `el_menu_item()`, `el_option()`, `el_skeleton_item()`, `el_step()`, `el_tab_pane()`, `el_table_v2_column()`, `el_timeline_item()`, `el_tour_step()`

Examples

```
el_table(
  "t",
  data = data.frame(date = "2016-05-03", name = "Tom", city = "LA"),
  columns = list(
    el_table_column("date", "Date", width = 150, sortable = TRUE),
    el_table_column(
      label = "Delivery Info",
      el_table_column("name", "Name"),
      el_table_column("city", "City")
    )
  )
)
```

`el_table_data` *The data a table shows*

Description

What the browser holds, as R: the data last rendered with `render_el_table()` or sent with `update_el_table()`, with every row `update_el_table()` has since inserted, replaced or deleted. A reactive read: an observer or output reading it runs again when the data changes. Row numbers in the table's inputs – `input$<id>_selection_rows`, a row event's `row_index` – index it.

Usage

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

Arguments

`session`	The Shiny session, the current one by default.
`id`	The table's id, the output's.

Value

The data, as given (a data.frame or a list of rows); `NULL` for a table the server has not rendered or updated.

See Also

`render_el_table()`, `update_el_table()`.

Examples

```
if (interactive()) {
  library(shiny)
  ui <- el_page(el_table_output("cars"), verbatimTextOutput("n"))
  server <- function(input, output, session) {
    output$cars <- render_el_table(el_table(data = head(mtcars)))
    output$n <- renderText(nrow(el_table_data(id = "cars")))
  }
  shinyApp(ui, server)
}
```

`el_table_output` *An Element Plus table as a Shiny output*

Description

The table of a Shiny app, as DT's `DTOutput()` and `renderDT()`: the page holds `el_table_output()`, the server renders `el_table()` into it with its data. Rendering again with new data updates the table in place – the user's sort, ticks and open rows stay (see `render_vue()`).

Usage

```
el_table_output(outputId, width = "100%", loading = TRUE)

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

Arguments

`outputId`	The output's id.
`width`	The table's width, as a CSS unit.
`loading`	Whether Element's loading mask covers the table while Shiny recalculates it, in place of Shiny fading the output – and of the spinner `shiny::useBusyIndicators()` would draw over it.
`expr`	An expression returning `el_table()`, given no `id`.
`env, quoted`	As for `shiny::renderUI()`.

Details

The output id names the table's inputs:

- `input$<id>_selection_rows` – the selected row numbers, integers; `NULL` with none.
- `input$<id>_selection_change` – the selected rows, `data[rows, , drop = FALSE]` of the data shown: its columns, types and row names.
- `input$<id>_current_change`, `_sort_change`, `_filter_change`, `_expand_change` – reported by every table; any other of Element's events with `el_table(events =)`: see `el_events()`.

`update_el_table()` and `call_el()` reach the table by the output id; `el_table_data()` reads the data it shows.

The first render sends the table; a render after it whose columns, templates and options are unchanged sends only the data that changed, as JSON – as Shiny's own outputs send values rather than markup. With `shiny::bindCache()` the table as rendered is cached, the same for every session, and each session's page still gets only what changed for it.

Value

`el_table_output()`, a tag; `render_el_table()`, a render function.

See Also

el_table(), el_events().

Examples

```
if (interactive()) {
  library(shiny)
  ui <- el_page(el_table_output("cars"), verbatimTextOutput("picked"))
  server <- function(input, output, session) {
    output$cars <- render_el_table(
      el_table(data = head(mtcars), selection = TRUE)
    )
    output$picked <- renderPrint(input$cars_selection_change)
  }
  shinyApp(ui, server)
}
```

---

el_table_row *Name a table row, an uploaded file or a tree node for a method*

Description

Element's table methods take the row object itself – `toggleRowSelection(row)`, `setCurrentRow(row)`, `toggleRowExpansion(row)` – and compare it by identity, so a copy sent from R would match nothing. Likewise the upload's `abort(file)` and `handleRemove(file)`, and a virtualized tree's `expandNode(node)` and `collapseNode(node)`. These stand for the object instead, and the page puts the real one in its place before the method runs.

Usage

```
el_table_row(index)

el_upload_file(name)

el_tree_node(key)
```

Arguments

index	A row's number, 1-based, as a table's `input$<id>_selection_rows` reports them.
name	A file's name, as it shows in the upload's list.
key	A node's key: the field `node_key` names, or the tree's `props$value`.

Value

A reference, for call_el()'s `args`.

Examples

```
if (interactive()) {
  # inside a server function: select the third row, then make it current
  call_el(session, "tbl", "toggleRowSelection", list(el_table_row(3), TRUE))
  call_el(session, "tbl", "setCurrentRow", list(el_table_row(3)))
  # stop one file
  call_el(session, "docs", "abort", list(el_upload_file("big.csv")))
  # open a node of a virtualized tree
  call_el(session, "files", "expandNode", list(el_tree_node("src")))
}
```

---

el_table_v2 *Element Plus Virtualized Table*

Description

A table drawing only the rows and columns in view, for very large data.

Usage

```
el_table_v2(
  id = NULL,
  cache = NULL,
  estimated_row_height = NULL,
  header_class = NULL,
  header_props = NULL,
  header_cell_props = NULL,
  header_height = NULL,
  footer_height = NULL,
  row_class = NULL,
  row_key = NULL,
  row_props = NULL,
  row_height = NULL,
  row_event_handlers = NULL,
  cell_props = NULL,
  columns = NULL,
  data = NULL,
  data_getter = NULL,
  fixed_data = NULL,
  expand_column_key = NULL,
  expanded_row_keys = NULL,
  default_expanded_row_keys = NULL,
  fixed = NULL,
  table_v2_width = NULL,
  height = NULL,
  max_height = NULL,
  indent_size = NULL,
```

```
  h_scrollbar_size = NULL,
  v_scrollbar_size = NULL,
  scrollbar_always_on = NULL,
  sort_by = NULL,
  sort_state = NULL,
  width = NULL,
  slots = NULL,
  methods = NULL,
  auto_resize = FALSE,
  events = NULL,
  on = NULL
)

update_el_table_v2(
  session = shiny::getDefaultReactiveDomain(),
  id,
  data = NULL,
  columns = NULL,
  sort_by = NULL,
  expanded_row_keys = NULL,
  cache = NULL,
  estimated_row_height = NULL,
  header_class = NULL,
  header_props = NULL,
  header_cell_props = NULL,
  header_height = NULL,
  footer_height = NULL,
  row_class = NULL,
  row_key = NULL,
  row_props = NULL,
  row_height = NULL,
  row_event_handlers = NULL,
  cell_props = NULL,
  data_getter = NULL,
  fixed_data = NULL,
  expand_column_key = NULL,
  fixed = NULL,
  table_v2_width = NULL,
  height = NULL,
  max_height = NULL,
  indent_size = NULL,
  h_scrollbar_size = NULL,
  v_scrollbar_size = NULL,
  scrollbar_always_on = NULL,
  sort_state = NULL
)
```

Arguments

id	Component ID. Auto-generated if NULL.
cache	Number of rows rendered in advance to boost the performance. Element Plus's cache.
estimated_row_height	The estimated row height for rendering dynamic height rows. Element Plus's estimated-row-height.
header_class	Customized class name passed to header wrapper. Element Plus's header-class. Give it as JS().
header_props	Customized props name passed to header component. Element Plus's header-props. Give it as JS().
header_cell_props	Customized props name passed to header cell component. Element Plus's header-cell-props. Give it as JS().
header_height	The height of the header is set by height. If given an array, it renders header rows equal to its length. Element Plus's header-height.
footer_height	The height of the footer element, when provided, will be part to the calculation of the table's height. Element Plus's footer-height.
row_class	Customized class name passed to row wrapper. Element Plus's row-class. Give it as JS().
row_key	The key of each row, if not provided, will be the index of the row. Element Plus's row-key.
row_props	Customized props name passed to row component. Element Plus's row-props. Give it as JS().
row_height	The height of each row, used for calculating the total height of the table. Element Plus's row-height.
row_event_handlers	A collection of handlers attached to each row. Element Plus's row-event-handlers.
cell_props	Extra props passed to each cell (except header cells). Element Plus's cell-props. Give it as JS().
columns	The columns, each an el_table_v2_column() – or a list(key =, dataKey =, title =, width =) with Element Plus's other column fields. NULL makes one per variable of a data.frame data, 150 pixels wide.
data	The rows: a data.frame, or a list of rows. Element Plus's data. A vector of one element travels as a single value, as in Shiny; wrap it in I() to keep it an array: tags = I("red").
data_getter	A method to customize data fetch from the data source. Element Plus's data-getter. Give it as JS().
fixed_data	Data for rendering rows above the main content and below the header. Element Plus's fixed-data.
expand_column_key	The column key indicates which row is expandable. Element Plus's expand-column-key.

`expanded_row_keys`
: An array of keys for expanded rows, can be used with `v-model`. Element Plus's `expanded-row-keys`.

`default_expanded_row_keys`
: An array of keys for default expanded rows, **NON REACTIVE**. Element Plus's `default-expanded-row-keys`.

`fixed`
: Flag indicates the table column's width to be fixed or flexible. Element Plus's `fixed`.

`table_v2_width`
: Width of the table, in pixels: Element Plus's `width`, which it needs as a number. Default `700`.

`height`
: Height of the table, in pixels. Default `400`.

`max_height`
: Maximum height of the table. Element Plus's `max-height`.

`indent_size`
: Horizontal indentation of tree table. Element Plus's `indent-size`.

`h_scrollbar_size`
: Indicates the horizontal scrollbar's size for the table, used to prevent the horizontal and vertical scrollbar to collapse. Element Plus's `h-scrollbar-size`.

`v_scrollbar_size`
: Indicates the vertical scrollbar's size for the table, used to prevent the horizontal and vertical scrollbar to collapse. Element Plus's `v-scrollbar-size`.

`scrollbar_always_on`
: If true, the scrollbar will always be shown instead of when mouse is placed above the table. Element Plus's `scrollbar-always-on`.

`sort_by`
: Sort indicator. Element Plus's `sort-by`.

`sort_state`
: Multiple sort indicator. Element Plus's `sort-state`.

`width`
: Component width, as a CSS unit.

`slots`
: Named list of Element slot contents: `cell`, `header`, `header-cell`, `row`, `footer`, `empty`, `overlay`. A scoped slot is written with `template()`.

`methods`
: Named list of `JS()` functions that the slot templates can call by name. Element Plus's own examples build cells, header rows and rows in JavaScript; such a function, given the slot's scope, can do the same and return the cells to draw (`Vue.h()`, `Vue.cloneVNode()`). Each is drawn with `<component :is="cell" />`.

`auto_resize`
: `TRUE` to size the table to its container, as Element Plus's `el-auto-resizer` does: `table_v2_width` and `height` are then ignored, and the container needs a height of its own.

`events`
: Element's events to report besides those reported unasked, by name: `events = "node_drop"` reports `input$<id>_node_drop`. The component's are listed under "Shiny inputs", and by `el_events()`; a name it does not have is an error.

`on`
: Handlers of your own, for an event not reported or to send something else: a named list of `JS()` functions, one per event – Element's, or a DOM event with Vue's modifiers (`"keyup.enter"`). Each is called with `report` and the event's arguments; `report(name, value)` sets `input$<id>_<name>`. See `el_widget()`.

`session`
: Shiny session; the current one by default, as for `shiny::updateTextInput()`.

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
input$<id>_column_sort	unasked	Invoked when column sorted
input$<id>_expanded_rows_change	events = "expanded_rows_change"	Invoked when expanded rows changed
input$<id>_end_reached	unasked	Invoked when the end of the table is reached
input$<id>_scroll	events = "scroll"	at most every 200 ms
input$<id>_rows_rendered	events = "rows_rendered"	at most every 200 ms
input$<id>_row_expand	events = "row_expand"	Invoked when expand/collapse the tree node

The same list as `el_events("el_table_v2")`, which says how an event's arguments travel.

Element methods

Callable with `call_el()`: `scrollTo()`, `scrollToLeft()`, `scrollToTop()`, `scrollToRow()`.

Updating from the server

Server-side update for `el_table_v2()`: every argument that can change once the table is drawn, under the same name. One left `NULL` stays as it is; `NA` returns a prop to Element's default. Rows are given as for `el_table_v2()`, a data.frame or a list of rows; a data.frame without `columns` keeps the table's columns. `methods`, `slots`, `width`, `auto_resize` and `default_expanded_row_keys`, which Element reads only when the table is created, are not here.

`update_el_table_v2()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_table_v2(
  "big",
  data = data.frame(x = 1:10000, y = rnorm(10000)),
  width = 600,
  height = 400
)
if (interactive()) {
  # inside a server function
  observeEvent(input$filter_on, {
    update_el_table_v2(id = "big", data = subset(big, keep))
  })
  # any other argument of el_table_v2()
  update_el_table_v2(id = "big", sort_state = list(id = "desc"))
}
```

`el_table_v2_column` *A column of* `el_table_v2()`

Description

A column definition for `el_table_v2(columns =)` and `update_el_table_v2()`, under Element Plus's own field names.

Usage

```
el_table_v2_column(
  key,
  title = NULL,
  data_key = key,
  width = 150,
  min_width = NULL,
  max_width = NULL,
  align = NULL,
  fixed = NULL,
  flex_grow = NULL,
  flex_shrink = NULL,
  sortable = NULL,
  hidden = NULL,
  class = NULL,
  header_class = NULL,
  style = NULL,
  cell_renderer = NULL,
  header_cell_renderer = NULL
)
```

Arguments

`key`	The column's unique key.
`title`	The text of its header cell.
`data_key`	The field of each row it shows. Defaults to `key`.
`width`	The column's width, in pixels.
`min_width`, `max_width`	Its bounds, in pixels.
`align`	Alignment of its cells: `"left"`, `"center"` or `"right"`.
`fixed`	Fix it at the `"left"` (or `TRUE`) or the `"right"`.
`flex_grow`, `flex_shrink`	Flex grow and shrink, in a table that is not `fixed`.
`sortable`	Whether it sorts; the sort is reported, the rows are yours to reorder.
`hidden`	Whether it is hidden.

class, header_class
: Class of its cells and of its header cell.

style
: Style of its cells, a list of CSS properties.

cell_renderer, header_cell_renderer
: JS() functions drawing a cell and the header cell, returning Vue's h().

Value

A column, for el_table_v2(columns =).

See Also

Other items: el_anchor_link(), el_breadcrumb_item(), el_carousel_item(), el_collapse_item(), el_descriptions_item(), el_dropdown_item(), el_menu_item(), el_option(), el_skeleton_item(), el_step(), el_tab_pane(), el_table_column(), el_timeline_item(), el_tour_step()

Examples

```
el_table_v2(
  "tv",
  data = data.frame(id = 1:3, name = c("a", "b", "c")),
  columns = list(
    el_table_v2_column("id", "Id", width = 80, fixed = "left"),
    el_table_v2_column("name", "Name", width = 200, sortable = TRUE)
  )
)
```

---

el_tabs *Element Plus Tabs*

Description

A tabbed panel.

Usage

```
el_tabs(
  id = NULL,
  tabs = list(),
  selected = NULL,
  type = NULL,
  tab_position = "top",
  closable = FALSE,
  addable = FALSE,
  editable = FALSE,
  stretch = FALSE,
  before_leave = NULL,
  add_icon = NULL,
```

```
  value = NULL,
  events = NULL,
  session = NULL
)

update_el_tabs(
  session = shiny::getDefaultReactiveDomain(),
  id,
  selected = NULL,
  type = NULL,
  tab_position = NULL,
  closable = NULL,
  addable = NULL,
  editable = NULL,
  stretch = NULL,
  value = NULL
)
```

Arguments

id	Tabs ID. Auto-generated UUID if NULL.
tabs	A list of tabs, each an el_tab_pane() – or a named list with the same fields: **name** Unique tab identifier (string). Required. **label** Tab label. Required. **content** Tab body. Any tag or tagList, including this package's own components. **disabled** Whether the tab can be selected. Default FALSE. **closable** Whether this one tab can be closed, when closable is off for the rest. **lazy** Render the content only when the tab is first selected. Its components do not exist, and report nothing, until then.
selected, value	Name of the initially selected tab; the first by default. selected is Shiny's name, as in shiny::tabsetPanel(), value Element's (its v-model); give either.
type	NULL for plain tabs, "card" or "border-card".
tab_position	"top" (default), "right", "bottom" or "left".
closable	Show a close button on each tab. Closing removes the tab from the page; the server is told through input$<id>_tab_remove.
addable	Show a "+" button; clicking it reports input$<id>_tab_add, and the server adds a tab with insert_el_tab().
editable	closable and addable together.
stretch	Stretch the tabs to fill the available width.
before_leave	JS() function function(activeName, oldActiveName) run before switching tabs; return false, or a promise that rejects, to stay put.
add_icon	The add button's icon, by name. Default "Plus".

`events` Element's events to report besides those reported unasked, by name: `events = "node_drop"` reports `input$<id>_node_drop`. The component's are listed under "Shiny inputs", and by `el_events()`; a name it does not have is an error.

`session` In `el_tabs()`, deprecated: inside a module, wrap `id` in `ns()`, as for any Shiny input; a session given here namespaces `id` once more, with a warning. In `update_el_tabs()`, the Shiny session, the current one by default, as for `shiny::updateTextInput()`.

Details

Rendered as plain markup carrying Element's own classes, driven by a Shiny input binding rather than a Vue instance. That is what lets a tab hold other components from this package: a Vue instance mounted here would rebuild the DOM underneath them, leaving them rendered but disconnected from the server.

Value

An `htmltools` tag.

