Package: mascot (via r-universe)

November 12, 2026

Title Live Mood-Driven Mascots for 'Shiny' Apps and 'Quarto' Sites

Version 0.1.0

Description Adds a character drawn in code to 'Shiny' apps and 'Quarto' sites. The character shows a mood with status text beside it and animates when the mood changes; a 'Shiny' server can change its mood, status, speech bubble and gestures, and a click can run an action or open a menu. It follows light and dark mode and takes its colours from built-in palettes or custom colours. Custom characters can be described in 'JSON' and checked before use, and an interactive app demonstrates every function.

License MIT + file LICENSE

URL https://insightsurge.github.io/mascot/, https://github.com/insightsurge/mascot

BugReports https://github.com/insightsurge/mascot/issues

Depends R (>= 4.1.0)

Imports checkmate, cli, htmltools (>= 0.5.3), jsonlite, shiny (>= 1.8.1)

Suggests bslib (>= 0.7.0), chromote, covr, golem, knitr, later, lifecycle, promises, rmarkdown, shinytest2 (>= 0.5.0), testthat (>= 3.2.1), withr

VignetteBuilder knitr

Config/cran-gate/coverage 70

Config/cran-gate/extra-dirs tools

Config/testthat/edition 3

Config/testthat/parallel true

Encoding UTF-8

Language en-GB

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Harsh Verma [aut, cre, cph]

Maintainer Harsh Verma <analyticswithharsh@gmail.com>

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

Date/Publication 2026-11-12 12:58:23 UTC

RemoteUrl https://github.com/cran/mascot

RemoteRef HEAD

RemoteSha c7771c6f2e06bafaa00802db0983151356079701

Contents

mascot . . . 3
mascot_action . . . 6
mascot_anchors . . . 8
mascot_bubble . . . 9
mascot_character . . . 10
mascot_character_format . . . 11
mascot_characters . . . 13
mascot_check_character . . . 14
mascot_colour_roles . . . 15
mascot_example . . . 16
mascot_examples . . . 17
mascot_families . . . 18
mascot_gesture . . . 18
mascot_gestures . . . 19
mascot_mood . . . 20
mascot_moods . . . 21
mascot_motion_settings . . . 22
mascot_palettes . . . 24
mascot_placement . . . 25
mascot_popup_menu . . . 27
mascot_preview . . . 29
mascot_read_character . . . 31
mascot_size . . . 32
mascot_status . . . 33
mascot_studio . . . 34
mascot_styling . . . 35
mascot_theme . . . 37
update_mascot . . . 40

Index **46**

`mascot` *Add a mascot to a page*

Description

Draw a character with a mood, a status line and an optional message bubble. Until the app sets its own status text, the status line shows the character's line for the mood (the `status` column of `mascot_moods(character)`). `mascot_status()` with no text hands the status back to the mood.

Usage

```
mascot(
  id,
  ...,
  character = "tessa",
  mood = "idle",
  status = NULL,
  bubble = NULL,
  size = 160,
  placement = "bottom-right",
  theme = NULL,
  motion = TRUE,
  click = NULL
)
```

Arguments

`id`	A required non-empty string without whitespace, unique on the page. In a Shiny module use `ns("helper")` in the UI and `"helper"` in the server; `update_mascot()` adds the module namespace once.
`...`	Must be empty. Forces the arguments after it to be named.
`character`	A character id such as `"tessa"`, or a `mascot_character()` helper (Knurl and Neer are experimental). Defaults to `"tessa"`. `NULL` is not accepted.
`mood`	A mood name or `mascot_mood()` object. Defaults to `"idle"`. See `mascot_moods()` for available choices. `NULL` is not accepted.
`status`	A non-empty string or `mascot_status()` object. `NULL` (the default) follows the mood. Text is always escaped, even inside `htmltools::HTML()`.
`bubble`	A non-empty string, an `htmltools` tag or tag list, or a `mascot_bubble()` object. `NULL` (the default) shows no bubble. A plain string gets the defaults of `mascot_bubble()`: five seconds and a close button. Plain text is escaped; `htmltools::HTML()` or tags supply HTML. Tags with dependencies or head content are not supported.
`size`	A finite number of CSS pixels (at least 24), a positive percentage string such as `"20%"`, or a `mascot_size()` object. Defaults to `160`. Percentages follow the parent width when inline and the window width when floating, with a 24-pixel

floor. On screens at most 575.98 CSS pixels wide, floating mascots scale down to a phone size by default (`"auto"`), while inline mascots stay unchanged. Pass `mascot_size()` with `small_screen = NULL` to keep the same size everywhere. `NULL` is not accepted.

`placement` `"bottom-right"` (the default), `"inline"`, a corner such as `"top-left"`, a named list such as `list(position = "top-left", minimise = TRUE, drag = "snap")`, or a `mascot_placement()` object. Floating mascots show a minimise control by default; dragging is off. Inline mascots sit in the page flow and ignore `minimise` and `drag`. If you set either one yourself, a message says so; the defaults give no message. With `drag`, visitors can also drag the minimised restore button; each drop sets `input$<id>_position` to the corner (`"snap"`) or a list with `x` and `y` (`"free"`, experimental); saving it between visits is the app's job. See `mascot_placement()`. `NULL` is not accepted.

`theme` A palette id string from `mascot_palettes()`, a named list of colour roles, a `bslib::bs_theme()` or a `mascot_theme()` object. `NULL` (the default) keeps the character's own colours. A palette sets the base colours for light and dark mode; `"original"` selects the character's own colours. Roles you leave unset follow the character's rules.

`motion` A logical flag or `mascot_motion_settings()` object. `TRUE` (the default) allows movement, unless the visitor prefers reduced motion. `FALSE` stops all movement, including panels, drag snapping, minimising and restoring. `NULL` is not accepted.

`click` What a click on the mascot does in a Shiny app. `NULL` (the default) does nothing: the mascot is a picture, and keyboard focus skips it. A label that is not blank, such as `"Open help"`, or a `mascot_action()` object makes it a button for one action of your app. A `mascot_popup_menu()` object makes it open a menu, whose choice sets `input$<id>_menu`. With a click, each time the visitor activates the action, `input$<id>_click` is set (see Details).

Details

In `mascot()` and its helpers, `NULL` (where accepted) means none or the default; in `update_mascot()` it means unchanged. An empty helper (`mascot_bubble()`, `mascot_status()`, `mascot_theme()`, `mascot_action()` or `mascot_popup_menu()`) resets that setting.

`mascot()` and the helpers raise an error on an invalid value; `update_mascot()` warns instead and leaves the mascot unchanged. See mascot-package for condition classes.

Use the always visible status line for a short description of the current state. Use a bubble for a message about something that happened; it may time out. Use `mascot_bubble(..., duration = Inf)` for a message the user must act on, such as an invalid form, because a timed message can vanish before it is read.

When minimised, the status remains available to screen readers. See `mascot_placement()` for restore controls and message timing.

In Shiny the mascot is also an input. `input$<id>` only tells the server that the mascot has appeared; its contents are internal, so do not rely on them.

In modules, `golem` and rhino apps. Use the module's `ns()` in the UI and its local id in the server. `session$ns("helper")` also works in the server; the namespace is not added twice. In nested modules, namespace the child module's id in the parent UI and call its server with the local

child id. Inputs such as `input$helper_menu`, `input$helper_click`, `input$helper_position` and `input$helper_minimised` belong to that child module. A module can put the mascot in its UI and update it from its server:

```
helper_ui <- function(id) {
  ns <- shiny::NS(id)
  mascot(ns("helper"))
}
helper_server <- function(id) {
  shiny::moduleServer(id, function(input, output, session) {
    update_mascot("helper", mood = "happy")
  })
}
```

The same pattern works in the function-built `app_ui()` of a `golem` app and in rhino's `app/view/` modules using `shiny::moduleServer()`. The package's scripts and styles are attached through `htmltools::htmlDependency()`, including with `shiny::renderUI()`, and shared by mascots on the page. A built-in character's data travels the same way, once per page for each character in use; a custom character's data stays inside its own tag. You do not need to copy any of the package's files into your app. Try `mascot_example("golem")` (requires `golem`) or `mascot_example("rhino")` (a plain Shiny structure without rhino or box).