Shiny inputs

Input	Reported	Value
`input$<id>`	unasked	the name of the selected tab
`input$<id>_tab_click`	`events = "tab_click"`	the name of the tab clicked
`input$<id>_tab_change`	`events = "tab_change"`	the name of the tab selected
`input$<id>_tab_remove`	unasked	the name of the tab closed
`input$<id>_tab_add`	unasked	triggers when tab-add button is clicked
`input$<id>_edit`	`events = "edit"`	`list(target, action)`: the tab's name (`NULL` for an add) and `"add"`

The same list as `el_events("el_tabs")`, which says how an event's arguments travel.

Updating from the server

Server-side update for `el_tabs()`: the tab selected, and the tabs' look – `type` (`NA` for plain), `tab_position`, `closable`, `addable`, `editable`, `stretch` – as Element's props, all reactive there.

`update_el_tabs()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_tabs(
  "t1",
  selected = "a",
  tabs = list(
    list(name = "a", label = "First", content = shiny::tags$p("One")),
    list(name = "b", label = "Second", content = shiny::tags$p("Two"))
  )
)

# A tab can hold other components
```

```
el_tabs(
  "t2",
  tabs = list(
    list(
      name = "data",
      label = "Data",
      content = el_table(data = head(iris, 3))
    ),
    list(name = "opts", label = "Options", content = el_switch("live"))
  )
)
if (interactive()) {
  # inside a server function
  observeEvent(input$go, {
    update_el_tabs(session, "section", selected = "data")
  })
}
```

---

el_tab_pane *A tab of* el_tabs()

Description

Element Plus's el-tab-pane, for el_tabs(tabs =) and update_el_tabs(): el_tabs("t", tabs = list(el_tab_pane("One", ...), el_tab_pane("Two", ...))).

Usage

```
el_tab_pane(
  label,
  ...,
  name = label,
  disabled = NULL,
  closable = NULL,
  lazy = NULL
)
```

Arguments

label	Title of the tab.
...	The tab's content: tags, text, this package's components.
name	The tab's value, input$<id> while it is selected. Defaults to label.
disabled	Whether the tab is disabled.
closable	Whether this tab can be closed, when the tabs' own closable is off.
lazy	Render the content only when the tab is first selected. Its components do not exist, and report nothing, until then.

Value

A tab, for `el_tabs(tabs =)`.

See Also

Other items: `el_anchor_link()`, `el_breadcrumb_item()`, `el_carousel_item()`, `el_collapse_item()`, `el_descriptions_item()`, `el_dropdown_item()`, `el_menu_item()`, `el_option()`, `el_skeleton_item()`, `el_step()`, `el_table_column()`, `el_table_v2_column()`, `el_timeline_item()`, `el_tour_step()`

Examples

```
el_tabs(
  "t",
  tabs = list(
    el_tab_pane("User", "User settings"),
    el_tab_pane("Config", el_switch("dark"), lazy = TRUE)
  )
)
```

---

el_tag *Element Plus Tag*

Description

Creates a styled tag/chip component. Tracks click and close events as Shiny inputs.

Usage

```
el_tag(
  id = NULL,
  label = "Tag",
  type = NULL,
  closable = FALSE,
  size = NULL,
  effect = "light",
  color = NULL,
  hit = FALSE,
  disable_transitions = FALSE,
  round = NULL,
  width = NULL,
  slots = NULL,
  on = NULL,
  session = NULL
)

update_el_tag(
  session = shiny::getDefaultReactiveDomain(),
```

```
  id,
  label = NULL,
  type = NULL,
  closable = NULL,
  size = NULL,
  effect = NULL,
  color = NULL,
  hit = NULL,
  disable_transitions = NULL,
  round = NULL
)
```

Arguments

id	Tag ID. Auto-generated UUID if NULL.
label	Tag text.
type	Tag colour: NULL (default blue), "success", "info", "warning", "danger".
closable	Whether to show a close button. Default FALSE. When TRUE, input$<id>_close fires once when the user closes the tag.
size	Size: "large", "default" or "small"; NULL follows the form or the page.
effect	Visual effect: "light" (default), "dark", "plain".
color	Custom background colour (CSS string). NULL for themed colour.
hit	Whether to show a solid border. Default FALSE.
disable_transitions	Disable the zoom-in-center animation. Default FALSE.
round	Whether Tag is rounded. Element Plus's round (boolean).
width	Component width, as a CSS unit – "200px", "50%", or a number taken as pixels. Element's own markup carries it, so it behaves like the width argument of a Shiny input.
slots	Named list of Element slot contents, such as list(title = shiny::tags$b("Bold")). A shiny.element component given here is absorbed rather than nested. For a scoped slot, write the template with template().
on	Handlers of your own, for an event not reported or to send something else: a named list of JS() functions, one per event – Element's, or a DOM event with Vue's modifiers ("keyup.enter"). Each is called with report and the event's arguments; report(name, value) sets input$<id>_<name>. See el_widget().
session	In el_tag(), deprecated: inside a module, wrap id in ns(), as for any Shiny input; a session given here namespaces id once more, with a warning. In update_el_tag(), the Shiny session, the current one by default, as for shiny::updateTextInput().

Value

An htmltools tagList with a Vue-managed tag component.

Shiny inputs

Input	Reported	Value
`input$<id>`	unasked	the number of clicks on the tag, as `actionButton()` reports it
`input$<id>_close`	unasked	fires when the user closes it (`closable = TRUE`)

The same list as `el_events("el_tag")`, which says how an event's arguments travel.

The clicks are counted as `shiny::actionButton()` counts them, 0 on load.

Updating from the server

Server-side update for `el_tag()`.

Every other argument of `el_tag()` that can change once it is drawn is an argument here too, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_tag()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_tag("tag1", "Success", type = "success")
el_tag("tag2", "Closable", closable = TRUE)
if (interactive()) {
  # inside a server function
  observeEvent(input$go, {
    update_el_tag(session, "status", label = "done", type = "success")
  })
}
```

---

el_text *Element Plus Text*

Description

Text in Element's colours and sizes, truncated to a line or a number of lines if asked.

Usage

```
el_text(
  ...,
  id = NULL,
  type = NULL,
  size = NULL,
  truncated = NULL,
  line_clamp = NULL,
  tag = NULL,
  width = NULL,
  slots = NULL,
```

```
  on = NULL
)

update_el_text(
  session = shiny::getDefaultReactiveDomain(),
  id,
  type = NULL,
  size = NULL,
  truncated = NULL,
  line_clamp = NULL,
  tag = NULL
)
```

Arguments

...	Its content: any Shiny UI. Components of this package are folded into this one's Vue instance, as `el_button_group()` folds its buttons.
id	Component ID. Auto-generated if `NULL`.
type	Text type. Element Plus's `type` ('primary' | 'success' | 'warning' | 'danger' | 'info').
size	Text size. Element Plus's `size` ('large' | 'default' | 'small').
truncated	Render ellipsis. Element Plus's `truncated` (boolean).
line_clamp	Maximum lines. Element Plus's `line-clamp` (string / number).
tag	Custom element tag. Element Plus's `tag` (string).
width	Component width, as a CSS unit.
slots	Named list of Element slot contents. A scoped slot is written with `template()`.
on	Handlers of your own, for an event not reported or to send something else: a named list of `JS()` functions, one per event – Element's, or a DOM event with Vue's modifiers (`"keyup.enter"`). Each is called with `report` and the event's arguments; `report(name, value)` sets `input$<id>_<name>`. See `el_widget()`.
session	Shiny session; the current one by default, as for `shiny::updateTextInput()`.

Value

A Shiny UI element.

Shiny inputs

None: it reports nothing.

Updating from the server

`update_el_text()` changes the component from the server: every argument of `el_text()` that can change once it is drawn, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_text()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_text("Primary text", type = "primary")
el_text(strrep("A long sentence. ", 20), truncated = TRUE)
```

---

el_theme *Element Plus's look, as a Bootstrap theme*

Description

A `bslib::bs_theme()` carrying Element's own design tokens – its blue, its success, warning and danger colours, its greys, borders, 4px corners, 14px type and font stack – so that the Bootstrap side of a page, Shiny's own inputs and outputs among it, matches the Element components next to it. It is `el_page()`'s default.

Usage

```
el_theme(..., element = NULL, version = 5)
```

Arguments

`...`	Overrides, passed on to `bslib::bs_theme()`: its own arguments (`primary = "#7c3aed"`, `base_font =`), or any Bootstrap Sass variable by name (`"font-size-base" = "1rem"`). An override replaces the Element value of the same name.
`element`	Element Plus's own theme variables, as a named list – `list("border-radius-base" = "8px", "font-size-base" = "13px")` – by their CSS variable names without the `--el-`. They are set on the page, as Element Plus's theming guide sets them.
`version`	Bootstrap major version. Default 5.

Details

Element styles its components but not the page they sit on, and most of its components set no font of their own: they inherit one. Under a Bootswatch theme they inherit that theme's font instead of Element's, while Element's sizes and colours stay – a mix of the two looks.

The values follow Element Plus's CSS variables:

Bootstrap	Element Plus	value
`primary`	`-el-color-primary`	`#409EFF`
`success`	`-el-color-success`	`#67C23A`
`warning`	`-el-color-warning`	`#E6A23C`
`danger`	`-el-color-danger`	`#F56C6C`
`info`, `secondary`	`-el-color-info`	`#909399`
`fg`	`-el-text-color-primary`	`#303133`
`border-color`	`-el-border-color`	`#DCDFE6`
`border-radius`	`-el-border-radius-base`	`4px`
`font-size-base`	`-el-font-size-base`	`14px`

`primary`, `success`, `warning`, `danger` and `info` reach Element's components too, with the tints and shades Element derives from each, and so does anything given to `element`: `el_page()` sets Element Plus's CSS variables for the theme, as its theming guide does – no build involved.

Element puts white text on all five of its colours, some of which fall short of Bootstrap's default minimum contrast; left alone, Bootstrap would switch those buttons to black text. `min-contrast-ratio` is lowered so the text stays white, as Element has it. Pass `"min-contrast-ratio" = 4.5` to put contrast first.

Value

A `bs_theme` object, for `el_page()`'s `theme` or any page that takes a bslib theme.

Examples

```
el_theme()

# Element's look with another brand colour
el_theme(primary = "#7c3aed")

# Rounder and smaller, all through Element (compiles Element's Sass)

el_theme(
  element = list("border-radius-base" = "10px", "font-size-base" = "13px")
)


if (interactive()) {
  el_page(theme = el_theme(), shiny::actionButton("go", "Shiny's own button"))
}
```

---

el_timeline *Element Plus Timeline*

Description

A vertical sequence of events.

Usage

```
el_timeline(
  id = NULL,
  items = list(),
  reverse = FALSE,
  html = FALSE,
  mode = NULL,
  width = NULL,
  slots = NULL,
  on = NULL,
```

```
  session = NULL
)

update_el_timeline(
  session = shiny::getDefaultReactiveDomain(),
  id,
  items = NULL,
  reverse = NULL,
  mode = NULL
)
```

Arguments

id	Timeline ID (auto-generated if NULL).
items	A list of entries, each an el_timeline_item() – or a list with the same fields: content and optionally timestamp, type ("primary", "success", "warning", "danger" or "info"), color, size ("normal" or "large"), hide_timestamp, icon (an icon's name), placement ("bottom" or "top", where the timestamp goes), center (centre the dot against the content) and hollow (draw the dot hollow).
reverse	Show the entries newest first.
html	Render each entry's content as HTML rather than text – a string of markup, or tags. Only use it with content you control: it goes through v-html, which does not escape anything.
mode	Relative position of timeline and content. Element Plus's mode ('start' | 'alternate' | 'alternate-reverse' | 'end').
width	Component width, as a CSS unit – "200px", "50%", or a number taken as pixels. Element's own markup carries it, so it behaves like the width argument of a Shiny input.
slots	Named list of Element slot contents, such as list(title = shiny::tags$b("Bold")). A shiny.element component given here is absorbed rather than nested. For a scoped slot, write the template with template().
on	Handlers of your own, for an event not reported or to send something else: a named list of JS() functions, one per event – Element's, or a DOM event with Vue's modifiers ("keyup.enter"). Each is called with report and the event's arguments; report(name, value) sets input$<id>_<name>. See el_widget().
session	In el_timeline(), deprecated: inside a module, wrap id in ns(), as for any Shiny input; a session given here namespaces id once more, with a warning. In update_el_timeline(), the Shiny session, the current one by default, as for shiny::updateTextInput().

Details

Entries are rendered with v-for from a data field, so update_el_timeline() can replace them – useful for a log that grows. Content given as tags – el_card(...) – is drawn as markup; a string is text unless html = TRUE.

Value

A Shiny UI element.

Updating from the server

`update_el_timeline()` changes the component from the server.

Every other argument of `el_timeline()` that can change once it is drawn is an argument here too, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_timeline()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_timeline(
  id = "log",
  items = list(
    list(content = "Order placed", timestamp = "2026-03-01", type = "primary"),
    list(
      content = "Order shipped",
      timestamp = "2026-03-02",
      type = "success",
      icon = "el-icon-check",
      size = "large"
    ),
    list(content = "Delivered", timestamp = "2026-03-04", color = "#0bbd87")
  )
)

# Newest first, timestamps above each entry
el_timeline(
  id = "log",
  reverse = TRUE,
  items = list(
    list(content = "Second", timestamp = "10:30", placement = "top"),
    list(content = "First", timestamp = "09:15", placement = "top")
  )
)
if (interactive()) {
  # Append an entry to a growing log
  observeEvent(input$refresh, {
    log_entries(c(
      log_entries(),
      list(list(content = "Refreshed", timestamp = format(Sys.time(), "%H:%M")))
    ))
    update_el_timeline(session, "log", items = log_entries())
  })
}
```

el_timeline_item *An entry of* el_timeline()

Description

Element Plus's el-timeline-item, for el_timeline(items =).

Usage

```
el_timeline_item(
  content,
  timestamp = NULL,
  hide_timestamp = NULL,
  center = NULL,
  placement = NULL,
  type = NULL,
  color = NULL,
  size = NULL,
  icon = NULL,
  hollow = NULL
)
```

Arguments

content	The entry's text (markup with el_timeline(html = TRUE)).
timestamp	Timestamp text.
hide_timestamp	Whether to hide the timestamp. By default it is hidden when there is none.
center	Whether to centre the node vertically.
placement	Where the timestamp goes: "bottom" (Element's default) or "top".
type	Node type: "primary", "success", "warning", "danger" or "info".
color	Background colour of the node.
size	Node size: "normal" or "large".
icon	Icon of the node, by name.
hollow	Whether the node is hollow.

Value

An entry, for el_timeline(items =).

See Also

Other items: el_anchor_link(), el_breadcrumb_item(), el_carousel_item(), el_collapse_item(), el_descriptions_item(), el_dropdown_item(), el_menu_item(), el_option(), el_skeleton_item(), el_step(), el_tab_pane(), el_table_column(), el_table_v2_column(), el_tour_step()

Examples

```
el_timeline(
  "tl",
  items = list(
    el_timeline_item("Event start", timestamp = "2018-04-15"),
    el_timeline_item("Approved", timestamp = "2018-04-13", type = "success")
  )
)
```

`el_time_picker` *Element Plus Time Picker*

Description

Pick a time of day. `el_time_picker()` takes any time, or a range of them with `is_range = TRUE`; `el_time_select()` offers fixed times at a set interval, such as every half hour between nine and six.