Showing the mascot later. To add the mascot only when it is needed, build it in the server with `shiny::renderUI()` and place `shiny::uiOutput()` in the UI. Updates sent before it appears are kept until it does, so the first `update_mascot()` can run in the same observer:

```
library(shiny)
library(mascot)

ui <- fluidPage(
  actionButton("start", "Start the review"),
  actionButton("finish", "Finish"),
  uiOutput("helper_ui")
)

server <- function(input, output, session) {
  shown <- reactiveVal(FALSE)
  output$helper_ui <- renderUI({
    req(shown())
    mascot("helper", status = "Reviewing")
  })
  observeEvent(input$start, {
    shown(TRUE)
    # Sent before the mascot exists: kept until it appears.
    update_mascot("helper", mood = "thinking")
  })
  observeEvent(input$finish, {
    update_mascot("helper", mood = "happy", status = "Review complete")
  })
```

```
}

shinyApp(ui, server)
```

The `reactiveVal()` makes the mascot appear once: clicking **Start the review** again does not rebuild it, so it keeps its mood and position.

Clicks. With `click`, the mascot is a button: an action named by its label, or a menu. Each time the visitor activates the action (a click or tap, Enter or Space while it has keyboard focus, or a screen reader's activate command), `input$<id>_click` is set to the number of activations since the mascot appeared, so `shiny::observeEvent(input$helper_click, ...)` runs each time for an action. Opening and closing a menu never sets this input. Dragging a floating mascot never counts as a click. See `mascot_action()` and `mascot_popup_menu()`.

Value

An `htmltools` tag with the mascot's web dependency attached.

See Also

`update_mascot()`

Other mascots in Shiny: `update_mascot()`

Examples

```
ui <- mascot("helper", mood = "happy")
ui <- mascot("small", size = 96, placement = "bottom-right")
ui <- mascot("responsive", size = mascot_size(120, small_screen = 48))
ui <- mascot("still", motion = FALSE)
ui <- mascot("helpful", click = "Open help")

shiny::shinyApp(
  ui = shiny::fluidPage(mascot("helper")),
  server = function(input, output, session) {}
)
```

---

mascot_action *Make the mascot a button for one action*

Description

Turn the mascot into a button for one action of your Shiny app, such as opening a help panel or a chat. Pass it as `click` to `mascot()` or `update_mascot()`, and handle the click in your server with `shiny::observeEvent(input$<id>_click, ...)`. For a choice of several actions, use `mascot_popup_menu()` instead.

Usage

```
mascot_action(label = NULL, ..., tooltip = TRUE, disabled = FALSE)
```

Arguments

label	The action's name: one string, not blank, that says what a click does, such as "Open help". Screen readers announce the mascot as a button with this name, and the tooltip shows it. NULL (the default) makes an empty helper: in update_mascot(), it removes the click, so the mascot is a picture again. With label = NULL, setting tooltip or disabled yourself is an error; use mascot_action() without options to remove the click.
...	Must be empty. Forces the arguments after it to be named.
tooltip	A flag. TRUE (the default) shows the label in a tooltip when the pointer rests on the mascot or the keyboard focuses it. FALSE hides the tooltip; screen readers still announce the label.
disabled	A flag. TRUE keeps the mascot a button that does nothing: clicks are not sent, and screen readers announce the button as unavailable. Defaults to FALSE. Use it while the action makes no sense, such as during a long task, and send mascot_action(..., disabled = FALSE) afterwards.

Details

What a click sends. Every activation of the button sets input$<id>_click to the number of activations since the mascot appeared, so shiny::observeEvent() runs each time. An activation is a click or tap, Enter or Space while the mascot has keyboard focus, or a screen reader's activate command. Holding a key down counts once. Opening and closing a mascot_popup_menu() never sets this input. Dragging a floating mascot, clicking elsewhere, pressing Escape, the minimise and restore buttons and updates from the server never count. In a module, use ns("helper") in the UI and read input$helper_click in that module's server.

Without a click. The default, click = NULL in mascot(), keeps the mascot a picture: keyboard focus skips it and it sends no clicks. Use a click only when it does something, so keyboard users never land on a control that does nothing.

The tooltip shows the label next to the mascot, on the side with room. It stays while the pointer is over the mascot or the tooltip, and Escape hides it without moving the focus. It never shows on touch screens or while a menu or bubble is open. On static pages the mascot stays a picture.

Value

A mascot_action object. Its fields are internal; pass it on rather than reading them.

See Also

mascot(), update_mascot(), mascot_popup_menu()

Other mascot helpers: mascot_anchors(), mascot_bubble(), mascot_character(), mascot_characters(), mascot_check_character(), mascot_colour_roles(), mascot_families(), mascot_gesture(), mascot_gestures(), mascot_mood(), mascot_moods(), mascot_motion_settings(), mascot_palettes(), mascot_placement(), mascot_popup_menu(), mascot_preview(), mascot_read_character(), mascot_size(), mascot_status(), mascot_theme()

Examples

```
mascot_action("Open help")
mascot_action("Open help", tooltip = FALSE)
mascot_action()
ui <- mascot("helper", click = "Open help")

shiny::shinyApp(
  shiny::fluidPage(mascot("helper", click = "Open help")),
  function(input, output, session) {
    shiny::observeEvent(input$helper_click, {
      shiny::showModal(shiny::modalDialog("Help for this app"))
    })
  }
)
```

`mascot_anchors` *List a character's anchors*

Description

Anchors are named points on a character where effects such as sweat and sparkles, gesture effects and accessories attach. They are listed in the order the author wrote them. A character may have no anchors.

Usage

```
mascot_anchors(character = "tessa")
```

Arguments

`character`	A character, defaulting to `"tessa"`: an id string or a `mascot_character()` helper. `NULL` is not accepted.

Value

A data frame with character columns `name` (the anchor id), `description`, `part` (the body part the anchor belongs to) and `side` (`"l"` or `"r"`, or `NA` for an anchor on neither side). A character with no anchors returns a zero-row data frame with the same columns and column types. A base data frame. Columns may be added at the end in later versions; existing columns keep their names and types. The first column is the anchor name.

See Also

`mascot_characters()`

Other mascot helpers: `mascot_action()`, `mascot_bubble()`, `mascot_character()`, `mascot_characters()`, `mascot_check_character()`, `mascot_colour_roles()`, `mascot_families()`, `mascot_gesture()`, `mascot_gestures()`, `mascot_mood()`, `mascot_moods()`, `mascot_motion_settings()`, `mascot_palettes()`, `mascot_placement()`, `mascot_popup_menu()`, `mascot_preview()`, `mascot_read_character()`, `mascot_size()`, `mascot_status()`, `mascot_theme()`

Examples

```
mascot_anchors()
mascot_anchors("tessa")
```

`mascot_bubble` *Set a mascot's message bubble*

Description

A speech bubble stays visible across mood changes and character swaps. Plain text is escaped; use `htmltools::HTML()` only for HTML you trust. A timed countdown pauses while the mascot is hidden or minimised, while the pointer is over the bubble, or while keyboard focus is anywhere inside the mascot, and resumes with the time left. Pressing Escape while focused inside the mascot closes the bubble. Closing a bubble removes it until the app supplies a new one. While the bubble shows, the mascot looks towards it: up when the bubble is above (inline and bottom corners), down when it is below (top corners).

Usage

```
mascot_bubble(text = NULL, ..., duration = 5, closable = TRUE)
```

Arguments

`text`	A non-empty string, optionally marked with `htmltools::HTML()`, or an `htmltools` tag or tag list. HTML-marked strings and tags are shown as HTML. Tags with dependencies or head content are not supported and give an error. `NULL` (the default) clears the bubble. With `text = NULL`, setting `duration` or `closable` yourself is an error; use `mascot_bubble()` without options to clear the bubble.
`...`	Must be empty. Forces the arguments after it to be named.
`duration`	How long the bubble stays visible: a number greater than zero, in seconds. Defaults to `5`. Use `Inf` to keep the bubble until it is closed, replaced or cleared.
`closable`	A logical flag, defaulting to `TRUE`, to show a close button. `FALSE` hides the button; the Escape key still closes the bubble. `NULL` is not accepted.

Details

Use the always visible status line for a short description of the current state. Use a bubble for a message about something that happened; it may time out. Use `duration = Inf` for a message the user must act on, such as an invalid form, because a timed message can vanish before it is read.

Bubble or menu? A bubble is a message the app shows; the server supplies or updates it with `update_mascot(id, bubble = ...)`. For an action the visitor chooses, use `mascot_popup_menu()` with `click =` instead.

Value

A `mascot_bubble` object. Its fields are internal; pass it on rather than reading them.

See Also

mascot(), update_mascot(), mascot_popup_menu()

Other mascot helpers: mascot_action(), mascot_anchors(), mascot_character(), mascot_characters(), mascot_check_character(), mascot_colour_roles(), mascot_families(), mascot_gesture(), mascot_gestures(), mascot_mood(), mascot_moods(), mascot_motion_settings(), mascot_palettes(), mascot_placement(), mascot_popup_menu(), mascot_preview(), mascot_read_character(), mascot_size(), mascot_status(), mascot_theme()

Examples

```
mascot_bubble("3 rows skipped")
mascot_bubble("Ready", duration = Inf)
mascot_bubble(htmltools::HTML("<strong>Ready</strong>"))
mascot_bubble()
```

---

mascot_character *Choose a mascot character*

Description

The helper behind the character argument of mascot(). Tessa and Sela are stable built-in characters; Knurl and Neer are experimental.

Usage

```
mascot_character(id = "tessa")
```

Arguments

id	A non-empty string naming a known character, or checked mascot_character_data from mascot_read_character() or mascot_check_character(). Defaults to "tessa"; NULL is not accepted. A checked object is used as passed and is not stored anywhere; pass it each time you choose that character.

Details

Tessa and Sela are stable: their ids, names, five shared moods, gesture names, colour-role names and palette names stay the same in every release. Their drawings, motion and exact palette colours may be polished. Knurl and Neer are experimental: any part may change, and neither will be removed without a warning one release earlier. The character file format and character authoring functions are experimental too.

Value

A mascot_character helper carrying the character's identity and, for a custom character, its checked drawing data. The returned object's fields are internal; pass the object on rather than reading its fields.

Experimental characters

[Experimental] Knurl and Neer.

See Also

`mascot()`

Other mascot helpers: `mascot_action()`, `mascot_anchors()`, `mascot_bubble()`, `mascot_characters()`, `mascot_check_character()`, `mascot_colour_roles()`, `mascot_families()`, `mascot_gesture()`, `mascot_gestures()`, `mascot_mood()`, `mascot_moods()`, `mascot_motion_settings()`, `mascot_palettes()`, `mascot_placement()`, `mascot_popup_menu()`, `mascot_preview()`, `mascot_read_character()`, `mascot_size()`, `mascot_status()`, `mascot_theme()`

Examples

```
mascot_character()
mascot_character("tessa")
path <- system.file("examples", "characters", "study_bot.json", package = "mascot")
mascot_character(mascot_read_character(path))
```

---

mascot_character_format

Character format version 1

Description

A guide to the character file format read by `mascot_read_character()` and `mascot_check_character()`. Descriptions are JSON objects with `format_version = 1`, `id`, `name`, `description`, a `family`, colours, ground, parts and five shared moods (`idle`, `thinking`, `happy`, `worried`, `celebrating`). Optional anchors, palettes, face kits and gestures describe their own supported names. Palette ids cannot be `original` or the reserved style aliases `minimal`, `pastel`, `vivid`, `corporate`, `earthy` and `night`. Small sizes may hide details the character marks for small sizes. Text is displayed as text, never HTML.

Details

Start from `system.file("examples", "characters", "study_bot.json", package = "mascot")`. To change a character, edit the plain list or JSON source and check it again; a checked object cannot be edited. A source file names its `family` and gives that family's definition in `family_definition` (`id`, `name`, `description` and optional `defaults`, such as shared gestures and palettes); a custom character cannot take a definition from the installed package. A resolved file has its family defaults merged in and embeds its face kits, so it does not depend on the installed package. A source file that names a shipped face kit (`classic` or `screen`) draws with that kit as installed; a change to a shipped kit's drawing is listed in NEWS.

Drawing coordinates are design units in a 300 by 300 frame. The point where the character stands on the ground is `(0, 0)`: 150 units from the left edge and 270 from the top. `y` grows downwards, so a part above the ground has a negative `y`. Every part, movement and effect must stay inside the frame at every detail size.

Capabilities and `requires`

Every version-1 mascot can draw the eight version-1 capabilities: `parts`, `paints`, `clip`, `outline`, `glow`, `face_kit`, `tracks` and `lod`. A file does not need to ask for them, so it needs no `requires` field. `requires` lists only capabilities added after version 1. A file that lists the version-1 names still loads. A name this version of mascot does not know, or a reserved one, stops the check with an error that names it and says to update mascot. Unsupported and reserved capabilities used as keys (such as `connector` or `morph`) are refused too.

Keys for other tools

- A top-level `$schema` string is accepted and kept, so an editor can link the file to a schema. mascot never reads it.
- A key that starts with `x_`, in any object whose fields the format fixes (the character itself, a part, an item, a mood, a gesture, a face and so on), is an extension key. Its value can be any JSON value (from R, a list or an unnamed vector; a data frame or a named vector is refused). It is kept as written when the character is checked, resolved and saved, and it is never read; it counts towards the file's size, depth and node limits. Resolution keeps the character's own extension keys; those of a family's `defaults` object stay in the resolved family snapshot. In an object whose keys are ids or names (`colours`, `moods`, `gestures`, `palettes` and the other maps), a key that starts with `x_` is an ordinary id.
- A `note` string, accepted in every fixed-field object, is a comment.

Diagnostics

Checks stop at the first problem, giving a diagnostic code, the JSON path and a repair instruction, and the value where one is at fault. A value outside its range or choices is named with the range or the choices. A drawing budget gives the band, the count and the limit; a character too big for its frame names the part, face, effect or shadow that reaches furthest outside, the side and by how much. The limits on input bytes, depth and node count, and the drawing budgets, apply to every input form. The format version, the capability set and the version of the input and drawing limits can each change on their own. The installed files are in `system.file("format", package = "mascot")`: `character-v1.json` describes the allowed fields, types and values; `policy-v1.json` holds the input limits, the capability names and the drawing budget of each detail band.