Usage

```
el_time_picker(
  id = NULL,
  value = NULL,
  is_range = FALSE,
  value_format = "HH:mm:ss",
  arrow_control = NULL,
  placeholder = NULL,
  start_placeholder = NULL,
  end_placeholder = NULL,
  range_separator = NULL,
  clearable = NULL,
  disabled = NULL,
  editable = NULL,
  readonly = NULL,
  size = NULL,
  popper_class = NULL,
  default_value = NULL,
  prefix_icon = NULL,
  clear_icon = NULL,
  label = NULL,
  label_position = c("top", "left", "right"),
  label_width = NULL,
  label_suffix = NULL,
  required = FALSE,
  error = NULL,
  show_message = TRUE,
  inline_message = FALSE,
```

```
  format = NULL,
  popper_style = NULL,
  popper_options = NULL,
  placement = NULL,
  fallback_placements = NULL,
  disabled_hours = NULL,
  disabled_minutes = NULL,
  disabled_seconds = NULL,
  teleported = NULL,
  tabindex = NULL,
  aria_label = NULL,
  empty_values = NULL,
  value_on_clear = NULL,
  save_on_blur = NULL,
  width = NULL,
  slots = NULL,
  events = NULL,
  on = NULL,
  session = NULL
)

el_time_select(
  id = NULL,
  value = NULL,
  placeholder = NULL,
  clearable = NULL,
  disabled = NULL,
  editable = NULL,
  size = NULL,
  popper_class = NULL,
  prefix_icon = NULL,
  clear_icon = NULL,
  label = NULL,
  label_position = c("top", "left", "right"),
  label_width = NULL,
  label_suffix = NULL,
  required = FALSE,
  error = NULL,
  show_message = TRUE,
  inline_message = FALSE,
  start = NULL,
  end = NULL,
  step = NULL,
  min_time = NULL,
  max_time = NULL,
  include_end_time = NULL,
  format = NULL,
  effect = NULL,
```

```
  popper_style = NULL,
  empty_values = NULL,
  value_on_clear = NULL,
  width = NULL,
  slots = NULL,
  events = NULL,
  on = NULL,
  session = NULL
)

update_el_time_picker(
  session = shiny::getDefaultReactiveDomain(),
  id,
  value = NULL,
  disabled = NULL,
  label = NULL,
  error = NULL,
  is_range = NULL,
  value_format = NULL,
  arrow_control = NULL,
  placeholder = NULL,
  start_placeholder = NULL,
  end_placeholder = NULL,
  range_separator = NULL,
  clearable = NULL,
  editable = NULL,
  readonly = NULL,
  size = NULL,
  popper_class = NULL,
  prefix_icon = NULL,
  clear_icon = NULL,
  format = NULL,
  popper_style = NULL,
  popper_options = NULL,
  placement = NULL,
  fallback_placements = NULL,
  disabled_hours = NULL,
  disabled_minutes = NULL,
  disabled_seconds = NULL,
  teleported = NULL,
  tabindex = NULL,
  aria_label = NULL,
  empty_values = NULL,
  value_on_clear = NULL,
  save_on_blur = NULL
)

update_el_time_select(
```

```
  session = shiny::getDefaultReactiveDomain(),
  id,
  value = NULL,
  disabled = NULL,
  label = NULL,
  error = NULL,
  placeholder = NULL,
  clearable = NULL,
  editable = NULL,
  size = NULL,
  popper_class = NULL,
  prefix_icon = NULL,
  clear_icon = NULL,
  start = NULL,
  end = NULL,
  step = NULL,
  min_time = NULL,
  max_time = NULL,
  include_end_time = NULL,
  format = NULL,
  effect = NULL,
  popper_style = NULL,
  empty_values = NULL,
  value_on_clear = NULL
)
```

Arguments

`id`	Picker ID. Auto-generated if `NULL`.
`value`	Initial time, as `"HH:mm:ss"` text – two of them for a range.
`is_range`	Pick a start and an end rather than a single time.
`value_format`	Format of the value reported to Shiny, in day.js's tokens. Default `"HH:mm:ss"`.
`arrow_control`	Whether hours, minutes and seconds are changed with arrow buttons rather than by scrolling.
`placeholder`, `start_placeholder`, `end_placeholder`	Placeholder text, the latter two for a range.
`range_separator`	Text between the two times of a range. Default `"-"`.
`clearable`, `disabled`, `editable`, `readonly`	As for an input.
`size`	Size: `"large"`, `"default"` or `"small"`; `NULL` follows the form or the page.
`popper_class`, `popper_style`	Extra class name and style for the panel.
`default_value`	Time the panel opens on when nothing is picked.
`prefix_icon`, `clear_icon`	Icons, by name: `"Clock"`, `"CircleClose"`.

label — A label shown with the component, as Shiny's inputs have: text or a tag. NULL, the default, shows none. It is the component's accessible name too – tied to it with for where the component has a native input that takes the id <id>-input, else with aria-labelledby.

label_position — Where the label sits, as el_form()'s label_position: "top" (the default, as Shiny's labels sit), or beside the component, its text aligned "left" or "right" – which shows once label_width gives the labels a common width.

label_width — Width of a label beside the component, as a CSS unit, so that several line up. Element's label-width.

label_suffix — Text after the label, such as ":". Element's label-suffix.

required — Draw Element's red asterisk before the label. It marks the field; it does not check it – shinyvalidate or el_form() does that.

error — An error message shown under the component in Element's style, the field framed in red. Element's error.

show_message, inline_message — Whether error's message is shown, and whether beside the component rather than under it. Element's show-message and inline-message.

format — Format of the time shown in the input, in day.js's tokens.

popper_options, placement, fallback_placements — Where the panel opens, as Element Plus's tooltip takes them.

disabled_hours, disabled_minutes, disabled_seconds — JS() functions returning the hours, minutes or seconds that cannot be picked – what Element UI's selectableRange did.

teleported — Whether the panel is moved to <body>.

tabindex, aria_label — Native attributes of the input.

empty_values, value_on_clear — What counts as empty, and the value a cleared picker reports. See Element Plus's config provider.

save_on_blur — Whether the time typed is kept when the input loses focus.

width — Component width, as a CSS unit.

slots — Named list of Element slot contents.

events — Element's events to report besides those reported unasked, by name: events = "node_drop" reports input$<id>_node_drop. The component's are listed under "Shiny inputs", and by el_events(); a name it does not have is an error.

on — Handlers of your own, for an event not reported or to send something else: a named list of JS() functions, one per event – Element's, or a DOM event with Vue's modifiers ("keyup.enter"). Each is called with report and the event's arguments; report(name, value) sets input$<id>_<name>. See el_widget().

session — In el_time_picker(), deprecated: inside a module, wrap id in ns(), as for any Shiny input; a session given here namespaces id once more, with a warning. In update_el_time_picker(), the Shiny session, the current one by default, as for shiny::updateTextInput().

`include_end_time`, `start`, `end`, `step`, `min_time`, `max_time`
: For `el_time_select()`: the first and last time offered, the interval, whether `end` itself is offered, and the bounds of what can be picked.

`effect`
: `"light"` (default) or `"dark"` panel, for `el_time_select()`.

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
`input$<id>`	unasked	the time, two for a range
`input$<id>_blur`	`events = "blur"`	triggers when Input blurs
`input$<id>_focus`	`events = "focus"`	triggers when Input focuses
`input$<id>_clear`	`events = "clear"`	triggers when the clear icon is clicked in a clearable TimeP
`input$<id>_visible_change`	`events = "visible_change"`	triggers when the TimePicker's dropdown appears/disappea

The same list as `el_events("el_time_picker")`, which says how an event's arguments travel.

`el_time_select()`:

Input	Reported	Value
`input$<id>`	unasked	the time
`input$<id>_blur`	`events = "blur"`	triggers when Input blurs
`input$<id>_focus`	`events = "focus"`	triggers when Input focuses
`input$<id>_clear`	`events = "clear"`	triggers when the clear icon is clicked in a clearable TimeSelect

The same list as `el_events("el_time_select")`, which says how an event's arguments travel.

Element methods

Callable with `call_el()`: `focus()`, `blur()`; and for `el_time_picker()`, `handleOpen()` and `handleClose()`.

Updating from the server

Server-side update for `el_time_picker()` and `el_time_select()`; `update_el_time_select()` is the time select's, with its own arguments.

Every other argument of `el_time_picker()` that can change once it is drawn is an argument here too, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_time_picker()` is called for its side effect and returns `NULL` invisibly.

Updating a time select

`update_el_time_select()` changes the time select from the server: every argument of `el_time_select()` that can change once it is drawn, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_time_select()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_time_picker("start", value = "09:30:00")

# Only office hours
el_time_picker(
  "start",
  disabled_hours = JS(
    "function() {",
    "  var h = [];",
    "  for (var i = 0; i < 24; i++) if (i < 9 || i > 18) h.push(i);",
    "  return h;",
    "}"
  )
)

# A range
el_time_picker("shift", is_range = TRUE, value = c("09:00:00", "17:30:00"))

# Every half hour between nine and six
el_time_select("slot", start = "09:00", step = "00:30", end = "18:00")
if (interactive()) {
  # inside a server function
  observeEvent(
    input$reset,
    update_el_time_picker(session, "start", value = "09:00:00")
  )
}
```

---

el_tooltip *Element Plus Tooltip*

Description

A hint shown when the pointer rests on something.

Usage

```
el_tooltip(
  id = NULL,
  reference = NULL,
  content = NULL,
```

```
  trigger = NULL,
  placement = NULL,
  effect = NULL,
  disabled = NULL,
  offset = NULL,
  hide_after = NULL,
  enterable = NULL,
  transition = NULL,
  popper_class = NULL,
  popper_options = NULL,
  append_to = NULL,
  aria_label = NULL,
  arrow_offset = NULL,
  auto_close = NULL,
  fallback_placements = NULL,
  focus_on_target = NULL,
  persistent = NULL,
  popper_style = NULL,
  raw_content = NULL,
  show_after = NULL,
  show_arrow = NULL,
  teleported = NULL,
  trigger_keys = NULL,
  virtual_ref = NULL,
  virtual_triggering = NULL,
  visible = NULL,
  width = NULL,
  slots = NULL,
  events = NULL,
  on = NULL,
  session = NULL
)

update_el_tooltip(
  session = shiny::getDefaultReactiveDomain(),
  id,
  content = NULL,
  disabled = NULL,
  visible = NULL,
  virtual_ref = NULL
)
```

Arguments

id	Tooltip ID. Auto-generated if NULL.
reference	The element the tooltip describes. Any Shiny UI, including another shiny.element component – that component is folded into the tooltip's own Vue instance rather than nested inside it, so its inputs keep reporting. Its update_el_*() no longer

reaches it, though; see Details.

`content` Text of the hint.

`trigger` How it opens: `"hover"` (default), `"click"`, `"focus"` or `"contextmenu"`, or several of them as a character vector. Element Plus's `trigger`.

`placement` Where the hint appears: `"top"` (default), `"bottom"`, `"left"`, `"right"`, each also with `-start` and `-end`.

`effect` `"dark"` (default) or `"light"`.

`disabled` Whether the hint is suppressed.

`offset` Offset from the trigger, in pixels.

`hide_after` Hide it automatically after this many milliseconds. `0` (the default) keeps it until the pointer leaves.

`enterable` Whether the pointer may move onto the hint itself.

`transition` Name of the transition to animate with.

`popper_class` Extra class name for the hint.

`popper_options` Additional Popper.js options, as a named list.

`append_to` Which element the tooltip CONTENT appends to. Element Plus's `append-to` (CSSSelector / HTMLElement).

`aria_label` Same as `aria-label`. Element Plus's `aria-label` (string).

`arrow_offset` Controls the offset (padding) of the tooltip's arrow relative to the popper. Element Plus's `arrow-offset` (number).

`auto_close` Timeout in milliseconds to hide tooltip, not valid in controlled mode. Element Plus's `auto-close` (number).

`fallback_placements`
List of possible positions for Tooltip popper.js. Element Plus's `fallback-placements` (`Placement[]`).

`focus_on_target`
When triggering tooltips through hover, whether to focus the trigger element, which improves accessibility. Element Plus's `focus-on-target` (boolean).

`persistent` When tooltip inactive and `persistent` is `false` , tooltip will be destroyed. Element Plus's `persistent` (boolean).

`popper_style` Custom style for Tooltip's popper. Element Plus's `popper-style` (string / object).

`raw_content` Whether `content` is treated as HTML string. Element Plus's `raw-content` (boolean).

`show_after` Delay of appearance, in millisecond, not valid in controlled mode. Element Plus's `show-after` (number).

`show_arrow` Whether the tooltip content has an arrow. Element Plus's `show-arrow` (boolean).

`teleported` Whether tooltip content is teleported, if `true` it will be teleported to where `append-to` sets. Element Plus's `teleported` (boolean).

`trigger_keys` When you click the mouse to focus on the trigger element, you can define a set of keyboard codes to control the display of tooltip through the keyboard, not valid in controlled mode. Element Plus's `trigger-keys` (Array).

`virtual_ref` A CSS selector, `"#help-icon"`, for an element elsewhere on the page that the tooltip is attached to, in place of a trigger of its own. Element Plus's `virtual-ref` takes the element itself; the selector is looked up in the browser.

`virtual_triggering` Whether virtual triggering is enabled. Element Plus's `virtual-triggering` (boolean); `TRUE` when `virtual_ref` is given.

`visible` Visibility of Tooltip. Element Plus's `visible` (boolean).

`width` Component width, as a CSS unit.

`slots` Named list of Element slot contents, such as `list(title = shiny::tags$b("Bold"))`. A shiny.element component given here is absorbed rather than nested. For a scoped slot, write the template with `template()`.

`events` Element's events to report besides those reported unasked, by name: `events = "node_drop"` reports `input$<id>_node_drop`. The component's are listed under "Shiny inputs", and by `el_events()`; a name it does not have is an error.

`on` Handlers of your own, for an event not reported or to send something else: a named list of `JS()` functions, one per event – Element's, or a DOM event with Vue's modifiers (`"keyup.enter"`). Each is called with `report` and the event's arguments; `report(name, value)` sets `input$<id>_<name>`. See `el_widget()`.

`session` In `el_tooltip()`, deprecated: inside a module, wrap `id` in `ns()`, as for any Shiny input; a session given here namespaces `id` once more, with a warning. In `update_el_tooltip()`, the Shiny session, the current one by default, as for `shiny::updateTextInput()`.

Details

A component passed as `reference` becomes part of the tooltip's Vue instance rather than a separate one, which is what lets it survive being compiled into the tooltip's markup. It reports its inputs as usual, but it no longer has a host of its own, so its `update_el_*()` cannot find it – drive it through `update_vue()` on the tooltip's id instead.

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
`input$<id>_show`	`events = "show"`	Triggers when tooltip is shown. Passes trigger reason as argument.
`input$<id>_hide`	`events = "hide"`	Triggers when tooltip is hidden. Passes trigger reason as argument
`input$<id>_before_show`	`events = "before_show"`	Triggers before tooltip is shown. Passes trigger reason as argument
`input$<id>_before_hide`	`events = "before_hide"`	Triggers before tooltip is hidden. Passes trigger reason as argumen

The same list as `el_events("el_tooltip")`, which says how an event's arguments travel.

Updating from the server

Server-side update for el_tooltip(). Setting visible shows or hides the hint, which then stays as set: Element Plus's tooltip is controlled by its visible once that is given.

update_el_tooltip() is called for its side effect and returns NULL invisibly.

Examples

```
# A plain tag as the reference
el_tooltip(
  "hint",
  el$button(type = "primary", "Save"),
  content = "Writes to disk"
)

# Or a component, which keeps working
el_tooltip("hint", el_button("save", "Save"), content = "Writes to disk")

el_tooltip(
  "hint",
  reference = el$button(type = "danger", "Delete"),
  content = "This cannot be undone",
  placement = "right",
  effect = "light"
)
if (interactive()) {
  # inside a server function
  observeEvent(input$explain, {
    update_el_tooltip(session, "hint", content = why_disabled())
  })
}
```

---

el_tour *Element Plus Tour*

Description

A guided tour: a card pointing at one element of the page after another, with the rest of the page dimmed.

Usage

```
el_tour(
  id = NULL,
  steps = list(),
  visible = FALSE,
  current = NULL,
  show_arrow = NULL,
  placement = NULL,
```

```
  content_style = NULL,
  mask = NULL,
  gap = NULL,
  type = NULL,
  scroll_into_view_options = NULL,
  z_index = NULL,
  show_close = NULL,
  close_icon = NULL,
  close_on_press_escape = NULL,
  target_area_clickable = NULL,
  append_to = NULL,
  width = NULL,
  slots = NULL,
  events = NULL,
  on = NULL
)

update_el_tour(
  session = shiny::getDefaultReactiveDomain(),
  id,
  visible = NULL,
  current = NULL,
  show_arrow = NULL,
  placement = NULL,
  content_style = NULL,
  mask = NULL,
  gap = NULL,
  type = NULL,
  scroll_into_view_options = NULL,
  z_index = NULL,
  show_close = NULL,
  close_icon = NULL,
  close_on_press_escape = NULL,
  target_area_clickable = NULL,
  append_to = NULL
)
```

Arguments

id	Tour ID. Auto-generated if NULL.
steps	The steps, each an el_tour_step() – or a list(target = "#css-selector", title =, description =), each with Element Plus's other step props if wanted – placement, mask, type, show_arrow, show_close, content_style, scroll_into_view_options, and header, markup in place of the title. A step without a target shows in the middle of the screen.
visible	Whether it starts open: Element Plus's model-value, named visible as on the other overlays. Open it later with update_el_tour().
current	The step it starts on, from 0.

`show_arrow`, `placement`, `content_style`, `mask`, `gap`, `type`, `scroll_into_view_options`, `z_index`, `show_close`, `close_icon`, `close_on_press_escape`, `target_area_clickable`, `append_to`
: Element Plus's tour props of those names: the defaults for every step.

`width`
: Component width, as a CSS unit.

`slots`
: Named list of Element slot contents: `indicators`.

`events`
: Element's events to report besides those reported unasked, by name: `events = "node_drop"` reports `input$<id>_node_drop`. The component's are listed under "Shiny inputs", and by `el_events()`; a name it does not have is an error.

`on`
: Handlers of your own, for an event not reported or to send something else: a named list of `JS()` functions, one per event – Element's, or a DOM event with Vue's modifiers (`"keyup.enter"`). Each is called with `report` and the event's arguments; `report(name, value)` sets `input$<id>_<name>`. See `el_widget()`.

`session`
: Shiny session; the current one by default, as for `shiny::updateTextInput()`.

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
`input$<id>`	unasked	`TRUE` while it is open
`input$<id>_close`	unasked	the step it was closed on
`input$<id>_change`	unasked	the step, from 0
`input$<id>_finish`	unasked	callback function on finished

The same list as `el_events("el_tour")`, which says how an event's arguments travel.

Updating from the server

Open or close an `el_tour()`, or move it to a step.

Every other argument of `el_tour()` that can change once it is drawn is an argument here too, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_tour()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_tour(
  "intro",
  visible = TRUE,
  steps = list(
    list(
      target = "#upload",
      title = "Upload",
      description = "Put your file here."
```

```
    ),
    list(target = "#run", title = "Run", description = "Then press this.")
  )
)
if (interactive()) {
  # inside a server function
  observeEvent(
    input$help,
    update_el_tour(session, "intro", visible = TRUE, current = 0)
  )
}
```

---

el_tour_step *A step of* el_tour()

Description

Element Plus's el-tour-step, for el_tour(steps =).

Usage

```
el_tour_step(
  target = NULL,
  title = NULL,
  description = NULL,
  header = NULL,
  show_arrow = NULL,
  placement = NULL,
  content_style = NULL,
  mask = NULL,
  type = NULL,
  next_button_props = NULL,
  prev_button_props = NULL,
  scroll_into_view_options = NULL,
  show_close = NULL,
  close_icon = NULL
)
```

Arguments

target	The element the step points at, as a CSS selector. NULL shows the step in the middle of the screen.
title, description	The step's title and text.
header	Markup in place of the title.
show_arrow	Whether to show the arrow.

placement — Where the card goes, relative to the target: "top", "bottom" (Element's default), "left", "right", each also with "-start" or "-end".

content_style — Style of the content, a list of CSS properties.

mask — Whether to mask the page, or the mask's list(style =, color =).

type — "default" or "primary": the card's colours.

next_button_props, prev_button_props — The Next and Previous buttons' properties: list(children = "Go on").

scroll_into_view_options — Whether to scroll the target into view, or the options for scrollIntoView().

show_close — Whether to show a close button.

close_icon — The close button's icon, by name.

Value

A step, for el_tour(steps =).

See Also

Other items: el_anchor_link(), el_breadcrumb_item(), el_carousel_item(), el_collapse_item(), el_descriptions_item(), el_dropdown_item(), el_menu_item(), el_option(), el_skeleton_item(), el_step(), el_tab_pane(), el_table_column(), el_table_v2_column(), el_timeline_item()

Examples

```
el_tour(
  "tour",
  steps = list(
    el_tour_step("#upload", "Upload File", "Put your files here."),
    el_tour_step(title = "Done", description = "That is all.")
  )
)
```

---

el_transfer *Element Plus Transfer*

Description

Two lists side by side, for moving items from one to the other.

Usage

```
el_transfer(
  id = NULL,
  data = list(),
  value = NULL,
  titles = NULL,
```

```
  button_texts = NULL,
  filterable = NULL,
  filter_placeholder = NULL,
  filter_method = NULL,
  target_order = NULL,
  format = NULL,
  props = NULL,
  left_default_checked = NULL,
  right_default_checked = NULL,
  render_content = NULL,
  label = NULL,
  label_position = c("top", "left", "right"),
  label_width = NULL,
  label_suffix = NULL,
  required = FALSE,
  error = NULL,
  show_message = TRUE,
  inline_message = FALSE,
  item_size = NULL,
  validate_event = NULL,
  virtual_scroll = NULL,
  width = NULL,
  slots = NULL,
  events = NULL,
  on = NULL,
  session = NULL
)

update_el_transfer(
  session = shiny::getDefaultReactiveDomain(),
  id,
  value = NULL,
  data = NULL,
  titles = NULL,
  filterable = NULL,
  label = NULL,
  error = NULL,
  button_texts = NULL,
  filter_placeholder = NULL,
  filter_method = NULL,
  target_order = NULL,
  format = NULL,
  props = NULL,
  left_default_checked = NULL,
  right_default_checked = NULL,
  render_content = NULL,
  item_size = NULL,
  validate_event = NULL,
```

```
  virtual_scroll = NULL
)
```

Arguments

id	Transfer ID. Auto-generated if NULL.
data	The items to choose from, as a data.frame with columns key and label (and optionally disabled), or a list of list(key =, label =, disabled =).
value	Keys that start out on the right.
titles	Headings of the two panels, as a length-2 character vector. Default c("List 1", "List 2").
button_texts	Labels of the two buttons, as a length-2 character vector. Default is arrows only.
filterable	Whether each panel gets a search box.
filter_placeholder	Placeholder of the search boxes.
filter_method	JS() function function(query, item) returning whether an item survives the search.
target_order	Order of the right-hand panel: "original" (default), "push" or "unshift".
format	Counts shown in each heading, as list(noChecked =, hasChecked =), for example list(noChecked = "${total}", hasChecked = "${checked}/${total}").
props	Field names when data uses other ones, as list(key =, label =, disabled =).
left_default_checked, right_default_checked	Keys ticked at the start.
render_content	JS() render function for an item.
label	A label shown with the component, as Shiny's inputs have: text or a tag. NULL, the default, shows none. It is the component's accessible name too – tied to it with for where the component has a native input that takes the id <id>-input, else with aria-labelledby.
label_position	Where the label sits, as el_form()'s label_position: "top" (the default, as Shiny's labels sit), or beside the component, its text aligned "left" or "right" – which shows once label_width gives the labels a common width.
label_width	Width of a label beside the component, as a CSS unit, so that several line up. Element's label-width.
label_suffix	Text after the label, such as ":". Element's label-suffix.
required	Draw Element's red asterisk before the label. It marks the field; it does not check it – shinyvalidate or el_form() does that.
error	An error message shown under the component in Element's style, the field framed in red. Element's error.
show_message, inline_message	Whether error's message is shown, and whether beside the component rather than under it. Element's show-message and inline-message.
item_size	Item height for virtual scrolling. Element Plus's item-size (number).

validate_event	Whether to trigger form validation. Element Plus's validate-event (boolean).
virtual_scroll	Whether to enable virtual scrolling. Element Plus's virtual-scroll (boolean).
width	Component width, as a CSS unit.
slots	Named list of Element slot contents, such as list(title = shiny::tags$b("Bold")). A shiny.element component given here is absorbed rather than nested. For a scoped slot, write the template with template().
events	Element's events to report besides those reported unasked, by name: events = "node_drop" reports input$<id>_node_drop. The component's are listed under "Shiny inputs", and by el_events(); a name it does not have is an error.
on	Handlers of your own, for an event not reported or to send something else: a named list of JS() functions, one per event – Element's, or a DOM event with Vue's modifiers ("keyup.enter"). Each is called with report and the event's arguments; report(name, value) sets input$<id>_<name>. See el_widget().
session	In el_transfer(), deprecated: inside a module, wrap id in ns(), as for any Shiny input; a session given here namespaces id once more, with a warning. In update_el_transfer(), the Shiny session, the current one by default, as for shiny::updateTextInput().

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
input$<id>	unasked	the keys on the right
input$<id>_change	events = "change"	list(value, direction, moved)
input$<id>_left_check_change	events = "left_check_change"	list(checked, changed)
input$<id>_right_check_change	events = "right_check_change"	list(checked, changed)

The same list as el_events("el_transfer"), which says how an event's arguments travel.

Element methods

Callable with call_el():

- clearQuery() – clear one panel's search box; pass "left" or "right"

Updating from the server

Server-side update for el_transfer().

Every other argument of el_transfer() that can change once it is drawn is an argument here too, under the same name. One left NULL stays as it is; NA returns it to Element's default.

update_el_transfer() is called for its side effect and returns NULL invisibly.

Examples

```
el_transfer(
  "cols",
  data = data.frame(key = names(iris), label = names(iris)),
  value = c("Species")
)

el_transfer(
  "cols",
  data = data.frame(key = names(mtcars), label = names(mtcars)),
  titles = c("Available", "Chosen"),
  filterable = TRUE,
  width = "100%"
)
if (interactive()) {
  # inside a server function
  observeEvent(input$reset, {
    update_el_transfer(session, "cols", value = list())
  })
}
```

---

el_tree *Element Plus Tree*

Description

A tree view, optionally with checkboxes.

Usage

```
el_tree(
  id = NULL,
  data = list(),
  node_key = "id",
  props = NULL,
  show_checkbox = FALSE,
  check_strictly = FALSE,
  default_expand_all = FALSE,
  expand_on_click_node = TRUE,
  accordion = FALSE,
  highlight_current = FALSE,
  default_expanded_keys = NULL,
  default_checked_keys = NULL,
  empty_text = NULL,
  indent = NULL,
  lazy = NULL,
  draggable = NULL,
  auto_expand_parent = NULL,
```

```
  check_on_click_node = NULL,
  current_node_key = NULL,
  render_after_expand = NULL,
  load = NULL,
  filter_node_method = NULL,
  render_content = NULL,
  allow_drag = NULL,
  allow_drop = NULL,
  label = NULL,
  label_position = c("top", "left", "right"),
  label_width = NULL,
  label_suffix = NULL,
  required = FALSE,
  error = NULL,
  show_message = TRUE,
  inline_message = FALSE,
  check_on_click_leaf = NULL,
  icon = NULL,
  width = NULL,
  slots = NULL,
  events = NULL,
  on = NULL,
  session = NULL
)

update_el_tree(
  session = shiny::getDefaultReactiveDomain(),
  id,
  data = NULL,
  default_expanded_keys = NULL,
  default_checked_keys = NULL,
  props = NULL,
  label = NULL,
  error = NULL,
  node_key = NULL,
  show_checkbox = NULL,
  check_strictly = NULL,
  expand_on_click_node = NULL,
  accordion = NULL,
  highlight_current = NULL,
  empty_text = NULL,
  indent = NULL,
  lazy = NULL,
  draggable = NULL,
  auto_expand_parent = NULL,
  check_on_click_node = NULL,
  current_node_key = NULL,
  render_after_expand = NULL,
```

```
  load = NULL,
  filter_node_method = NULL,
  render_content = NULL,
  allow_drag = NULL,
  allow_drop = NULL,
  check_on_click_leaf = NULL,
  icon = NULL
)
```

Arguments

id	Tree ID (auto-generated if NULL).
data	A list of nodes. Each is a list with its key (node_key) and its label, and optionally children, disabled for an uncheckable node, or isLeaf – under these names, or those props gives.
node_key	Field holding each node's unique key. The keys are what the server sees and what default_expanded_keys and default_checked_keys refer to.
props	Which field of a node holds what, as Element Plus's props: list(label =, children =, disabled =, isLeaf =, class =), each a field name or a JS() function of the node's data (and node) – class gives a node a class of its own. Those left out are Element's: "label", "children", "disabled", "isLeaf". is_leaf is read as isLeaf.
show_checkbox	Show a checkbox beside every node.
check_strictly	Treat a parent's checkbox as independent of its children, rather than checking them together.
default_expand_all	Expand every node initially.
expand_on_click_node	Expand a node when its label is clicked, as well as its arrow. Set FALSE to make clicking select rather than expand.
accordion	Keep only one node expanded per level.
highlight_current	Highlight the clicked node.
default_expanded_keys, default_checked_keys	Keys of the nodes to expand and to check, as Element Plus's default-expanded-keys and default-checked-keys: the tree keeps its own state from there, and reports the checked ones as input$<id>_checked. update_el_tree() sets them again.
empty_text	Text shown when data is empty.
indent	Horizontal indent between levels, in pixels. Default 16.
lazy	Whether child nodes are loaded on demand – from the server, unless load is given. See "Shiny inputs".
draggable	Whether nodes can be dragged.
auto_expand_parent	Whether expanding a node expands its parents. Default TRUE.

check_on_click_node
: Whether clicking a node's label also checks it.

current_node_key
: Key of the node that starts out highlighted.

render_after_expand
: Whether child nodes are rendered only once expanded. Default TRUE.

load
: JS() function loading child nodes in the browser instead of from the server. Needs lazy = TRUE.

filter_node_method
: JS() function deciding whether a node survives filtering. By default a node is kept when its label contains the text, ignoring case, so call_el(session, id, "filter", list(text)) works as it stands.

render_content
: JS() render function for a node's content.

allow_drag
: JS() function deciding whether a node may be dragged.

allow_drop
: JS() function deciding whether a node may be dropped somewhere.

label
: A label shown with the component, as Shiny's inputs have: text or a tag. NULL, the default, shows none. It is the component's accessible name too – tied to it with for where the component has a native input that takes the id <id>-input, else with aria-labelledby.

label_position
: Where the label sits, as el_form()'s label_position: "top" (the default, as Shiny's labels sit), or beside the component, its text aligned "left" or "right" – which shows once label_width gives the labels a common width.

label_width
: Width of a label beside the component, as a CSS unit, so that several line up. Element's label-width.

label_suffix
: Text after the label, such as ":". Element's label-suffix.

required
: Draw Element's red asterisk before the label. It marks the field; it does not check it – shinyvalidate or el_form() does that.

error
: An error message shown under the component in Element's style, the field framed in red. Element's error.

show_message, inline_message
: Whether error's message is shown, and whether beside the component rather than under it. Element's show-message and inline-message.

check_on_click_leaf
: Whether to check or uncheck node when clicking on leaf node (last children). Element Plus's check-on-click-leaf (boolean).

icon
: Custom tree node icon component. Element Plus's icon (string / Component). An icon's name, such as "Search".

width
: Component width, as a CSS unit – "200px", "50%", or a number taken as pixels. Element's own markup carries it, so it behaves like the width argument of a Shiny input.

slots
: Named list of Element slot contents, such as list(title = shiny::tags$b("Bold")). A shiny.element component given here is absorbed rather than nested. For a scoped slot, write the template with template().

`events` Element's events to report besides those reported unasked, by name: `events = "node_drop"` reports `input$<id>_node_drop`. The component's are listed under "Shiny inputs", and by `el_events()`; a name it does not have is an error.

`on` Handlers of your own, for an event not reported or to send something else: a named list of `JS()` functions, one per event – Element's, or a DOM event with Vue's modifiers (`"keyup.enter"`). Each is called with `report` and the event's arguments; `report(name, value)` sets `input$<id>_<name>`. See `el_widget()`.

`session` In `el_tree()`, deprecated: inside a module, wrap `id` in `ns()`, as for any Shiny input; a session given here namespaces `id` once more, with a warning. In `update_el_tree()`, the Shiny session, the current one by default, as for `shiny::updateTextInput()`.

Details

Unlike the menu, a tree takes its whole structure through a `data` prop rather than nested tags, so the nesting is plain R data all the way down.

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
`input$<id>`	unasked	the key of the node last clicked
`input$<id>_checked`	unasked	the keys of the checked nodes
`input$<id>_load`	unasked	with `lazy = TRUE`, a node asking for its children; answ
`input$<id>_check_change`	`events = "check_change"`	`list(data, checked, indeterminate)`
`input$<id>_current_change`	`events = "current_change"`	`list(data, key, level)` of the current node
`input$<id>_node_expand`	`events = "node_expand"`	`list(data, key, level)`
`input$<id>_node_collapse`	`events = "node_collapse"`	`list(data, key, level)`
`input$<id>_node_contextmenu`	`events = "node_contextmenu"`	`list(data, key, level)`
`input$<id>_node_drag_start`	`events = "node_drag_start"`	`list(data)` of the dragged node
`input$<id>_node_drag_enter`	`events = "node_drag_enter"`	`list(dragging, drop)`, the nodes' data
`input$<id>_node_drag_leave`	`events = "node_drag_leave"`	`list(dragging, drop)`, the nodes' data
`input$<id>_node_drag_over`	`events = "node_drag_over"`	`list(dragging, drop)`, the nodes' data, at most ever
`input$<id>_node_drag_end`	`events = "node_drag_end"`	`list(dragging, drop, type)`
`input$<id>_node_drop`	`events = "node_drop"`	`list(dragging, drop, type)`: `type "before", "aft`

The same list as `el_events("el_tree")`, which says how an event's arguments travel.
`input$<id>` and `input$<id>_checked` are reported on load, where they start empty and therefore arrive as `NULL`, as Shiny reports any empty selection.

With `lazy = TRUE` and no `load` of your own, the server loads each node's children: `input$<id>_load` asks, with `level` (0 for the top), `key` (the node's `node_key` field), `data` (the node) and `request`; answer with `el_load_children()`.

Element methods

Callable with `call_el()`:

- `append()` – Append a child node to a given node in the tree
- `filter()` – Filter all tree nodes, filtered nodes will be hidden
- `getCheckedKeys()` – If the node can be selected (show-checkbox is true), it returns the currently selected array of node's keys
- `getCheckedNodes()` – If the node can be selected (show-checkbox is true), it returns the currently selected array of nodes
- `getCurrentKey()` – Return the highlight node's key (null if no node is highlighted)
- `getCurrentNode()` – Return the highlight node's data (null if no node is highlighted)
- `getHalfCheckedKeys()` – If the node can be selected (show-checkbox is true), it returns the currently half selected array of...
- `getHalfCheckedNodes()` – If the node can be selected (show-checkbox is true), it returns the currently half selected array of nodes
- `getNode()` – Get node by data or key
- `insertAfter()` – Insert a node after a given node in the tree
- `insertBefore()` – Insert a node before a given node in the tree
- `remove()` – Remove a node, only works when node-key is assigned
- `setChecked()` – Set node to be checked or not, only works when node-key is assigned
- `setCheckedKeys()` – Set certain nodes to be checked, only works when node-key is assigned
- `setCheckedNodes()` – Set certain nodes to be checked, only works when node-key is assigned
- `setCurrentKey()` – Set highlighted node by key, only works when node-key is assigned
- `setCurrentNode()` – Set highlighted node, only works when node-key is assigned
- `updateKeyChildren()` – Set new data to node, only works when node-key is assigned

Updating from the server

`update_el_tree()` changes the component from the server.

Every other argument of `el_tree()` that can change once it is drawn is an argument here too, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_tree()` is called for its side effect and returns `NULL` invisibly.

Examples

```
nodes <- list(
  list(
    id = "fruit",
    label = "Fruit",
    children = list(
      list(id = "apple", label = "Apple"),
      list(id = "cherry", label = "Cherry")
    )
  ),
```

```
  list(
    id = "veg",
    label = "Vegetables",
    children = list(
      list(id = "leek", label = "Leek", disabled = TRUE)
    )
  )
)

el_tree(id = "picker", data = nodes)

# With checkboxes, two nodes checked and the first branch open
el_tree(
  id = "picker",
  data = nodes,
  show_checkbox = TRUE,
  default_checked_keys = c("apple", "cherry"),
  default_expanded_keys = "fruit"
)

# Nodes whose fields are named otherwise
el_tree(
  data = list(list(
    id = 1,
    name = "Docs",
    kids = list(list(id = 2, name = "R"))
  )),
  props = list(label = "name", children = "kids")
)
if (interactive()) {
  # inside a server function
  observeEvent(input$go, {
    update_el_tree(session, "picker", default_checked_keys = "apple")
  })
}
```

`el_tree_select` *Element Plus Tree Select*

Description

A select whose options are a tree: Element Plus's `el-tree-select`, which takes the props of both `el_select()` and `el_tree()`.