Compatibility

The format is experimental and its future direction is open; these points hold either way. Additions (new optional keys, wider ranges, new categories) stay in format version 1, so existing files keep working. A file that uses an addition lists it in `requires`, and an older version of mascot then stops with an error naming it instead of drawing it wrongly. Only a change that would break existing files introduces format version 2; retiring an older format version is announced in NEWS at least one release ahead. The checker may only get stricter in a new format version, except for a documented security or resource-safety fix, which comes with a diagnostic and migration guidance. Files are never modified or upgraded silently. A resolved file is self-contained.

Experimental

[Experimental] What the format accepts may grow in a later release; version-1 files keep loading (see Compatibility).

See Also

mascot_check_character(), mascot_read_character()

Examples

```
# Start from the bundled source, give the copy its own id, then check it again.
path <- system.file("examples", "characters", "study_bot.json", package = "mascot")
source <- jsonlite::read_json(path)
source$id <- "my_bot"
source$name <- "My Bot"
source$colours$body <- "#2a9d8f"
bot <- mascot_check_character(source)
bot
ui <- mascot("helper", character = bot)
```

---

mascot_characters *List the characters you can use*

Description

Lists the characters shipped with the package. Pass an `id` to the `character` argument of `mascot()`, or to the other listings, such as `mascot_moods()`.

Usage

```
mascot_characters(..., family = NULL)
```

Arguments

`...`	Must be empty. Forces the arguments after it to be named.
`family`	`NULL` (the default) for every character, or a family `id` from `mascot_families()` to list only its members.

Details

Stable built-ins keep their id, name, five shared moods, gesture names, colour-role names and palette names in every release; their drawing, motion and exact palette colours may be polished. Experimental built-ins may change in any way, and none is removed without a warning one release earlier. The character file format and authoring functions are experimental too.

Value

A data frame with the character columns `id` (the value you pass to `character`), `name` (the display name), `description` and `family` (the family `id`); the logical columns `is_default` (`TRUE` only for the character `mascot()` uses when you name none) and `built_in` (`TRUE` for characters shipped with the package); `lifecycle` (`"stable"` or `"experimental"` for every character that ships with mascot); and `palettes` (comma-separated palette ids, for reading only; its exact shape is not promised, so use `mascot_palettes()` for ids you rely on). A base data frame. Columns may be added at the end in later versions; existing columns keep their names and types.

Experimental characters

[Experimental] Knurl and Neer.

See Also

`mascot_character()`, `mascot_families()`

Other mascot helpers: `mascot_action()`, `mascot_anchors()`, `mascot_bubble()`, `mascot_character()`, `mascot_check_character()`, `mascot_colour_roles()`, `mascot_families()`, `mascot_gesture()`, `mascot_gestures()`, `mascot_mood()`, `mascot_moods()`, `mascot_motion_settings()`, `mascot_palettes()`, `mascot_placement()`, `mascot_popup_menu()`, `mascot_preview()`, `mascot_read_character()`, `mascot_size()`, `mascot_status()`, `mascot_theme()`

Examples

```
mascot_characters()
mascot_characters(family = "hexonians")
```

---

mascot_check_character

Check a character description

Description

Checks a character description against format version 1 and the package's limits on input, drawing and capabilities. It never repairs or recolours anything. To change a character, edit the JSON or the plain list and check it again; a checked object cannot be opened or edited.

Usage

```
mascot_check_character(x)
```

Arguments

`x`	A JSON string, a parsed plain list, or checked `mascot_character_data`. A string always means JSON, never a path or id. `NULL` is not accepted.

Details

Content problems inherit `mascot_error_character` and `mascot_error`, with `code`, `path`, `value` and `fix`. Argument problems inherit `mascot_error_argument` and `mascot_error`, with `argument`, `code` and `value`. Every character needs `wave` and `nod` tracks that change something. `wave` must change a channel of a part other than `root`; `nod` may change any channel, including `root`. A checked object is not stored anywhere, so pass it each time you use it. The format version, the set of supported capabilities and the version of the input and drawing limits can each change on their own. Built-in character ids are reserved and refused here, so give your character its own id before checking it.

Value

A checked, resolved, self-contained `mascot_character_data` object, visibly. Its fields are internal implementation details.

Experimental

[Experimental] Its arguments, results and diagnostics may change in a later release.

See Also

mascot_character_format

Other mascot helpers: `mascot_action()`, `mascot_anchors()`, `mascot_bubble()`, `mascot_character()`, `mascot_characters()`, `mascot_colour_roles()`, `mascot_families()`, `mascot_gesture()`, `mascot_gestures()`, `mascot_mood()`, `mascot_moods()`, `mascot_motion_settings()`, `mascot_palettes()`, `mascot_placement()`, `mascot_popup_menu()`, `mascot_preview()`, `mascot_read_character()`, `mascot_size()`, `mascot_status()`, `mascot_theme()`

Examples

```
path <- system.file("examples", "characters", "study_bot.json", package = "mascot")
source <- jsonlite::read_json(path)
bot <- mascot_check_character(source)
```

---

mascot_colour_roles *List a character's colour roles*

Description

Lists a character's colour roles (the parts you can colour) with their own colours, or with a named palette applied. Roles are written in snake case in R. A role can follow other roles, its leaders, listed in priority order. A colour you supply for a role always wins; otherwise the role follows the first leader that has been given a colour. If no leader has one, the role keeps its original colour.

Usage

```
mascot_colour_roles(character = "tessa", ..., palette = NULL)
```

Arguments

`character`	A character, defaulting to `"tessa"`: an id string or a `mascot_character()` helper. `NULL` is not accepted.
`...`	Must be empty. Forces the arguments after it to be named.
`palette`	A palette id from `mascot_palettes()`, or `NULL` (the default) for the character's own colours. Roles that follow others take their colours from the palette's leaders.

Value

A data frame with character columns `role`, `colour`, `follows` (comma-separated leaders), `shade` (`same`, `lighter` or `darker`), `description` (what the role colours) and `brand` (the brand slots the role fills, `primary` and `secondary`, comma-separated). Roles without a follow rule have `NA` in `follows` and `shade`; roles that fill no brand slot have `NA` in `brand`. A base data frame. Columns may be added at the end in later versions; existing columns keep their names and types.

See Also

`mascot_theme()`, `mascot()`

Other mascot helpers: `mascot_action()`, `mascot_anchors()`, `mascot_bubble()`, `mascot_character()`, `mascot_characters()`, `mascot_check_character()`, `mascot_families()`, `mascot_gesture()`, `mascot_gestures()`, `mascot_mood()`, `mascot_moods()`, `mascot_motion_settings()`, `mascot_palettes()`, `mascot_placement()`, `mascot_popup_menu()`, `mascot_preview()`, `mascot_read_character()`, `mascot_size()`, `mascot_status()`, `mascot_theme()`

Examples

```
mascot_colour_roles()
mascot_colour_roles("tessa", palette = "coral_navy")
```

---

mascot_example *Run or locate a bundled example app*

Description

Opens only the example you ask for, like `shiny::runExample()`, and copies or writes no files. Use `run = FALSE` to read the app's code or pass its directory to `shiny::runApp()`. The single-file scenarios can be copied into a project with Shiny, `bslib` and mascot installed (`long-task` also needs `promises` and `later`). Copy the whole folder for the studio, `golem` and rhino-shaped examples; `golem` also needs the `golem` package. `"theme"` is a page of the studio's app: `run = FALSE` returns the studio's `app.R`, and `run = TRUE` opens the app on that page, as `mascot_studio()` with `view = "theme"` does.

Usage

```
mascot_example(name, ..., run = TRUE)
```

Arguments

`name`	A required non-empty string from `mascot_examples()`'s `name` column. `NULL` is not accepted.
`...`	Must be empty. Forces the arguments after it to be named.
`run`	A logical flag, defaulting to `TRUE`, to launch the app in an interactive R session. `FALSE` returns its `app.R` path without launching it. `NULL` is not accepted.