Usage

```
el_tree_select(
  id,
  data = list(),
  value = NULL,
```

```
  multiple = NULL,
  show_checkbox = NULL,
  check_strictly = NULL,
  check_on_click_node = NULL,
  filterable = NULL,
  clearable = NULL,
  placeholder = NULL,
  node_key = NULL,
  props = NULL,
  default_expand_all = NULL,
  render_after_expand = NULL,
  collapse_tags = NULL,
  collapse_tags_tooltip = NULL,
  size = NULL,
  disabled = NULL,
  cache_data = NULL,
  lazy = NULL,
  load = NULL,
  ...,
  label = NULL,
  label_position = c("top", "left", "right"),
  label_width = NULL,
  label_suffix = NULL,
  required = FALSE,
  error = NULL,
  show_message = TRUE,
  inline_message = FALSE,
  width = NULL,
  slots = NULL,
  events = NULL,
  on = NULL
)

update_el_tree_select(
  session = shiny::getDefaultReactiveDomain(),
  id,
  value = NULL,
  data = NULL,
  disabled = NULL,
  label = NULL,
  error = NULL,
  multiple = NULL,
  show_checkbox = NULL,
  check_strictly = NULL,
  check_on_click_node = NULL,
  filterable = NULL,
  clearable = NULL,
  placeholder = NULL,
```

```
  node_key = NULL,
  props = NULL,
  render_after_expand = NULL,
  collapse_tags = NULL,
  collapse_tags_tooltip = NULL,
  size = NULL,
  cache_data = NULL,
  lazy = NULL,
  load = NULL
)
```

Arguments

id	Component ID; the value is reported as input$<id>.
data	The tree: a list of nodes, each list(value =, label =, children = list(...)).
value	The selected value, or several with multiple = TRUE.
multiple	Whether several nodes can be selected.
show_checkbox	Whether nodes have checkboxes.
check_strictly	Whether any node can be selected, not only leaves.
check_on_click_node	Whether clicking a node checks it, with show_checkbox.
filterable	Whether the options can be searched by typing.
clearable	Whether the selection can be cleared.
placeholder	Placeholder text.
node_key	The field that identifies a node. Default "value".
props	Where the tree's fields are: list(label =, children =, disabled =, isLeaf =).
default_expand_all	Whether every node starts expanded.
render_after_expand	Whether a node's children are drawn only once it is expanded. Default TRUE.
collapse_tags, collapse_tags_tooltip	With multiple, whether the selection is shown as one tag and a count, with the rest in a tooltip.
size	"large", "default" or "small".
disabled	Whether it can be changed.
cache_data	The nodes behind a value not yet loaded, for a lazy tree.
lazy	Whether child nodes are loaded on demand – from the server, which answers input$<id>_load with el_load_children(), unless load is given.
load	JS() function loading child nodes in the browser instead of from the server. Needs lazy = TRUE.
...	Any other prop of Element Plus's select or tree, in snake_case: max_collapse_tags = 2, expand_on_click_node = FALSE.

`label` A label shown with the component, as Shiny's inputs have: text or a tag. `NULL`, the default, shows none. It is the component's accessible name too – tied to it with `for` where the component has a native input that takes the id `<id>-input`, else with `aria-labelledby`.

`label_position` Where the label sits, as `el_form()`'s `label_position`: `"top"` (the default, as Shiny's labels sit), or beside the component, its text aligned `"left"` or `"right"` – which shows once `label_width` gives the labels a common width.

`label_width` Width of a label beside the component, as a CSS unit, so that several line up. Element's `label-width`.

`label_suffix` Text after the label, such as `":"`. Element's `label-suffix`.

`required` Draw Element's red asterisk before the label. It marks the field; it does not check it – shinyvalidate or `el_form()` does that.

`error` An error message shown under the component in Element's style, the field framed in red. Element's `error`.

`show_message, inline_message` Whether `error`'s message is shown, and whether beside the component rather than under it. Element's `show-message` and `inline-message`.

`width` Component width, as a CSS unit.

`slots` Named list of Element slot contents.

`events` Element's events to report besides those reported unasked, by name: `events = "node_drop"` reports `input$<id>_node_drop`. The component's are listed under "Shiny inputs", and by `el_events()`; a name it does not have is an error.

`on` Handlers of your own, for an event not reported or to send something else: a named list of `JS()` functions, one per event – Element's, or a DOM event with Vue's modifiers (`"keyup.enter"`). Each is called with `report` and the event's arguments; `report(name, value)` sets `input$<id>_<name>`. See `el_widget()`.

`session` Shiny session; the current one by default, as for `shiny::updateSelectInput()`.

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
`input$<id>`	unasked	the value, several with `multiple`
`input$<id>_load`	unasked	with `lazy = TRUE`, a node asking for its children; answer wi
`input$<id>_visible_change`	`events = "visible_change"`	triggers when the dropdown appears/disappears
`input$<id>_clear`	`events = "clear"`	triggers when the clear icon is clicked in a clearable Select
`input$<id>_remove_tag`	`events = "remove_tag"`	triggers when a tag is removed in multiple mode
`input$<id>_node_click`	`events = "node_click"`	triggers when a node is clicked
`input$<id>_check`	`events = "check"`	triggers after clicking the checkbox of a node

The same list as `el_events("el_tree_select")`, which says how an event's arguments travel.

Element methods

Callable with call_el(): focus(), blur().

Updating from the server

update_el_tree_select() changes the component from the server.

Every other argument of el_tree_select() that can change once it is drawn is an argument here too, under the same name. One left NULL stays as it is; NA returns it to Element's default.

update_el_tree_select() is called for its side effect and returns NULL invisibly.

Examples

```
el_tree_select(
  "dept",
  placeholder = "Department",
  data = list(
    list(
      value = "eng",
      label = "Engineering",
      children = list(
        list(value = "web", label = "Web"),
        list(value = "data", label = "Data")
      )
    ),
    list(value = "ops", label = "Operations")
  )
)
if (interactive()) {
  # inside a server function
  observeEvent(
    input$reset,
    update_el_tree_select(session, "dept", value = "ops")
  )
}
```

---

el_tree_v2 *Element Plus Virtualized Tree*

Description

A tree drawing only the nodes in view, for tens of thousands of nodes.

Usage

```
el_tree_v2(
  id = NULL,
  data = NULL,
  empty_text = NULL,
```

```
  highlight_current = NULL,
  expand_on_click_node = NULL,
  check_on_click_node = NULL,
  check_on_click_leaf = NULL,
  default_expanded_keys = NULL,
  show_checkbox = NULL,
  check_strictly = NULL,
  default_checked_keys = NULL,
  current_node_key = NULL,
  filter_method = NULL,
  indent = NULL,
  icon = NULL,
  item_size = NULL,
  scrollbar_always_on = NULL,
  props = NULL,
  height = NULL,
  width = NULL,
  slots = NULL,
  events = NULL,
  on = NULL
)

update_el_tree_v2(
  session = shiny::getDefaultReactiveDomain(),
  id,
  data = NULL,
  empty_text = NULL,
  highlight_current = NULL,
  expand_on_click_node = NULL,
  check_on_click_node = NULL,
  check_on_click_leaf = NULL,
  show_checkbox = NULL,
  check_strictly = NULL,
  indent = NULL,
  icon = NULL,
  item_size = NULL,
  scrollbar_always_on = NULL,
  props = NULL,
  height = NULL,
  current_node_key = NULL,
  filter_method = NULL
)
```

Arguments

id	Component ID. Auto-generated if NULL.
data	Tree data. Element Plus's data (Array<{[key: string]: any}>).
empty_text	Text displayed when data is void. Element Plus's empty-text (string).

highlight_current
: Whether current node is highlighted. Element Plus's highlight-current (boolean).

expand_on_click_node
: Whether to expand or collapse node when clicking on the node, if false, then expand or collapse node only when clicking on the arrow icon. Element Plus's expand-on-click-node (boolean).

check_on_click_node
: Whether to check or uncheck node when clicking on the node, if false, the node can only be checked or unchecked by clicking on the checkbox. Element Plus's check-on-click-node (boolean).

check_on_click_leaf
: Whether to check or uncheck node when clicking on leaf node (last children). Element Plus's check-on-click-leaf (boolean).

default_expanded_keys
: Array of keys of initially expanded nodes. Element Plus's default-expanded-keys (Array<string | number>).

show_checkbox
: Whether node is selectable. Element Plus's show-checkbox (boolean).

check_strictly
: Whether checked state of a node not affects its father and child nodes when show-checkbox is true. Element Plus's check-strictly (boolean).

default_checked_keys
: Array of keys of initially checked nodes. Element Plus's default-checked-keys (Array<string | number>).

current_node_key
: Key of initially selected node. Element Plus's current-node-key (string / number).

filter_method
: This function will be executed on each node when use filter method. if return false, tree node will be hidden. Element Plus's filter-method ((query: string, data: TreeNodeData, node: TreeNode) => boolean). Give it as JS().

indent
: Horizontal indentation of nodes in adjacent levels in pixels. Element Plus's indent (number).

icon
: Custom tree node icon component. Element Plus's icon (string / Component). An icon's name, such as "Search".

item_size
: Custom tree node height. Element Plus's item-size (number).

scrollbar_always_on
: Always show scrollbar. Element Plus's scrollbar-always-on (boolean).

props
: Which field of a node holds what: list(value =, label =, children =, disabled =, class =), Element Plus's props. class may be a JS() function of the node's data returning its class.

height
: Height of the tree. Element Plus's height (number).

width
: Component width, as a CSS unit.

slots
: Named list of Element slot contents: empty. A scoped slot is written with template().

events
: Element's events to report besides those reported unasked, by name: events = "node_drop" reports input$<id>_node_drop. The component's are listed under "Shiny inputs", and by el_events(); a name it does not have is an error.

on — Handlers of your own, for an event not reported or to send something else: a named list of JS() functions, one per event – Element's, or a DOM event with Vue's modifiers ("keyup.enter"). Each is called with report and the event's arguments; report(name, value) sets input$<id>_<name>. See el_widget().

session — Shiny session; the current one by default, as for shiny::updateTextInput().

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
input$<id>_node_click	unasked	triggers when a node is clicked
input$<id>_node_drop	events = "node_drop"	triggers when drag something and drop on a node
input$<id>_node_contextmenu	events = "node_contextmenu"	triggers when a node is clicked by right button
input$<id>_check_change	events = "check_change"	triggers when the selected state of the node changes
input$<id>_check	unasked	triggers after clicking the checkbox of a node
input$<id>_current_change	events = "current_change"	triggers when current node changes
input$<id>_node_expand	events = "node_expand"	triggers when current node open
input$<id>_node_collapse	events = "node_collapse"	triggers when current node close

The same list as el_events("el_tree_v2"), which says how an event's arguments travel.

Updating from the server

update_el_tree_v2() changes the component from the server: every argument of el_tree_v2() that can change once it is drawn, under the same name. One left NULL stays as it is; NA returns it to Element's default.

update_el_tree_v2() is called for its side effect and returns NULL invisibly.

Examples

```
el_tree_v2(
  "big",
  data = lapply(1:1000, function(i) list(id = i, label = paste("Node", i))),
  height = 300
)
```

el_upload *Element Plus Upload*

Description

A file upload area, as a button or a drop zone.

Usage

```
el_upload(
  id = NULL,
  button_label = "Upload",
  drag = FALSE,
  multiple = FALSE,
  accept = NULL,
  limit = NULL,
  show_file_list = TRUE,
  list_type = "text",
  auto_upload = TRUE,
  disabled = FALSE,
  name = NULL,
  tip = NULL,
  action = NULL,
  headers = NULL,
  data = NULL,
  file_list = NULL,
  with_credentials = NULL,
  before_upload = NULL,
  before_remove = NULL,
  on_change = NULL,
  on_progress = NULL,
  on_preview = NULL,
  on_remove = NULL,
  on_exceed = NULL,
  label = NULL,
  label_position = c("top", "left", "right"),
  label_width = NULL,
  label_suffix = NULL,
  required = FALSE,
  error = NULL,
  show_message = TRUE,
  inline_message = FALSE,
  crossorigin = NULL,
  directory = NULL,
  width = NULL,
  slots = NULL,
  on = NULL,
```

```
  session = NULL
)

update_el_upload(
  session = shiny::getDefaultReactiveDomain(),
  id,
  disabled = NULL,
  limit = NULL,
  label = NULL,
  error = NULL,
  button_label = NULL,
  drag = NULL,
  multiple = NULL,
  accept = NULL,
  show_file_list = NULL,
  list_type = NULL,
  auto_upload = NULL,
  headers = NULL,
  data = NULL,
  file_list = NULL,
  with_credentials = NULL,
  before_upload = NULL,
  before_remove = NULL,
  on_change = NULL,
  on_progress = NULL,
  on_preview = NULL,
  on_remove = NULL,
  on_exceed = NULL,
  crossorigin = NULL,
  directory = NULL
)
```

Arguments

id	Upload ID (auto-generated if NULL).
button_label	Text for the trigger, as buttonLabel is for shiny::fileInput(): on the button when drag = FALSE, inside the drop zone otherwise.
drag	Render a drop zone rather than a button.
multiple	Allow selecting several files at once.
accept	File types to accept, as an accept attribute would have them, e.g. ".csv,.tsv" or "image/*".
limit	Maximum number of files.
show_file_list	Show the list of chosen files.
list_type	"text" (default), "picture" or "picture-card".
auto_upload	Start uploading as soon as files are chosen.
disabled	Disable the control.

`name` Field name Element posts the file under. Only meaningful with `action`, where it names the multipart field; it defaults to `"file"` there. Without `action` the field has no name: Shiny's own file-input binding claims every `input[type=file]` on the page that has an id or a name, and would add an `input$<name>` of its own beside `input$<id>`.

`tip` Help text shown under the control.

`action` Post to this URL using Element's own upload instead of Shiny's channel. See details.

`headers` Request headers, as a named list.

`data` Extra fields sent alongside each file, as a named list: Element Plus's `data`. Element's own request sends them, so they apply only with `action`; Shiny's upload, the default, does not send them – give the server what it needs through another input.

`file_list` Files shown initially, each `list(name=, url=)`.

`with_credentials`
Whether to send cookies with the request.

`before_upload` `JS()` function called before a file is sent; returning `false` cancels it.

`before_remove` `JS()` function called before a file is removed; returning `false` cancels it.

`on_change` `JS()` function called when a file is added, or finishes.

`on_progress` `JS()` function called as a file uploads.

`on_preview` `JS()` function called when an uploaded file is clicked.

`on_remove` `JS()` function called after a file is removed.

`on_exceed` `JS()` function called when more files are picked than `limit`.

`label` A label shown with the component, as Shiny's inputs have: text or a tag. `NULL`, the default, shows none. It is the component's accessible name too – tied to it with `for` where the component has a native input that takes the id `<id>-input`, else with `aria-labelledby`.

`label_position` Where the label sits, as `el_form()`'s `label_position`: `"top"` (the default, as Shiny's labels sit), or beside the component, its text aligned `"left"` or `"right"` – which shows once `label_width` gives the labels a common width.

`label_width` Width of a label beside the component, as a CSS unit, so that several line up. Element's `label-width`.

`label_suffix` Text after the label, such as `":"`. Element's `label-suffix`.

`required` Draw Element's red asterisk before the label. It marks the field; it does not check it – shinyvalidate or `el_form()` does that.

`error` An error message shown under the component in Element's style, the field framed in red. Element's `error`.

`show_message, inline_message`
Whether `error`'s message is shown, and whether beside the component rather than under it. Element's `show-message` and `inline-message`.

`crossorigin` Native attribute crossorigin. Element Plus's `crossorigin` (" | 'anonymous' | 'use-credentials').

`directory` Whether to support uploading directory. After enabling it, only folders can be selected, and after selecting a folder, the files within the folder will be flattened. Element Plus's `directory` (boolean).

`width` Component width, as a CSS unit – `"200px"`, `"50%"`, or a number taken as pixels. Element's own markup carries it, so it behaves like the `width` argument of a Shiny input.

`slots` Named list of Element slot contents: `default` (the trigger, in place of the button), `tip`, `file` (each file of the list, scoped with `file`), `trigger`. A shiny.element component given here is absorbed rather than nested. For a scoped slot, write the template with `template()`.

`on` Handlers of your own, for an event not reported or to send something else: a named list of `JS()` functions, one per event – Element's, or a DOM event with Vue's modifiers (`"keyup.enter"`). Each is called with `report` and the event's arguments; `report(name, value)` sets `input$<id>_<name>`. See `el_widget()`.

`session` In `el_upload()`, deprecated: inside a module, wrap `id` in `ns()`, as for any Shiny input; a session given here namespaces `id` once more, with a warning. In `update_el_upload()`, the Shiny session, the current one by default, as for `shiny::updateTextInput()`.

Details

By default the files travel through Shiny's own upload channel, so `input$<id>` is the same data frame `shiny::fileInput()` produces, complete with a `datapath` pointing at a temporary file. Element's file list, progress bars and hooks all keep working: only the transport is replaced.

Give `action` instead to use Element's own upload, posting straight to that URL. Shiny then plays no part – useful for a pre-signed S3 URL or an existing file service, but the server sees no `datapath`.

Value

A Shiny UI element.

Shiny inputs

Input	Reported	Value
`input$<id>`	unasked	the files, as `fileInput()` reports them (without `action`)
`input$<id>_success`	unasked	with `action`, the names of the files uploaded
`input$<id>_error`	unasked	the name of a file that failed

The same list as `el_events("el_upload")`, which says how an event's arguments travel.

Without `action`, `input$<id>` is a data frame of `name`, `size`, `type` and `datapath`, one row per file in the last batch, exactly as `shiny::fileInput()` reports it. Uploads inherit Shiny's `shiny.maxRequestSize` limit and its temporary-file cleanup.

With `action`, Shiny never sees the files; `input$<id>_success` lists the names of files Element uploaded successfully.

Either way, input$<id>_error is the name of a file that failed, as an event. Without action, the rest of its batch is sent again without it, so input$<id> holds the files that arrived; a file stopped with abort() is left out the same way.

Files go up one at a time, as shiny::fileInput() sends them, and each is marked done once the whole batch has reached the server.

Element methods

Callable with call_el():

- abort() – Cancel upload request: one file, given as its uid, or every file in flight
- clearFiles() – Clear the uploaded file list (this method is not supported in the before-upload hook)
- submit() – Upload the file list manually

Updating from the server

update_el_upload() changes the component from the server.

Every other argument of el_upload() that can change once it is drawn is an argument here too, under the same name. One left NULL stays as it is; NA returns it to Element's default.

update_el_upload() is called for its side effect and returns NULL invisibly.

Examples

```
# A drop zone taking several CSVs, read on the server like fileInput()
el_upload(
  "files",
  drag = TRUE,
  multiple = TRUE,
  accept = ".csv",
  tip = "CSV files only"
)

# A plain button
el_upload("avatar", button_label = "Choose a picture", accept = "image/*")

# Element's own upload, straight to a pre-signed URL
el_upload("direct", action = "https://example.invalid/presigned")

if (interactive()) {
  library(shiny)
  library(shiny.element)
  ui <- el_page(
    el_upload("files", drag = TRUE, multiple = TRUE),
    tableOutput("info")
  )
  server <- function(input, output, session) {
    output$info <- renderTable({
      req(input$files)
      input$files[, c("name", "size", "type")]
```

```
    })
  }
  shinyApp(ui, server)
}
if (interactive()) {
  # inside a server function
  observeEvent(input$go, {
    update_el_upload(session, "files", disabled = TRUE)
  })
}
```

`el_upload_clear` *Clear an Element Plus Upload's file list*

Description

Empties the list of chosen files, as you would after a form is submitted. It does not undo an upload that has already happened.

Usage

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

Arguments

`session`	Shiny session; the current one by default, as for `shiny::updateTextInput()`.
`id`	Upload ID (un-namespaced).

Value

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

Examples

```
if (interactive()) {
  # inside a server function
  observeEvent(input$go, {
    el_upload_clear(session, "files")
  })
}
```

el_watermark *Element Plus Watermark*

Description

Text or an image repeated over its content, faintly.

Usage

```
el_watermark(
  ...,
  id = NULL,
  watermark_width = NULL,
  height = NULL,
  rotate = NULL,
  z_index = NULL,
  image = NULL,
  content = NULL,
  font = NULL,
  gap = NULL,
  offset = NULL,
  width = NULL,
  slots = NULL,
  on = NULL
)

update_el_watermark(
  session = shiny::getDefaultReactiveDomain(),
  id,
  watermark_width = NULL,
  height = NULL,
  rotate = NULL,
  z_index = NULL,
  image = NULL,
  content = NULL,
  font = NULL,
  gap = NULL,
  offset = NULL
)
```

Arguments

...	Its content: any Shiny UI. Components of this package are folded into this one's Vue instance, as el_button_group() folds its buttons.
id	Component ID. Auto-generated if NULL.

watermark_width
: The width of the watermark, the default value of `content` is its own width. Element Plus's `width` (number).

`height`
: The height of the watermark, the default value of `content` is its own height. Element Plus's `height` (number).

`rotate`
: When the watermark is drawn, the rotation angle, in degrees. Element Plus's `rotate` (number).

`z_index`
: The z-index of the appended watermark element. Element Plus's `z-index` (number).

`image`
: Image source, it is recommended to export 2x or 3x image, high priority. Element Plus's `image` (string).

`content`
: Watermark text content. Element Plus's `content` (`string` / `string[]`).

`font`
: Text style. Element Plus's `font`.

`gap`
: The spacing between watermarks. Element Plus's `gap` (`[number, number]`).

`offset`
: The offset of the watermark from the upper left corner of the container. The default is `gap/2`. Element Plus's `offset` (`[number, number]`).

`width`
: Component width, as a CSS unit.

`slots`
: Named list of Element slot contents. A scoped slot is written with `template()`.

`on`
: Handlers of your own, for an event not reported or to send something else: a named list of `JS()` functions, one per event – Element's, or a DOM event with Vue's modifiers (`"keyup.enter"`). Each is called with `report` and the event's arguments; `report(name, value)` sets `input$<id>_<name>`. See `el_widget()`.

`session`
: Shiny session; the current one by default, as for `shiny::updateTextInput()`.

Value

A Shiny UI element.

Shiny inputs

None: it reports nothing.

Updating from the server

`update_el_watermark()` changes the component from the server: every argument of `el_watermark()` that can change once it is drawn, under the same name. One left `NULL` stays as it is; `NA` returns it to Element's default.

`update_el_watermark()` is called for its side effect and returns `NULL` invisibly.

Examples

```
el_watermark(content = "Confidential", shiny::tags$div(style = "height: 300px"))
```

`el_widget` *Assemble a component: host, Vue instance, Shiny input binding*

Description

Every control in this package has the same shape, and this builds it: a host element carrying the id, the Element markup inside it, and the Vue options beside them, which the package's bridge script compiles in place. The host is a Shiny input binding, so it *is* the component to the rest of Shiny – `shinyjs::hide("id")` hides it, `removeUI("#id")` removes it and destroys its Vue instance, and its value is `input$<id>`.

Usage

```
el_widget(
  id,
  markup,
  data,
  methods = NULL,
  watch = NULL,
  mounted = NULL,
  computed = NULL,
  emits = NULL,
  on = NULL,
  dependency = NULL,
  head = NULL,
  width = NULL,
  slots = NULL,
  input = NULL,
  rate = NULL,
  type = NULL,
  label = NULL,
  label_position = c("top", "left", "right"),
  label_width = NULL,
  label_suffix = NULL,
  required = FALSE,
  error = NULL,
  show_message = TRUE,
  inline_message = FALSE,
  props = NULL,
  absorbed = NULL
)
```

Arguments

`id`	The element id – inside a module, wrapped in `ns()`. It is the input id of the value `input` names.
`markup`	The Element markup to mount on, usually one `htmltools::tag()`.

`data` The Vue instance's data. Every field that `update_el_*()` may set has to be declared here – Vue does not track one that is not.

`methods`, `watch`, `mounted`, `computed`
Vue options, included when not `NULL`.

`emits` Events the component sends with `$emit()`: each arrives as `input$<id>_<event>`, as for `vue_app()`.

`on` Handlers of your own, for events the component does not report or to send something else than they carry: a named list of `JS()` functions, one per event – Element's, by its name, or a DOM event of the element it draws, with Vue's modifiers (`"keyup.enter"`). Each is called with `report` first, then the event's arguments, `this` being the component's Vue instance; `report(name, value)` sets `input$<id>_<name>`, namespaced as the id is: `on = list("keyup.enter" = JS("function(report, e) { report('enter', e.target.value); }"))` reports `input$<id>_enter`. An event the component reports too runs both.

`dependency` htmlDependency objects to attach, beside Vue, Element Plus and the bridge, which every component carries.

`head` Tags to place before the host, such as a `<style>` block.

`width` Component width, as a CSS unit. Applied to the Element markup itself – the host carries `display: contents` and generates no box, so a width set on it would do nothing.

`slots` Named list of slot contents, one entry per Element slot: `list(title = tags$b("Bold"))` fills the `title` slot. A component given here is absorbed like any other (`.el_absorb()`). For a scoped slot, where Element hands the template its own data, write the template with `template()` and the value is used as it stands.

`input` The field of `data` that is `input$<id>`, or several for one value made of them, as `vue_app()`'s `input`: the Shiny binding reads it on load and on every change, and a test driver or `shinyjs` sees it. An `update_vue()` from the server counts as a change.

`rate` How often the value is sent while it changes: `list(policy = "debounce", delay = 250)`, as Shiny's `textInput()` does, or `"throttle"`. `NULL`, the default, sends every change.

`type` An input type for `shiny::registerInputHandler()`, which converts the value on its way into R.

`label` A label shown with the component, as Shiny's inputs have: text or a tag. `NULL`, the default, shows none. It is the component's accessible name too – tied to it with `for` where the component has a native input that takes the id `<id>-input`, else with `aria-labelledby`.

`label_position` Where the label sits, as `el_form()`'s `label_position`: `"top"` (the default, as Shiny's labels sit), or beside the component, its text aligned `"left"` or `"right"` – which shows once `label_width` gives the labels a common width.

`label_width` Width of a label beside the component, as a CSS unit, so that several line up. Element's `label-width`.

`label_suffix` Text after the label, such as `":"`. Element's `label-suffix`.

`required` Draw Element's red asterisk before the label. It marks the field; it does not check it – shinyvalidate or `el_form()` does that.

`error` An error message shown under the component in Element's style, the field framed in red. Element's `error`.

`show_message`, `inline_message`
Whether `error`'s message is shown, and whether beside the component rather than under it. Element's `show-message` and `inline-message`.

`props` Optional props from `.el_props()`: bound on the root tag of `markup`, with their fields added to `data`.

`absorbed` The components folded into this one, by id: their fields as named here and the ref on each, so their updates reach them. Built by the package's wrappers; leave it `NULL`.

Details

Reach for it to wrap an Element component this package does not cover, or to build an input of your own from `el` tags; `input` names the value. It is what the package's own components are made of: `vue_app()` with Element Plus installed (`use`), and a label in Element's form-item style.

The raw Element tags come from `el`, and `template()` writes a slot.

Value

A Shiny UI element with its dependencies attached.

Examples

```
# Wrapping el-avatar, which this package does not provide
my_avatar <- function(id, src, size = 50) {
  el_widget(
    id = id,
    markup = el$avatar(":src" = "src", ":size" = "size"),
    data = list(src = src, size = size)
  )
}
my_avatar("face", "https://example.org/face.png")

# An input of your own: v-model keeps `value` in step with the control,
# and `input` makes it input$score -- on load, on change, and after
# update_vue(session, "score", value = 5) from the server.
el_widget(
  id = "score",
  markup = el$rate("v-model" = "value", ":max" = "max"),
  data = list(value = 3, max = 5),
  input = "value"
)
```

`flush_vue` *Send queued updates now*

Description

Updates and method calls – `update_vue()`, `call_vue()`, every `update_el_*()` and `call_el()` – are sent as Shiny's own `update*Input()` are: with the flush, once the observer that made them, and the rest of the flush, is done. An update made before a long computation in the same observer therefore shows after it. `flush_vue()` sends them at once: wrapped around the updates, it runs them and sends what is queued; called alone, it sends what is queued so far.

Usage

```
flush_vue(expr = NULL, session = shiny::getDefaultReactiveDomain())
```

Arguments

`expr`	Code making updates, run before they are sent.
`session`	Shiny session; the current one by default.

Details

Often there is a better way: a long computation in an shiny::ExtendedTask leaves the observer at once, and the update goes with that flush. A component library may send some updates at once by their nature – a progress bar, as `shiny::withProgress()` does.

Only the components' messages are sent: outputs are still calculated with the flush, as Shiny does.

Value

The value of `expr`, invisibly.

Examples

```
if (interactive()) {
  # inside a server function
  observeEvent(input$run, {
    flush_vue({
      update_el_table(session, "tbl", loading = TRUE)
    })
    result <- slow_query()
    update_el_table(session, "tbl", data = result, loading = FALSE)
  })
}
```

`insert_el_tab` *Add or remove a tab from the server*

Description

Element's addable and editable tabs leave adding to the app: clicking "+" reports `input$<id>_tab_add`, and the app decides what the new tab holds. These are how, in the manner of `shiny::insertTab()` and `shiny::removeTab()`. The content may hold any UI, this package's components included.

Usage

```
insert_el_tab(
  session = shiny::getDefaultReactiveDomain(),
  id,
  tab,
  label = NULL,
  content = NULL,
  closable = NULL,
  select = TRUE
)

remove_el_tab(session = shiny::getDefaultReactiveDomain(), id, name)
```

Arguments

`session`	Shiny session; the current one by default, as for `shiny::updateTextInput()`.
`id`	Tabs ID (un-namespaced).
`tab`	The new tab: an `el_tab_pane()`, as `bslib::nav_insert()` takes a `nav_panel()` – or its name, with `label` and `content`.
`label, content`	The new tab's label and content, when `tab` is a name.
`closable`	Whether it can be closed. `NULL` follows the tabs' own setting.
`select`	Whether to switch to it. Default `TRUE`.
`name`	For `remove_el_tab()`, the name of the tab to remove.

Value

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

Examples

```
if (interactive()) {
  library(shiny)
  ui <- el_page(el_tabs(
    "docs",
    editable = TRUE,
    tabs = list(
```

```
      el_tab_pane("Tab 1", tags$p("First"), name = "t1")
    )
  ))
  server <- function(input, output, session) {
    n <- 1
    observeEvent(input$docs_tab_add, {
      n <<- n + 1
      insert_el_tab(
        session,
        "docs",
        el_tab_pane(paste("Tab", n), tags$p("New"), name = paste0("t", n))
      )
    })
  }
  shinyApp(ui, server)
}
```

JS *Mark a string as JavaScript*

Description

A prop that takes a function – a table's `formatter`, a tree's `filter_node_method`, a date picker's shortcuts – takes JavaScript source marked with `JS()`, which the page evaluates rather than passing on as text. It is the same mark `htmlwidgets::JS()` makes, so either works, and so does `DT::JS()`.

Usage

```
JS(...)
```

Arguments

`...`	JavaScript source, as one or more strings, joined by newlines.

Value

The source, classed `"JS_EVAL"`.

Examples

```
JS("function(row, column, value) { return value.toFixed(2); }")
```

render_vue_data *Data for a component, from the server*

Description

An output that sends a component's `data` – some of its fields, by name – rather than markup: the server renders `list(mean = 1, sd = 2)` and the fields `mean` and `sd` of every component following the output take those values; the template redraws what depends on them, and nothing else is touched. It is the declarative twin of `update_vue()`, with the same rule: a field must be declared in the component's `data` (or returned by `setup()`), as Vue tracks only the fields a component starts with; one it does not have is left alone, with a warning in the browser's console. Fields not in a render keep their values.

Usage

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

vue_data_output(outputId)
```

Arguments

`expr`	An expression returning the fields: a named list.
`env, quoted`	As for `shiny::renderText()`.
`outputId`	The output's id. Only needed to place the output by hand: a component listing it in `outputs` places it itself.

Details

The component names the output in its `outputs` (`vue_app(outputs = "stats")`, inside a module `ns("stats")`). It is an output like any other: rendered again when what it reads changes, held back while its component is hidden, an error shown as Shiny shows one, a promise – an `ExtendedTask`'s result – waited for. While it recalculates, `$recalculating.<id>` is `true` in the component's templates. It can be cached with `shiny::bindCache()`.

Values travel as `vue_app()`'s `data` does: a list an object, a data.frame its rows, `JS()` a function; `I()` keeps a vector of one an array.

Where `render_vue()` draws a component the server writes, this fills one the UI writes: the template stays where it is, the data comes from the server. One output can feed several components, or a `vue_store()` they all read – shared state, as Vue's guide recommends, with the server as its source. As shinyreact's `reactive_output()` does for React.

Value

`render_vue_data()`, a render function; `vue_data_output()`, a tag.

See Also

`vue_app()`, `vue_store()`, `update_vue()`.

Examples

```
if (interactive()) {
  library(shiny)
  ui <- fluidPage(
    sliderInput("n", "Draws", 10, 1000, 100),
    vue_app(
      "summary",
      template = htmltools::tags$p(
        `:style` = "{opacity: $recalculating.stats ? 0.5 : 1}",
        "Mean {{ mean }}, sd {{ sd }}"
      ),
      data = list(mean = NA, sd = NA),
      outputs = "stats"
    )
  )
  server <- function(input, output, session) {
    output$stats <- render_vue_data({
      x <- rnorm(input$n)
      list(mean = round(mean(x), 3), sd = round(sd(x), 3))
    })
  }
  shinyApp(ui, server)
}
```

---

template *Generate a* <template> *tag for Vue/Element Plus slot usage*

Description

Generate a <template> tag for Vue/Element Plus slot usage

Usage

```
template(..., slot = NULL, scope = NULL)
```

Arguments

...	Content inside the template
slot	Slot name (optional)
scope	Slot scope or v-slot syntax (optional)

Value

A Shiny UI element.

Examples

```
# Basic usage with text
template("Hello World")

# With slot name
template("Custom content", slot = "header")

# With HTML tag content
template(shiny::tags$a(href = "https://posit.co", "Posit"), slot = "footer")

# Combine multiple tags
template(
  shiny::tags$span("A"),
  shiny::tags$span("B"),
  slot = "extra"
)

# Custom dateCell slot with check mark (Unicode)
template(
  shiny::tags$p(
    `:class` = "data.isSelected ? 'is-selected' : ''",
    "{{ data.day.split('-').slice(1).join('-') }}",
    shiny::tags$span("\u2714\ufe0f", `v-if` = "data.isSelected")
  ),
  slot = "date-cell",
  scope = "{date, data}"
)
```

---

update_vue *Set fields of a Vue component from the server*

Description

The Vue layer's `update*Input()`: assigns fields of a component's state – its `data`, or what its `setup()` returned, or a `vue_store()`'s – and the component, and every template showing them, follow. As with Shiny's updaters, the component's value is reported back to `input$<id>` afterwards, and the update is sent with the flush, after the outputs (`flush_vue()`).

Usage