Value

With `run = FALSE`, an absolute path to the existing `app.R` file. With `run = TRUE`, the value returned by `shiny::runApp()` after the app stops.

See Also

`mascot_examples()`

Other mascot examples: `mascot_examples()`, `mascot_studio()`

Examples

```
basename(mascot_example("form-check", run = FALSE))

mascot_example("studio")
```

`mascot_examples` *List the bundled example apps*

Description

Start with the studio (experimental): every exported function and argument, live, with explanations and the code to copy. `"theme"` is a page of the same app that themes one character and gives the code to copy. The other examples show one scenario each. All work offline. Some need optional packages: the studio, the theme page and the single-file examples need `bslib`, and the studio also needs `knitr`; the `long-task` example needs `promises` and `later`, and the `golem` example needs `golem`. The rhino-shaped example uses plain Shiny without rhino or box.

Usage

```
mascot_examples()
```

Value

A data frame with character columns `name`, `title` and `description`, in teaching order. `name` is accepted by `mascot_example()`. A base data frame. Columns may be added at the end in later versions; existing columns keep their names and types.

See Also

`mascot_example()`

Other mascot examples: `mascot_example()`, `mascot_studio()`

Examples

```
mascot_examples()
```

mascot_families *List the character families*

Description

A family is a group of characters that share a body plan, such as hexagonal bodies. Use a family `id` to filter `mascot_characters()`.

Usage

```
mascot_families()
```

Value

A data frame with character columns `id` (the value you pass to the `family` argument of `mascot_characters()`), `name`, `description` and `character` (`NA` for every family that ships with mascot). The family of the default character comes first. A base data frame. Columns may be added at the end in later versions; existing columns keep their names and types.

See Also

`mascot_characters()`

Other mascot helpers: `mascot_action()`, `mascot_anchors()`, `mascot_bubble()`, `mascot_character()`, `mascot_characters()`, `mascot_check_character()`, `mascot_colour_roles()`, `mascot_gesture()`, `mascot_gestures()`, `mascot_mood()`, `mascot_moods()`, `mascot_motion_settings()`, `mascot_palettes()`, `mascot_placement()`, `mascot_popup_menu()`, `mascot_preview()`, `mascot_read_character()`, `mascot_size()`, `mascot_status()`, `mascot_theme()`

Examples

```
mascot_families()
```

mascot_gesture *Choose a one-off mascot gesture*

Description

Play a single movement without changing the current mood. Use this helper or a plain gesture name in `update_mascot()`.

Usage

```
mascot_gesture(name)
```

Arguments

name	A required non-empty gesture id: a lowercase letter followed by lowercase letters, digits or underscores. "nod" acknowledges an action; "wave" greets (characters without arms rock instead). See mascot_gestures() for descriptions. There is no default; NULL is not accepted.

Details

A character or mood change in the same call applies first, then the gesture. During a character swap the gesture is skipped: the wave within the swap replaces it. Reduced motion skips the movement. A gesture sent before the mascot appears is kept until it appears; only the latest early gesture plays, once, after the queued character and mood are applied.

This helper checks only the form of the id. update_mascot() checks whether the chosen character declares that gesture. If it does not, the warning lists that character's available gestures and leaves the mascot unchanged.

Value

A mascot_gesture object. Its fields are internal; pass it on rather than reading them.

See Also

update_mascot()

Other mascot helpers: mascot_action(), mascot_anchors(), mascot_bubble(), mascot_character(), mascot_characters(), mascot_check_character(), mascot_colour_roles(), mascot_families(), mascot_gestures(), mascot_mood(), mascot_moods(), mascot_motion_settings(), mascot_palettes(), mascot_placement(), mascot_popup_menu(), mascot_preview(), mascot_read_character(), mascot_size(), mascot_status(), mascot_theme()

Examples

```
mascot_gesture("nod")
mascot_gesture("wave")
```

---

mascot_gestures *List the mascot gestures*

Description

Gestures play a quick movement once, without changing the mascot's mood. They are available only in update_mascot(). With no character, it lists the gestures every character shares, with their nominal lengths. With a character, it lists that character's gestures: "nod" and "wave" first, then its own in the order its author wrote them.

Usage

```
mascot_gestures(character = NULL)
```

Arguments

character	NULL (the default) for the shared gestures, or a character: an id string such as "tessa" or a mascot_character() helper.

Value

A data frame with character columns name (the value you pass to mascot_gesture()), description and fallback (the gesture declared as the next choice when this one cannot play, or NA), and an integer column frames. Without a character, frames is the nominal length (18 for "nod", 30 for "wave") and fallback is NA. For a character, frames is the length of the movement that actually plays: a "wave" that falls back to a rock reports the length of the rock. Frames run at 30 per second. A base data frame. Columns may be added at the end in later versions; existing columns keep their names and types.

See Also

mascot_gesture()

Other mascot helpers: mascot_action(), mascot_anchors(), mascot_bubble(), mascot_character(), mascot_characters(), mascot_check_character(), mascot_colour_roles(), mascot_families(), mascot_gesture(), mascot_mood(), mascot_moods(), mascot_motion_settings(), mascot_palettes(), mascot_placement(), mascot_popup_menu(), mascot_preview(), mascot_read_character(), mascot_size(), mascot_status(), mascot_theme()

Examples

```
mascot_gestures()
mascot_gestures("tessa")
```

---

mascot_mood *Choose a mascot mood*

Description

Set how the mascot feels and looks. You can use this helper or a plain mood name in mascot() and update_mascot().

Usage

```
mascot_mood(name = "idle")
```

Arguments

name	A non-empty mood id: a lowercase letter followed by lowercase letters, digits or underscores. Defaults to "idle"; NULL is not accepted. See mascot_moods() for the available moods.

Details

This helper checks only the form of the id. `mascot()`, `update_mascot()` and `mascot_preview()` check whether the chosen character declares that mood. If it does not, their diagnostic lists that character's available moods.

Value

A `mascot_mood` object. Its fields are internal; pass it on rather than reading them.

See Also

`mascot()`

Other mascot helpers: `mascot_action()`, `mascot_anchors()`, `mascot_bubble()`, `mascot_character()`, `mascot_characters()`, `mascot_check_character()`, `mascot_colour_roles()`, `mascot_families()`, `mascot_gesture()`, `mascot_gestures()`, `mascot_moods()`, `mascot_motion_settings()`, `mascot_palettes()`, `mascot_placement()`, `mascot_popup_menu()`, `mascot_preview()`, `mascot_read_character()`, `mascot_size()`, `mascot_status()`, `mascot_theme()`

Examples

```
mascot_mood()
mascot_mood("happy")
```

---

mascot_moods *List mascot moods*

Description

Every character has the five shared moods, in this order, and may add more of its own. With no `character`, it lists the shared moods. With a character, it lists that character's moods: the shared ones first, then its own in the order its author wrote them.

Usage

```
mascot_moods(character = NULL)
```

Arguments

`character`	`NULL` (the default) for the shared moods, or a character: an id string such as `"tessa"` or a `mascot_character()` helper.

Value

A data frame with character columns `name` (the value you pass to `mascot_mood()`), `description`, `status` (the text shown beside the mascot for that mood), a logical column `shared` (`TRUE` for the five shared moods), and a character column `fallback` (the mood used instead when this one is not available, or `NA`). Shared moods have no `fallback`. A base data frame. Columns may be added at the end in later versions; existing columns keep their names and types.

See Also

mascot(), mascot_mood()

Other mascot helpers: mascot_action(), mascot_anchors(), mascot_bubble(), mascot_character(), mascot_characters(), mascot_check_character(), mascot_colour_roles(), mascot_families(), mascot_gesture(), mascot_gestures(), mascot_mood(), mascot_motion_settings(), mascot_palettes(), mascot_placement(), mascot_popup_menu(), mascot_preview(), mascot_read_character(), mascot_size(), mascot_status(), mascot_theme()

Examples

```
mascot_moods()
mascot_moods("tessa")
```

`mascot_motion_settings` *Choose which parts of the mascot move*

Description

This helper plays nothing; use mascot_gesture() to play a movement. After the mascot appears or changes (a new mood, a gesture or a swap), it breathes and blinks for up to five seconds, then rests. The visitor's reduced-motion setting always wins over any TRUE. Switches you do not set, including any added in later versions, follow others; switches you set always win.

Usage

```
mascot_motion_settings(
  ...,
  idle = NULL,
  mood_change = NULL,
  gestures = NULL,
  swap = NULL,
  minimising = NULL,
  restoring = NULL,
  messages = NULL,
  others = TRUE
)
```

Arguments

...	Must be empty; all settings must be named.
idle	Logical flag or NULL (default): breathing and blinking. NULL follows others.
mood_change	Logical flag or NULL (default): the wiggle or hop on a mood change. NULL follows others.

gestures	Logical flag or NULL (default): nods, waves and other gestures. NULL follows others. Skipping a gesture leaves the mood, bubble and status of the same update intact.
swap	Logical flag or NULL (default): character-change transitions. NULL follows others.
minimising	Logical flag or NULL (default): movement to the round restore button. NULL follows others.
restoring	Logical flag or NULL (default): movement back from the restore button. NULL follows others.
messages	TRUE for all panels, FALSE for none, or a character vector of names from c("bubble", "tooltip", "menu") for only those panels. NULL (default) follows others. Panels still appear and work.
others	Logical flag, default TRUE, for every switch you do not set. With others = FALSE and no switch you set allowing movement, this is the same as motion = FALSE, including the reduced-motion poses and instant drag moves.

Value

A mascot_motion_settings object. Its fields are internal; pass the object on rather than reading its fields.

See Also

mascot(), update_mascot(), mascot_gesture()

Other mascot helpers: mascot_action(), mascot_anchors(), mascot_bubble(), mascot_character(), mascot_characters(), mascot_check_character(), mascot_colour_roles(), mascot_families(), mascot_gesture(), mascot_gestures(), mascot_mood(), mascot_moods(), mascot_palettes(), mascot_placement(), mascot_popup_menu(), mascot_preview(), mascot_read_character(), mascot_size(), mascot_status(), mascot_theme()

Examples

```
mascot_motion_settings()
mascot_motion_settings(idle = FALSE)
mascot_motion_settings(messages = "menu", others = FALSE)
ui <- mascot("still", motion = FALSE)

shiny::shinyApp(
  shiny::fluidPage(
    shiny::checkboxInput("idle", "Allow idle movement", TRUE),
    shiny::actionButton("nod", "Nod"),
    mascot("helper")
  ),
  function(input, output, session) {
    shiny::observeEvent(input$idle, {
      update_mascot("helper", motion = mascot_motion_settings(idle = input$idle))
    })
    shiny::observeEvent(input$nod, {
      update_mascot("helper", gesture = "nod")
    })
```

```
  }
)
```

`mascot_palettes` *List a character's colour palettes*

Description

A palette is a ready-made set of colours for a character, in light and dark mode. The first row stands for the character's own colours with no palette applied. It is called `"original"`, unless the character names its own colours (Tessa's are `"night_teal"`); `"original"` then still selects the same colours.

Usage

```
mascot_palettes(character = NULL)
```

Arguments

`character`	An id string or `mascot_character()` helper. `NULL` (the default) lists every character, with a `character` column at the end.

Value

A data frame with character columns `name` (the palette id), `description` and `source`: `"native"` for the character's own colours, `"character"` for a palette the character defines and `"family"` for one it takes from its family. The character's own palettes follow in the order its author wrote them. With `character = NULL`, a final `character` column names each id. A base data frame. Columns may be added at the end in later versions; existing columns keep their names and types.

See Also

`mascot_colour_roles()`, `mascot_theme()`

Other mascot helpers: `mascot_action()`, `mascot_anchors()`, `mascot_bubble()`, `mascot_character()`, `mascot_characters()`, `mascot_check_character()`, `mascot_colour_roles()`, `mascot_families()`, `mascot_gesture()`, `mascot_gestures()`, `mascot_mood()`, `mascot_moods()`, `mascot_motion_settings()`, `mascot_placement()`, `mascot_popup_menu()`, `mascot_preview()`, `mascot_read_character()`, `mascot_size()`, `mascot_status()`, `mascot_theme()`

Examples

```
mascot_palettes()
mascot_palettes("tessa")
ui <- mascot("helper", character = "tessa", theme = "coral_navy")
```

`mascot_placement` *Place a mascot in a floating corner or inline*

Description

Floating mascots sit in a window corner (by default, `"bottom-right"`), 1rem from the window edges, above fixed navigation bars and below modals. Bottom-right mascots move above Shiny notifications while they are visible. A floating mascot hides when the tab or container it was placed in hides. Inline mascots sit where `mascot()` is placed in the page flow. Switching back to inline returns the mascot to its original place in the UI.

Usage

```
mascot_placement(
  position = "bottom-right",
  ...,
  minimise = TRUE,
  drag = "none"
)
```

Arguments

`position`	A corner to float the mascot (`"bottom-right"`, the default; `"top-left"`, `"top-right"` or `"bottom-left"`), or `"inline"` to sit where the mascot is placed in the UI. `NULL` is not accepted.
`...`	Must be empty. Forces the arguments after it to be named.
`minimise`	A logical flag, defaulting to `TRUE`, to show a minimise control on a floating mascot. Has no effect inline; if you set it yourself there, a message says so, and the default gives none. `NULL` is not accepted.
`drag`	`"none"` (the default) for no dragging, `"snap"` to let visitors drag the mascot and move it to the nearest corner on release, or `"free"` to keep it where it is dropped, including while minimised. **[Experimental]** `"free"` is experimental: the position it reports and how to restore it may change. Has no effect inline; if you set it yourself there, a message says so. `TRUE`, `FALSE` and `NULL` are not accepted. See the Dragging section.

Details

Floating mascots show a minimise control by default; set `minimise = FALSE` to hide it. The neutral control appears on hover or keyboard focus. Touch screens keep it visible; an open mascot menu hides it. Its border has a quiet glow. A minimised mascot is a round restore button in the character's colours. Its status remains available to screen readers. Updates still apply, and bubble countdowns pause until it is restored. Click, Enter or Space restores the mascot. On hover or keyboard focus the restore button glows and grows slightly; with reduced motion it only glows.

Value

A `mascot_placement` object. Its fields are internal; pass it on rather than reading them.

Dragging

With `drag = "snap"` or `drag = "free"`, visitors can drag a floating mascot by its drawing, with a mouse, pen or finger. A click without movement still opens the mascot's menu. While minimised, drag the round restore button: a click without movement restores the mascot, but a drag never restores it. After a snap, restoring shows the mascot in the new corner. The minimise button, the bubble's close button and links in the bubble do not start a drag. The bubble and menu move with the mascot.

- `"snap"`: on release the mascot moves to the nearest corner, which becomes its `position`, so its bubble opens towards the page centre. Reduced motion, or `motion = FALSE`, makes the move instant.
- `"free"`: the mascot stays where it is dropped, at least 1rem inside the window, and stays inside when the window is resized.