```
update_vue(
  session = shiny::getDefaultReactiveDomain(),
  id,
  ...,
  value = NULL,
  insert = NULL,
  replace = NULL,
  delete = NULL,
  at = NULL,
```

```
  key = NULL,
  set = NULL
)
```

Arguments

session	Shiny session; the current one by default, as for shiny::updateTextInput().
id	The component's id (un-namespaced).
...	Fields to set, <field> = <value>. A function is JS().
value	The component's value: sets whichever field its input names.
insert, replace, delete	Rows to insert, rows to put in place of others, or the rows to delete – each list(<field> = ...), one of them per call.
at	Positions, counting from 1: where insert goes, which rows replace and delete change.
key	A field the rows are found by, instead of positions.
set	Values at paths, list("<path>" = value).

Details

A field given in ... is replaced whole. A long list, or a deep object, can be changed in place instead, sending only what changes:

- insert, replace, delete: rows of a list field – one field per call, list(items = rows). Rows are a data.frame or a list of rows. at says where, counting from 1: the position insert goes before (the end when NULL), the positions replace and delete change. With key, rows are found by that field's value instead: replace puts each row in place of the one with its key (or at the end), and delete = list(items = c("a", "c")) takes away the rows with those keys.
- set: values at paths, list("items[3].done" = TRUE, "user.name" = "Ann") – fields, then list positions counting from 1 and names.

Value

Called for its side effect; returns NULL invisibly.

See Also

vue_app(), call_vue(), flush_vue().

Examples

```
if (interactive()) {
  # inside a server function
  update_vue(id = "counter", value = 0)
  update_vue(id = "cart", note = "Free delivery today")
  # a list changed in place: one row added, one ticked
  update_vue(id = "todo", insert = list(items = list(text = "Call Ann")))
  update_vue(id = "todo", set = list("items[2].done" = TRUE))
```

```
  update_vue(id = "todo", delete = list(items = "t3"), key = "id")
}
```

`use_element` *Element Plus for a page of your own*

Description

What `el_page()` gives a page, for other page functions – `bslib::page_sidebar()`, `shiny::navbarPage()`: the theme Element follows, its locale and global config, the handlers of `el_message()` and the other feedback services, and Element served from the CDN when `offline = FALSE`. Components need none of it to work: each carries Vue and Element Plus, so on a page without `use_element()` they show in English, Element's own colours and sizes. Place it before the components, so its copy of Element is the one kept.

Usage

```
use_element(
  theme = NULL,
  offline = TRUE,
  dev = .vue_dev(),
  locale = getOption("shiny.element.locale", "en"),
  size = NULL,
  z_index = NULL,
  layout_css = el_layout_css_dependency()
)
```

Arguments

`theme`	The page's theme – an `el_theme()` or any `bslib::bs_theme()` – for Element's components to follow: its colours, and any Element variable given to `el_theme()`'s `element`. It styles Element only; the page function that owns the page applies it to Bootstrap. `NULL` leaves Element as it ships.
`offline`	Serve Element Plus from the copy bundled with this package rather than the unpkg CDN. See `element_plus_dependency()`.
`dev`	Load Vue's development build (`vue.global.js`) instead of the production one, so Vue's warnings are not stripped. Defaults to `getOption("shiny.vue.dev")` (or `shiny.element.dev`), `FALSE` unless set. Installed from CRAN, the development build is loaded from the unpkg CDN.
`locale`	Language for Element Plus's built-in text. English by default, or `getOption("shiny.element.locale")` when set. See `el_locale_dependency()`.
`size, z_index`	Element Plus's global config, as `app.use()` gives it: the size of every component not given one of its own (`"large"`, `"default"` or `"small"`), and the z-index its popups start from (2000 by default). `NULL` leaves Element's default.
`layout_css`	Element's layout CSS, `el_layout_css_dependency()`; `NULL` leaves it out.

Value

A list of htmlDependency objects

Examples

```
theme <- el_theme(primary = "#7c3aed")
ui <- bslib::page_sidebar(
  theme = theme,
  use_element(theme = theme),
  el_button("btn1", "Click me")
)
```

`vue_answer` *Answer a Vue component that asked the server*

Description

A component that needs the server – a lazy tree its children, a remote search its matches – asks with `shinyVue.ask(input, question)` in the browser: the question arrives as an input, with a `request` number, and the component waits for the answer. This sends it, or refuses (`failed`), which rejects the component's promise.

Usage

```
vue_answer(
  session = shiny::getDefaultReactiveDomain(),
  id,
  request,
  value = NULL,
  failed = FALSE
)
```

Arguments

`session`	Shiny session; the current one by default, as for `shiny::updateTextInput()`.
`id`	The component's id (un-namespaced).
`request`	The question as it arrived, or its `request` number.
`value`	The answer.
`failed`	`TRUE` to refuse the request instead.

Value

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

See Also

`vue_app()`.

Examples

```
if (interactive()) {
  # inside a server function: a component asked input$places_query
  observeEvent(input$places_query, {
    q <- input$places_query
    vue_answer(
      id = "places",
      request = q,
      value = grep(q$text, cities, value = TRUE)
    )
  })
}
```

---

vue_app *A Vue component, as Shiny UI*

Description

vue_app() writes a Vue 3 component in R and places it on the page: the arguments are Vue's own options, under Vue's own names, and the browser creates and mounts it with Vue.createApp(). Three things are added, as Shiny needs them: id, where it goes and what the server calls it; input, the field that is input$<id>; and dependencies. use is Vue's app.use().

Usage

```
vue_app(
  id,
  template,
  data = list(),
  methods = NULL,
  computed = NULL,
  watch = NULL,
  emits = NULL,
  setup = NULL,
  components = NULL,
  ...,
  input = NULL,
  type = NULL,
  rate = NULL,
  outputs = NULL,
  use = NULL,
  dependencies = NULL,
  events = NULL,
  on = NULL
)
```

Arguments

id	The component's id: input$<id>, and what update_vue() and call_vue() name. Inside a module, ns("id").
template	The template: htmltools tags, or a string.
data	Named list: the initial state. A data.frame in it is rows. A vector of one element travels as a single value, as in Shiny; wrap it in I() to keep it an array: tags = I("red").
methods, computed, watch	Named lists of JS() functions.
emits	Names of the events the component sends with $emit().
setup	A JS() function: Vue's Composition API.
components	Named list of vue_component()s, for the template.
...	Any other option of Vue's, by Vue's name or its snake_case: mounted = JS(...), before_unmount = JS(...), provide = JS(...).
input	The field whose value is input$<id>, or several, for one value made of them. NULL: the component reports no value.
type	An input handler for the value, as Shiny's inputs have: the name given to shiny::registerInputHandler(), which converts the value on its way into R – "shiny.date" makes a "2026-01-31" a Date.
rate	How often the value is sent while it changes: "debounce" or "throttle" (250 ms), or list(policy = "debounce", delay = 500), as Shiny's textInput() debounces. NULL sends every change.
outputs	Ids of outputs the component's data follows, rendered with render_vue_data(): each render sets the fields it names, which must be in data (or setup()'s state). Inside a module, ns("stats"). While Shiny recalculates one, $recalculating.<id> is true in templates.
use	Vue plugins to install: by the global name each is loaded under, "MyPlugin", or with options, list(MyPlugin = list(...)); or written here, JS("{ install(app) { app.config.errorHandler = ... } }") – the way to reach the app's own configuration.
dependencies	htmltools::htmlDependency()s the component needs: the plugins' scripts and stylesheets.
events	Events of the template's root – a library's component there, or a DOM event – each reported as input$<id>_<event>, by its name in snake_case or Vue's: events = "row_click". Arguments travel as emits' do – one as itself, several as list(arg1, arg2, ...), a key pressed as list(key, code, ctrl, shift, alt, meta), DOM objects dropped. The template must then be one tag.
on	Handlers of your own for events of the template's root: a named list of JS() functions, each called with report and the event's arguments; report(name, value) sets input$<id>_<name>.

Value

A tag, with its dependencies.

What goes where

Vue	Here
`createApp(options).mount('#app')`	`vue_app(id, ...)`
`data()`, `methods`, `computed`, `watch`, `setup()`, `emits`, hooks	the same names; functions as `JS()`
`beforeUnmount`, `inheritAttrs`, ...	Vue's name, or snake_case: `before_unmount`
`app.use(Plugin, options)`	`use = list(Plugin = options)`
`v-model` on the component (its value)	`input = "<field>": input$<id>`
`this.$emit("picked", x)`	`input$<id>_picked`, for an event in `emits`

Every function – a method, a watcher, `setup()` – is JavaScript, given with `JS()`. `data` is the initial state, as R writes it: a list becomes an object, a data.frame its rows. After that the component changes it, and the server does with `update_vue()` or `render_vue()`.

Shiny inputs

- `input$<id>` – the `input` field, on load, on change and after an update; several fields give one value, a named list of them.
- `input$<id>_<event>` – each event in `emits`, sent with `$emit()`: one argument as it is, several as a list (`arg1`, `arg2`, ...), none as `TRUE`.

Data from users

Show it through `data` – `{{ field }}`, `:prop="field"` – which Vue renders as text. Never paste it into the template: a template is code, and `{{ }}` in it runs.

See Also

`vue_component()`, `vue_store()`, `update_vue()`, `call_vue()`, `render_vue()`.

Examples

```
counter <- vue_app(
  "counter",
  template = htmltools::tags$button(`@click` = "n++", "Clicked {{ n }} times"),
  data = list(n = 0),
  input = "n"
)
if (interactive()) {
  library(shiny)
  shinyApp(
    fluidPage(counter, textOutput("n")),
    function(input, output) output$n <- renderText(input$counter)
  )
}
```

vue_component *A Vue child component, for a template to use*

Description

Vue's component options for one component that vue_app() registers under components =: components = list(todo_item = vue_component(...)) is <todo-item> in the template, and <todo_item> as written. As in Vue, a child takes props and sends emits to its parent's template (@toggle="..."); data gives each instance its own copy.

Usage

```
vue_component(
  template,
  props = NULL,
  emits = NULL,
  data = NULL,
  methods = NULL,
  computed = NULL,
  watch = NULL,
  setup = NULL,
  ...,
  dependencies = NULL
)
```

Arguments

template	The child's template: tags or a string.
props	Names of its props, or Vue's object form as a list.
emits	Names of the events it sends with $emit().
data	Named list: each instance's initial state.
methods, computed, watch	Named lists of JS() functions.
setup	A JS() function: Vue's Composition API.
...	Any other option of Vue's, by Vue's name or its snake_case – components for children of its own.
dependencies	htmltools::htmlDependency()s the child needs; the vue_app() that registers it attaches them, with those its template carries.

Value

A list of Vue options, of class vue_component.

Examples

```
item <- vue_component(
  template = "<li @click=\"$emit('toggle')\">{{ text }}</li>",
  props = "text",
  emits = "toggle"
)
```

---

vue_output *A Vue output that keeps the user's state across renders*

Description

`vue_output()` and `render_vue()` are the Vue layer's pair, as `shiny::uiOutput()` and `shiny::renderUI()` are Shiny's, with one difference. `renderUI()` replaces what it drew on every render, and with it whatever the user had done: a table's sort, a tree's open nodes, the tab they were on. `render_vue()` applies a render as a change, comparing the server's last render, its new one and the page:

Usage

```
vue_output(id)

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

Arguments

`id`	Output id.
`expr`	An expression returning UI: one component or several, with any markup around them.
`env, quoted`	As for `shiny::renderUI()`.

Details

- markup the server did not change stays as the page has it;
- text and attributes the server changed are set;
- each component whose template and options are unchanged gets only the `data` fields the server changed since its last render – so a field the user changed and the server did not keeps the user's value.

Anything structural – an element added or removed, a component's template or options changed, a component given another id – renders afresh, as `renderUI()` does. (A component given no id draws a random one each render; that is not a change.) To keep a component, put what changes from render to render in its `data` and keep its template the same.

A field the server sends with the same value as last time is not sent again, so it does not undo what the user did; to set a value whatever the user did, use `update_vue()`.

Value

`vue_output()`, a tag; `render_vue()`, a render function.

render_vue(), render_vue_data(), update_vue()

Three ways the server shapes a component, by who owns it:

- `render_vue()` – the server writes the component: template, options, methods, data, dependencies. Rendered again, it keeps what the user did; a new template or new methods mount it afresh. As shiny.react's `renderReact()` is for React.
- `render_vue_data()` – the component is written in the UI and one of its fields follows a value the server renders: a value, not markup, for components whose structure is fixed. One output can feed several components, or a `vue_store()` they share.
- `update_vue()` – an observer sets fields when it decides to: an imperative change, sent whether or not the component is shown.

`render_vue()` can be cached with `shiny::bindCache()`, as `renderUI()` can.

See Also

`vue_app()`, `update_vue()`, `render_vue_data()`.

Examples

```
if (interactive()) {
  library(shiny)
  ui <- fluidPage(
    sliderInput("n", "Rows", 1, 10, 5),
    vue_output("list")
  )
  server <- function(input, output, session) {
    output$list <- render_vue(vue_app(
      "items",
      template = htmltools::tags$ul(htmltools::tags$li(
        `v-for` = "i in items",
        "{{ i }}"
      )),
      data = list(items = as.list(seq_len(input$n)))
    ))
  }
  shinyApp(ui, server)
}
```

vue_store *State shared by Vue components: a store*

Description

Each vue_app() is an application of its own, so Vue's provide and inject cannot reach from one to another. Vue's guide answers shared state with a store, one reactive() object every component refers to, and this is that: every template reads and writes it as $store.<id>.<field>, at once and in the browser. It is also a component of the bridge, so the server reaches it the way it reaches any other: input makes fields input$<id>, update_vue() sets them – and every template showing them follows – and a bookmark restores them.

Usage

```
vue_store(id, data = list(), input = NULL, outputs = NULL)
```

Arguments

id	The store's id: $store.<id> in templates.
data	Named list: the initial state.
input	Fields reported as input$<id>, as for vue_app().
outputs	Outputs the store's data follows, as for vue_app().

Details

A store can be anywhere on the page; stores are set up before the components that read them.

Value

A tag: a hidden host that holds the store.

Examples

```
vue_store("cart", data = list(count = 0), input = "count")
vue_app("add", htmltools::tags$button(`@click` = "$store.cart.count++", "Add"))
vue_app("show", htmltools::tags$span("{{ $store.cart.count }} in the cart"))
```

Index

∗ **items**
    el_anchor_link, 18
    el_breadcrumb_item, 35
    el_carousel_item, 52
    el_collapse_item, 74
    el_descriptions_item, 98
    el_dropdown_item, 111
    el_menu_item, 169
    el_option, 178
    el_skeleton_item, 230
    el_step, 244
    el_tab_pane, 276
    el_table_column, 261
    el_table_v2_column, 272
    el_timeline_item, 285
    el_tour_step, 299
.el_absorb(), *327*

bsicons::bs_icon(), *127*
bslib::bind_task_button(), *37*
bslib::bs_theme(), *180*, *281*, *336*
bslib::input_task_button(), *37*
bslib::nav_insert(), *330*
bslib::page_sidebar(), *336*

call_el, 5
call_el(), *13*, *17*, *24*, *51*, *58*, *62*, *81*, *90*, *102*, *106*, *120*, *134*, *141*, *145*, *149*, *153*, *168*, *210*, *219*, *227*, *250*, *259*, *265*, *266*, *271*, *291*, *303*, *309*, *314*, *322*
call_vue, 7
call_vue(), *5*, *329*, *335*, *339*, *340*

df_to_cascader_options, 8
df_to_cascader_options(), *55*, *60*
df_to_tree_data, 9

el, 9, *328*
el_affix, 11
el_affix(), *13*
el_alert, 13
el_alert(), *15*
el_anchor, 16
el_anchor(), *17*, *18*
el_anchor_link, 18
el_anchor_link(), *16*, *35*, *52*, *75*, *99*, *112*, *171*, *179*, *230*, *244*, *263*, *273*, *277*, *285*, *300*
el_aside, 19
el_aside(), *84*
el_autocomplete, 20
el_autocomplete(), *24*
el_avatar, 24
el_avatar(), *26*
el_avatar_group, 27
el_avatar_group(), *28*
el_backtop, 29
el_backtop(), *30*
el_badge, 31
el_badge(), *32*
el_breadcrumb, 33
el_breadcrumb(), *34*, *35*
el_breadcrumb_item, 35
el_breadcrumb_item(), *18*, *33*, *52*, *75*, *99*, *112*, *171*, *179*, *230*, *244*, *263*, *273*, *277*, *285*, *300*
el_button, 36
el_button(), *10*, *39*, *40*, *175*, *187*
el_button_group, 39
el_button_group(), *12*, *27*, *40*, *83*, *209*, *236*, *238*, *239*, *280*, *324*
el_calendar, 41
el_calendar(), *44*, *47*
el_calendar_events, 46
el_calendar_events(), *47*
el_calendar_output, 47
el_card, 48
el_carousel, 49
el_carousel(), *51*, *52*