In Shiny each drop sets `input$<id>_position`, which you read in the module that owns the mascot, as with `input$<id>_menu`. After a snap it is the corner name, such as `"top-left"`; after a free drop it is a list with `x` and `y`, the mascot's top-left corner in CSS pixels from the window's top-left. A later `update_mascot()` with `placement` replaces the dragged position. Dragging has no keyboard alternative; the minimise control stays reachable by keyboard. Remembering a position between visits is the app's job: for example, save the corner and pass it back as `position` next time. In Shiny, minimising or restoring sets `input$<id>_minimised` to a logical flag in the module that owns the mascot.

Experimental free dragging

[Experimental] "free" is experimental: the position it reports and how to restore it may change.

See Also

`mascot()`, `update_mascot()`

Other mascot helpers: `mascot_action()`, `mascot_anchors()`, `mascot_bubble()`, `mascot_character()`, `mascot_characters()`, `mascot_check_character()`, `mascot_colour_roles()`, `mascot_families()`, `mascot_gesture()`, `mascot_gestures()`, `mascot_mood()`, `mascot_moods()`, `mascot_motion_settings()`, `mascot_palettes()`, `mascot_popup_menu()`, `mascot_preview()`, `mascot_read_character()`, `mascot_size()`, `mascot_status()`, `mascot_theme()`

Examples

```
mascot_placement()
mascot_placement("top-left")
mascot_placement("bottom-right", minimise = TRUE)
mascot_placement("bottom-right", drag = "snap")
ui <- mascot("helper", placement = list(position = "top-left", drag = "free"))

shiny::shinyApp(
  shiny::fluidPage(
```

```
    mascot("helper", placement = mascot_placement("bottom-right", drag = "snap")),
    shiny::textOutput("where")
  ),
  function(input, output, session) {
    output$where <- shiny::renderText(paste("Dropped in", input$helper_position))
  }
)
```

`mascot_popup_menu` *Choose a mascot's click menu*

Description

Make a click on the mascot open a menu in a Shiny app. Pass it as `click` to `mascot()` or `update_mascot()`. Clicking or tapping the mascot opens the menu, and clicking outside or pressing Escape closes it. While the menu is open, the mascot looks towards it, on the side where it actually opened (see "Where the menu opens" below). You provide the menu choices and handle the selected item in your Shiny server. For a single action, use `mascot_action()` instead.

Usage

```
mascot_popup_menu(
  items = NULL,
  ...,
  disabled = FALSE,
  direction = "auto",
  label = NULL,
  tooltip = TRUE
)
```

Arguments

`items`	A named character vector of plain-text labels. Names are unique, non-empty item IDs without whitespace (for example `c(about = "About", help = "Help")`). Defaults to `NULL`, which gives an empty helper; in `update_mascot()` it removes the click, so the mascot is a picture again. An empty vector is not accepted. Labels are always escaped. With `items = NULL`, setting `disabled`, `direction`, `label` or `tooltip` yourself is an error; use `mascot_popup_menu()` without options to remove the click.
`...`	Must be empty. Forces the arguments after it to be named.
`disabled`	A flag. `TRUE` keeps the menu button reachable by keyboard, but opens nothing and sends no input. Screen readers announce it as unavailable. `FALSE` (the default) allows the menu to open.
`direction`	Which side of the mascot the menu opens on: `"auto"` (the default), `"top"`, `"bottom"`, `"left"` or `"right"`. `"auto"` picks the side with room (preferring above, then below, then the side towards the page centre). A chosen side is a preference: see "Where the menu opens" below.

`label` The name screen readers announce for the mascot's menu button, such as `"Help and settings"`: one string that is not blank. `NULL` (the default) uses the character's name, such as "Tessa, menu".

`tooltip` A flag. `TRUE` (the default) shows the label in a tooltip when the pointer rests on the mascot or the keyboard focuses it, provided a `label` is set. `FALSE` hides the tooltip; screen readers still announce the label.

Details

Bubble or menu? A menu lets the visitor ask the app to do something: attach it with `click =` and read the choice from `input$<id>_menu`. For a message the app sends to the visitor, use `mascot_bubble()` and `update_mascot(id, bubble = ...)`.

Where the menu opens. The menu must fit in the space where it can show. For an inline mascot, that space is bounded by the nearest container that scrolls or clips (such as a `bslib::card()` body or a sidebar), not the whole window. For a floating mascot, it uses the window. When the chosen `direction` has no room there, the menu opens on the opposite side; when neither side has room, it opens wherever `"auto"` finds space. So `direction = "bottom"` on a mascot near the bottom of a card opens above it. Give the mascot room on the chosen side, or use `"auto"`.

Keyboard navigation. Enter or Space on the focused mascot opens or closes the menu and moves the focus to its first item; holding the key down toggles it once. After opening it with a click, the down and up arrows move from the mascot to the first or last item. Use the arrow keys, Home and End to move between items, and press Escape to close the menu and return focus to the mascot.

Reading the selection in Shiny. Choosing an item sets `input$<id>_menu` to the item ID (for example, `input$helper_menu` for `mascot("helper")`). Choosing the same item twice triggers the input twice. Opening and closing the menu never set `input$<id>_click`; that input is only for `mascot_action()`. In a module, use `ns("helper")` in the UI and read `input$helper_menu` inside that module's server:

```
helper_ui <- function(id) {
  ns <- shiny::NS(id)
  mascot(ns("helper"), click = mascot_popup_menu(
    c(about = "About this app")))
}
helper_server <- function(id) {
  shiny::moduleServer(id, function(input, output, session) {
    shiny::observeEvent(input$helper_menu, {
      if (input$helper_menu == "about") {
        shiny::showModal(shiny::modalDialog("About this app"))
      }
    })
  })
}
shiny::shinyApp(
  shiny::fluidPage(helper_ui("demo")),
  function(input, output, session) helper_server("demo")
)
```

The menu opens next to the mascot, on the side you asked for. It is sized to fit the space on screen, and can change side or scroll when room is short. Action controls hide while the menu is open. Opening the menu closes any speech bubble that is showing. The restore button only restores a minimised mascot; it does not open the menu. On static pages the mascot stays an image. Run the Shiny example below to try the menu.

Value

A `mascot_popup_menu` object. Its fields are internal; pass it on rather than reading them.

See Also

`mascot()`, `update_mascot()`, `mascot_action()`, `mascot_bubble()`

Other mascot helpers: `mascot_action()`, `mascot_anchors()`, `mascot_bubble()`, `mascot_character()`, `mascot_characters()`, `mascot_check_character()`, `mascot_colour_roles()`, `mascot_families()`, `mascot_gesture()`, `mascot_gestures()`, `mascot_mood()`, `mascot_moods()`, `mascot_motion_settings()`, `mascot_palettes()`, `mascot_placement()`, `mascot_preview()`, `mascot_read_character()`, `mascot_size()`, `mascot_status()`, `mascot_theme()`

Examples

```
mascot_popup_menu(c(about = "About this app", help = "Help"))
mascot_popup_menu(c(about = "About this app"), direction = "top")
mascot_popup_menu(c(about = "About this app"), label = "Help and settings")
mascot_popup_menu(c(about = "About this app"), label = "Help and settings", tooltip = FALSE)
mascot_popup_menu()
ui <- mascot("helper", click = mascot_popup_menu(c(about = "About this app")))

shiny::shinyApp(
  shiny::fluidPage(mascot("helper", click = mascot_popup_menu(
    c(about = "About this app")))),
  function(input, output, session) {
    shiny::observeEvent(input$helper_menu, {
      shiny::showModal(shiny::modalDialog(paste("Selected", input$helper_menu)))
    })
  }
)
```

---

mascot_preview *Preview a mascot in light and dark mode*

Description

Compare moods and sizes on a single offline page. Creating the page starts no Shiny app, opens no viewer and writes no files. Printing the returned page opens the available Viewer or browser. It can also be embedded in an R Markdown or Quarto document.