el_carousel_item, 52
el_carousel_item(), *18*, *35*, *50*, *75*, *99*, *112*, *171*, *179*, *230*, *244*, *263*, *273*, *277*, *285*, *300*
el_cascader, 53
el_cascader(), *58*, *60–62*
el_cascader_panel, 59
el_cascader_panel(), *62*
el_check_tag, 69
el_check_tag(), *71*
el_checkbox, 63
el_checkbox(), *65*
el_checkbox_group, 66
el_checkbox_group(), *63*, *69*
el_col, 71
el_col(), *205*
el_collapse, 72
el_collapse(), *10*, *74*, *102*
el_collapse_item, 74
el_collapse_item(), *18*, *35*, *52*, *73*, *99*, *112*, *171*, *179*, *230*, *244*, *263*, *273*, *277*, *285*, *300*
el_color_picker, 75
el_color_picker(), *78*
el_color_picker_panel, 79
el_color_picker_panel(), *81*
el_config_provider, 82
el_config_provider(), *83*
el_config_provider(button =), *37*
el_config_provider(card =), *48*
el_config_provider(dialog =), *101*
el_config_provider(link =), *155*
el_container, 84
el_container(), *10*
el_countdown *(*el_statistic*)*, 240
el_countdown(), *243*
el_date_picker, 85
el_date_picker(), *91*
el_date_picker_panel, 92
el_date_picker_panel(), *95*
el_descriptions, 95
el_descriptions(), *97*, *98*
el_descriptions_item, 98
el_descriptions_item(), *18*, *35*, *52*, *75*, *96*, *112*, *171*, *179*, *230*, *244*, *263*, *273*, *277*, *285*, *300*
el_dialog, 99
el_dialog(), *10*, *102*
el_divider, 103
el_drawer, 104
el_drawer(), *10*, *107*
el_dropdown, 107
el_dropdown(), *111*
el_dropdown_item, 111
el_dropdown_item(), *18*, *35*, *52*, *75*, *99*, *109*, *171*, *179*, *230*, *244*, *263*, *273*, *277*, *285*, *300*
el_empty, 112
el_empty(), *113*
el_events, 114
el_events(), *12*, *17*, *23*, *26*, *30*, *43*, *57*, *61*, *78*, *90*, *94*, *101*, *106*, *110*, *119*, *130*, *133*, *140*, *145*, *148*, *152*, *164*, *168*, *181*, *185*, *192*, *209*, *219*, *227*, *234*, *238*, *242*, *246*, *257*, *265*, *266*, *270*, *275*, *290*, *295*, *298*, *303*, *308*, *313*, *316*
el_feedback_close, 115
el_feedback_dependency, 116
el_footer, 116
el_footer(), *84*
el_form, 117
el_form(), *22*, *42*, *56*, *61*, *68*, *77*, *80*, *88*, *94*, *120*, *124*, *125*, *140*, *144*, *148*, *152*, *164*, *197*, *202*, *212*, *217*, *226*, *233*, *249*, *290*, *302*, *307*, *313*, *320*, *327*
el_form_clear_validate, 122
el_form_field, 122
el_form_field(), *118*, *119*, *124*, *207*
el_form_item, 124
el_form_reset, 125
el_form_validate, 126
el_form_validate(), *118*
el_header, 126
el_header(), *84*
el_icon, 127
el_icon(), *37*
el_image, 128
el_image(), *131*
el_image_viewer, 132
el_image_viewer(), *134*
el_infinite_scroll, 134
el_infinite_scroll(), *136*
el_input, 137
el_input(), *141*, *164*
el_input_number, 142

el_input_number(), *146*
el_input_otp, 146
el_input_otp(), *149*
el_input_tag, 149
el_input_tag(), *153*
el_layout_css_dependency, 154
el_layout_css_dependency(), *180*, *336*
el_link, 154
el_link(), *156*
el_load_children, 158
el_load_children(), *58*, *258*, *308*, *312*
el_loading, 156
el_loading(), *157*, *256*
el_loading_close, 157
el_loading_close(), *156*
el_locale_dependency, 160
el_locale_dependency(), *83*, *336*
el_locales, 159
el_locales(), *180*
el_main, 161
el_main(), *84*
el_mention, 162
el_mention(), *165*
el_menu, 165
el_menu(), *169*
el_menu_item, 169
el_menu_item(), *18*, *35*, *52*, *75*, *99*, *112*, *167*, *179*, *230*, *244*, *263*, *273*, *277*, *285*, *300*
el_menu_item_group *(*el_menu_item*)*, 169
el_menu_item_group(), *167*
el_message, 171
el_message(), *116*, *175*, *336*
el_message_box, 173
el_message_close *(*el_feedback_close*)*, 115
el_message_close(), *172*
el_notification, 176
el_notification(), *116*, *175*
el_notification_close *(*el_feedback_close*)*, 115
el_notification_close(), *177*
el_option, 178
el_option(), *18*, *35*, *52*, *67*, *75*, *99*, *112*, *171*, *197*, *216*, *230*, *244*, *263*, *273*, *277*, *285*, *300*
el_option_group *(*el_option*)*, 178
el_option_group(), *216*
el_page, 179
el_page(), *116*, *160*, *281*, *282*, *336*
el_page_header, 181
el_page_header(), *182*
el_pagination, 183
el_pagination(), *186*
el_popconfirm, 186
el_popconfirm(), *10*, *188*
el_popover, 189
el_popover(), *10*, *188*, *192*
el_progress, 193
el_progress(), *195*
el_radio_group, 196
el_radio_group(), *198*
el_rate, 199
el_rate(), *203*
el_result, 203
el_result(), *204*
el_row, 205
el_row(), *10*, *71*, *124*
el_rule, 206
el_rule(), *119*, *123*, *124*
el_scrollbar, 207
el_scrollbar(), *210*
el_segmented, 210
el_segmented(), *212*
el_select, 213
el_select(), *123*, *178*, *220*, *310*
el_select_v2, 221
el_select_v2(), *227*
el_skeleton, 228
el_skeleton(), *229*, *230*
el_skeleton_item, 230
el_skeleton_item(), *18*, *35*, *52*, *75*, *99*, *112*, *171*, *179*, *229*, *244*, *263*, *273*, *277*, *285*, *300*
el_slider, 231
el_slider(), *234*
el_space, 235
el_space(), *237*
el_splitter, 237
el_splitter(), *238–240*
el_splitter_panel, 239
el_splitter_panel(), *237*
el_statistic, 240
el_statistic(), *243*
el_step, 244
el_step(), *18*, *35*, *52*, *75*, *99*, *112*, *171*, *179*,

230, *245*, *263*, *273*, *277*, *285*, *300*
el_steps, 245
el_steps(), *244*, *246*
el_sub_menu *(*el_menu_item*)*, 169
el_sub_menu(), *167*
el_switch, 247
el_switch(), *251*
el_tab_pane, 276
el_tab_pane(), *18*, *35*, *52*, *75*, *99*, *112*, *171*, *179*, *230*, *244*, *263*, *273*, *274*, *285*, *300*, *330*
el_table, 251
el_table(), *10*, *259*, *261*, *262*, *265*, *266*
el_table_column, 261
el_table_column(), *18*, *35*, *52*, *75*, *99*, *112*, *171*, *179*, *230*, *244*, *254*, *273*, *277*, *285*, *300*
el_table_data, 264
el_table_data(), *258*, *259*, *262*, *265*
el_table_output, 265
el_table_output(), *257*, *259*
el_table_row, 266
el_table_row(), *5*
el_table_v2, 267
el_table_v2(), *271*, *272*
el_table_v2_column, 272
el_table_v2_column(), *18*, *35*, *52*, *75*, *99*, *112*, *171*, *179*, *230*, *244*, *263*, *269*, *277*, *285*, *300*
el_tabs, 273
el_tabs(), *10*, *102*, *275*, *276*
el_tag, 277
el_tag(), *10*, *279*
el_text, 279
el_text(), *280*
el_theme, 281
el_theme(), *180*, *336*
el_time_picker, 286
el_time_picker(), *291*
el_time_select *(*el_time_picker*)*, 286
el_time_select(), *291*, *292*
el_timeline, 282
el_timeline(), *284*, *285*
el_timeline_item, 285
el_timeline_item(), *18*, *35*, *52*, *75*, *99*, *112*, *171*, *179*, *230*, *244*, *263*, *273*, *277*, *283*, *300*
el_tooltip, 292
el_tooltip(), *10*, *32*, *40*, *296*
el_tour, 296
el_tour(), *298*, *299*
el_tour_step, 299
el_tour_step(), *18*, *35*, *52*, *75*, *99*, *112*, *171*, *179*, *230*, *244*, *263*, *273*, *277*, *285*, *297*
el_transfer, 300
el_transfer(), *303*
el_tree, 304
el_tree(), *9*, *309*, *310*
el_tree_node *(*el_table_row*)*, 266
el_tree_select, 310
el_tree_select(), *314*
el_tree_v2, 314
el_tree_v2(), *317*
el_upload, 318
el_upload(), *322*
el_upload_clear, 323
el_upload_file *(*el_table_row*)*, 266
el_upload_file(), *5*
el_watermark, 324
el_watermark(), *325*
el_widget, 326
el_widget(), *10*, *13*, *15*, *17*, *23*, *26*, *28*, *30*, *32*, *33*, *38*, *40*, *50*, *57*, *61*, *65*, *68*, *70*, *78*, *81*, *90*, *94*, *97*, *110*, *113*, *114*, *119*, *131*, *133*, *135*, *141*, *145*, *148*, *152*, *155*, *164*, *168*, *182*, *185*, *188*, *192*, *194*, *198*, *202*, *204*, *209*, *212*, *219*, *227*, *229*, *234*, *236*, *238*, *240*, *242*, *246*, *250*, *257*, *270*, *278*, *280*, *283*, *290*, *295*, *298*, *303*, *308*, *313*, *317*, *321*, *325*
element_plus_dependency, 11
element_plus_dependency(), *180*, *336*

flush_vue, 329
flush_vue(), *195*, *334*, *335*
fontawesome::fa_i(), *128*

htmltools::htmlDependency(), *339*, *341*

insert_el_tab, 330
insert_el_tab(), *274*

JS, 331
JS(), *13*, *15*, *17*, *22*, *23*, *26*, *28*, *30*, *32*, *33*, *38*, *40*, *43*, *50*, *57*, *61*, *65*, *68*, *70*, *73*, *78*,

81, *83*, *89, 90*, *94*, *97*, *110*, *113*, *119*, *131*, *133*, *135*, *141*, *145*, *147, 148*, *152*, *155*, *157*, *163, 164*, *168*, *172*, *174*, *177*, *182*, *185*, *188*, *192*, *194*, *198*, *202*, *204*, *207*, *209*, *212*, *218, 219*, *224–227*, *229*, *234*, *236*, *238*, *240*, *242*, *246*, *250*, *257*, *262, 263*, *269, 270*, *273*, *278*, *280*, *283*, *290*, *295*, *298*, *303*, *306*, *308*, *312, 313*, *316, 317*, *321*, *325*, *327*, *332*, *335*, *339–341*

remove_el_tab *(*insert_el_tab*)*, 330
render_el_calendar *(*el_calendar_output*)*, 47
render_el_calendar(), *46*
render_el_table *(*el_table_output*)*, 265
render_el_table(), *257*, *259*, *264*
render_vue *(*vue_output*)*, 342
render_vue(), *265*, *332*, *340*
render_vue_data, 332
render_vue_data(), *339*, *343*

shiny::actionButton(), *38*, *279*
shiny::actionLink(), *155*
shiny::bindCache(), *265*, *332*, *343*
shiny::checkboxInput(), *63, 64*
shiny::dateInput(), *90*
shiny::ExtendedTask, *37*, *329*
shiny::fileInput(), *319*, *321, 322*
shiny::icon(), *127*
shiny::insertTab(), *330*
shiny::navbarPage(), *336*
shiny::registerInputHandler(), *327*, *339*
shiny::removeTab(), *330*
shiny::renderText(), *332*
shiny::renderUI(), *47*, *180*, *265*, *342*
shiny::tabsetPanel(), *274*
shiny::textInput(), *141*
shiny::uiOutput(), *136*, *342*
shiny::updateActionLink(), *155*
shiny::updateCheckboxInput(), *70*
shiny::updateSelectInput(), *259*, *313*
shiny::updateTextInput(), *5*, *7*, *13*, *15*, *17*, *23*, *26*, *28*, *30*, *32*, *34*, *38*, *40*, *50*, *57*, *61*, *65*, *68*, *73*, *78*, *81*, *83*, *90*, *94*, *97*, *101*, *106*, *110*, *113*, *115*, *119*, *122*, *125, 126*, *131*, *133*, *135*, *141*, *145*, *148*, *152*, *156*, *158*, *164*, *168*, *172*, *174*, *177*, *182*, *185*, *188*, *192*, *195*, *198*, *202*, *204*, *209*, *212*, *219*, *227*, *229*, *234*, *236*, *238*, *240*, *242*, *246*, *250*, *257*, *270*, *275*, *278*, *280*, *283*, *290*, *295*, *298*, *303*, *308*, *317*, *321*, *323*, *325*, *330*, *335*, *337*
shiny::useBusyIndicators(), *265*
shiny::withProgress(), *195*, *329*

template, 333
template(), *10*, *12*, *15*, *23*, *25*, *28*, *30*, *33*, *38*, *43*, *50*, *57*, *61*, *68*, *78*, *81*, *83*, *89*, *94*, *110*, *119*, *130*, *133*, *135*, *140*, *145*, *148*, *152*, *164*, *168*, *181*, *185*, *188*, *192*, *194*, *198*, *202*, *209*, *212*, *217*, *227*, *234*, *236*, *238*, *240*, *246*, *250*, *256*, *270*, *278*, *280*, *283*, *295*, *303*, *307*, *316*, *321*, *325*, *327, 328*

update_el_affix *(*el_affix*)*, 11
update_el_alert *(*el_alert*)*, 13
update_el_anchor *(*el_anchor*)*, 16
update_el_autocomplete *(*el_autocomplete*)*, 20
update_el_autocomplete(), *21*
update_el_avatar *(*el_avatar*)*, 24
update_el_avatar_group *(*el_avatar_group*)*, 27
update_el_backtop *(*el_backtop*)*, 29
update_el_badge *(*el_badge*)*, 31
update_el_badge(), *32*
update_el_breadcrumb *(*el_breadcrumb*)*, 33
update_el_button *(*el_button*)*, 36
update_el_button(), *37*
update_el_button_group *(*el_button_group*)*, 39
update_el_calendar *(*el_calendar*)*, 41
update_el_calendar(), *44*, *46, 47*
update_el_carousel *(*el_carousel*)*, 49
update_el_cascader *(*el_cascader*)*, 53
update_el_cascader_panel *(*el_cascader_panel*)*, 59
update_el_check_tag *(*el_check_tag*)*, 69
update_el_checkbox *(*el_checkbox*)*, 63
update_el_checkbox_group *(*el_checkbox_group*)*, 66
update_el_collapse *(*el_collapse*)*, 72
update_el_color_picker *(*el_color_picker*)*, 75

update_el_color_picker_panel (el_color_picker_panel), 79
update_el_config_provider (el_config_provider), 82
update_el_countdown (el_statistic), 240
update_el_date_picker (el_date_picker), 85
update_el_date_picker_panel (el_date_picker_panel), 92
update_el_descriptions (el_descriptions), 95
update_el_dialog (el_dialog), 99
update_el_drawer (el_drawer), 104
update_el_dropdown (el_dropdown), 107
update_el_empty (el_empty), 112
update_el_form (el_form), 117
update_el_form(), *123*, *207*
update_el_image (el_image), 128
update_el_image_viewer (el_image_viewer), 132
update_el_image_viewer(), *133*
update_el_infinite_scroll (el_infinite_scroll), 134
update_el_input (el_input), 137
update_el_input(), *141*
update_el_input_number (el_input_number), 142
update_el_input_otp (el_input_otp), 146
update_el_input_tag (el_input_tag), 149
update_el_link (el_link), 154
update_el_link(), *155*
update_el_mention (el_mention), 162
update_el_menu (el_menu), 165
update_el_page_header (el_page_header), 181
update_el_pagination (el_pagination), 183
update_el_popconfirm (el_popconfirm), 186
update_el_popover (el_popover), 189
update_el_progress (el_progress), 193
update_el_progress(), *193*
update_el_radio_group (el_radio_group), 196
update_el_rate (el_rate), 199
update_el_result (el_result), 203
update_el_scrollbar (el_scrollbar), 207
update_el_segmented (el_segmented), 210
update_el_select (el_select), 213
update_el_select(), *219*
update_el_select_v2 (el_select_v2), 221
update_el_skeleton (el_skeleton), 228
update_el_slider (el_slider), 231
update_el_space (el_space), 235
update_el_splitter (el_splitter), 237
update_el_splitter_panel (el_splitter_panel), 239
update_el_statistic (el_statistic), 240
update_el_steps (el_steps), 245
update_el_switch (el_switch), 247
update_el_table (el_table), 251
update_el_table(), *5*, *256*, *258*, *261*, *264*, *265*
update_el_table_v2 (el_table_v2), 267
update_el_table_v2(), *272*
update_el_tabs (el_tabs), 273
update_el_tabs(), *276*
update_el_tag (el_tag), 277
update_el_text (el_text), 279
update_el_time_picker (el_time_picker), 286
update_el_time_select (el_time_picker), 286
update_el_timeline (el_timeline), 282
update_el_timeline(), *283*
update_el_tooltip (el_tooltip), 292
update_el_tour (el_tour), 296
update_el_tour(), *297*
update_el_transfer (el_transfer), 300
update_el_tree (el_tree), 304
update_el_tree(), *306*
update_el_tree_select (el_tree_select), 310
update_el_tree_v2 (el_tree_v2), 314
update_el_upload (el_upload), 318
update_el_watermark (el_watermark), 324
update_vue, 334
update_vue(), *7*, *8*, *32*, *40*, *295*, *327*, *329*, *332*, *339*, *340*, *342–344*
use_element, 336
use_element(), *116*

vue_answer, 337
vue_app, 338
vue_app(), *327*, *328*, *332*, *335*, *337*, *341*, *343*, *344*
vue_component, 341

vue_component(), *339*, *340*
vue_data_output *(*render_vue_data*)*, 332
vue_output, 342
vue_store, 344
vue_store(), *332*, *334*, *340*, *343*