Usage

```
mascot_preview(
  character = "tessa",
  ...,
  mood = NULL,
  status = NULL,
  bubble = NULL,
  size = 160,
  theme = NULL,
  motion = FALSE
)
```

Arguments

character	A character id such as "tessa", or a mascot_character() helper (Knurl and Neer are experimental). Defaults to "tessa". NULL is not accepted.
...	Must be empty. Forces the arguments after it to be named.
mood	A mood name or mascot_mood() object, or NULL (the default) to show every mood from mascot_moods().
status	A non-empty string or mascot_status() object. NULL (the default) follows the mood. Text is always escaped, even inside htmltools::HTML().
bubble	A non-empty string, an htmltools tag or tag list, or a mascot_bubble() object. NULL (the default) shows no bubble. A plain string gets the defaults of mascot_bubble(): five seconds and a close button. Plain text is escaped; htmltools::HTML() or tags supply HTML. Tags with dependencies or head content are not supported.
size	A finite number of CSS pixels (at least 24), a positive percentage string such as "20%", or a mascot_size() object. Defaults to 160. Percentages follow the comparison cell width. NULL is not accepted.
theme	A palette id string from mascot_palettes(), a named list of colour roles, a bslib::bs_theme() or a mascot_theme() object. NULL (the default) keeps the character's own colours. A palette sets the base colours for light and dark mode; "original" selects the character's own colours. Roles you leave unset follow the character's rules.
motion	A logical flag or mascot_motion_settings() object. Defaults to FALSE for steady faces; TRUE allows movement unless the visitor prefers reduced motion. NULL is not accepted.

Details

Comparison grid. Rows are moods; columns are the requested size, then 160, 96 and 24 CSS pixels. The first column keeps your percentage and small_screen settings; the fixed columns do not change on narrow screens. Every size draws the full character. Each cell keeps its status text, since small faces lose fine detail.

Theme and contrast. A palette id string works as in mascot(). Built-in palette colours do not add contrast warnings. When the theme gives both a light and a dark map, both are shown, even if it sets

one mode. Backgrounds are white and #212529, or the bslib theme's bg and body-bg-dark; text uses fg and body-color-dark. Only supplied colour roles are checked, using the same contrast calculation as mascot(). The browser reads colours from experimental CSS variables and checks them. Any warnings appear in the browser's developer console, not in the R contrast attribute.

Bubbles. Cells without a bubble leave out its space; a supplied bubble keeps its normal space and timing. Each cell is wide enough to hold it, so bubbles never cover neighbouring cells.

Value

A browsable htmltools tag list. Its contrast attribute is a data frame with role, colour, background, ratio and mode columns, one row per supplied colour below 3:1. With no findings it is a zero-row data frame with the same columns. Colours are not changed. R gives one combined warning unless mascot.contrast_warnings is FALSE.

Experimental

[Experimental] Its purpose, arguments and page layout may change in a later release.

See Also

mascot(), mascot_colour_roles(), mascot_theme()

Other mascot helpers: mascot_action(), mascot_anchors(), mascot_bubble(), mascot_character(), mascot_characters(), mascot_check_character(), mascot_colour_roles(), mascot_families(), mascot_gesture(), mascot_gestures(), mascot_mood(), mascot_moods(), mascot_motion_settings(), mascot_palettes(), mascot_placement(), mascot_popup_menu(), mascot_read_character(), mascot_size(), mascot_status(), mascot_theme()

Examples

```
page <- mascot_preview(mood = "happy", size = 96)
attr(page, "contrast")

mascot_preview()
```

---

mascot_read_character *Read a local character file*

Description

Reads one local JSON file, the one you name, within the format's input limits. URLs and wildcards are not expanded. The checked result includes its family and face kits; keep the source file too if you want to edit the character later.

Usage

```
mascot_read_character(path)
```

Arguments

path	A non-empty string naming one local JSON file. NULL is not accepted.

Details

File problems inherit mascot_error_file and mascot_error, with the exact supplied path, operation = "read" and a stable code. Content and argument diagnostics are described in mascot_check_character().

Value

A checked mascot_character_data object, visibly.

Experimental

[Experimental] Its arguments, results and diagnostics may change in a later release.

See Also

mascot_character_format

Other mascot helpers: mascot_action(), mascot_anchors(), mascot_bubble(), mascot_character(), mascot_characters(), mascot_check_character(), mascot_colour_roles(), mascot_families(), mascot_gesture(), mascot_gestures(), mascot_mood(), mascot_moods(), mascot_motion_settings(), mascot_palettes(), mascot_placement(), mascot_popup_menu(), mascot_preview(), mascot_size(), mascot_status(), mascot_theme()

Examples

```
path <- system.file("examples", "characters", "study_bot.json", package = "mascot")
bot <- mascot_read_character(path)
```

---

mascot_size *Set the mascot's size*

Description

Control the size of the mascot in pixels or as a percentage. Percentage sizes follow the parent's content width when inline, and the window width when floating. The rendered canvas is never smaller than 24 CSS pixels. Screens at most 575.98 CSS pixels wide use small_screen (floating mascots scale down to a phone size by default). The full character is drawn at every size; fine face details are harder to read at 32 and 24 pixels, so the status text carries the meaning.

Usage

```
mascot_size(size = 160, ..., small_screen = "auto")
```

Arguments

size	A finite number of CSS pixels (at least 24), or a positive percentage string such as "20%". Defaults to 160; NULL is not accepted.
...	Must be empty. Forces the arguments after it to be named.
small_screen	An optional size for small screens (at most 575.98 CSS pixels wide). "auto" (the default) scales floating mascots down on phones to at most 96 pixels while leaving inline mascots unchanged. NULL keeps the same size on every screen. A number of pixels or a percentage string sets the phone size explicitly for both floating and inline mascots.

Value

A mascot_size object. Its fields are internal; pass it on rather than reading them.

See Also

mascot(), update_mascot()

Other mascot helpers: mascot_action(), mascot_anchors(), mascot_bubble(), mascot_character(), mascot_characters(), mascot_check_character(), mascot_colour_roles(), mascot_families(), mascot_gesture(), mascot_gestures(), mascot_mood(), mascot_moods(), mascot_motion_settings(), mascot_palettes(), mascot_placement(), mascot_popup_menu(), mascot_preview(), mascot_read_character(), mascot_status(), mascot_theme()

Examples

```
mascot_size()
mascot_size("20%")
mascot_size(120, small_screen = 48)
mascot_size(120, small_screen = NULL)
```

---

mascot_status *Set a mascot's status text*

Description

Sets the text label shown next to the mascot. Text is always shown as plain text, even when supplied with htmltools::HTML(). With no text, the status follows the shown mood.

Usage

```
mascot_status(text = NULL)
```

Arguments

text	A non-empty string for the label (for example "Saving..."), or NULL (the default) to follow the mood.

Value

A `mascot_status` object. Its fields are internal; pass it on rather than reading them.

See Also

`mascot()`, `update_mascot()`

Other mascot helpers: `mascot_action()`, `mascot_anchors()`, `mascot_bubble()`, `mascot_character()`, `mascot_characters()`, `mascot_check_character()`, `mascot_colour_roles()`, `mascot_families()`, `mascot_gesture()`, `mascot_gestures()`, `mascot_mood()`, `mascot_moods()`, `mascot_motion_settings()`, `mascot_palettes()`, `mascot_placement()`, `mascot_popup_menu()`, `mascot_preview()`, `mascot_read_character()`, `mascot_size()`, `mascot_theme()`

Examples

```
mascot_status("Saving")
mascot_status()
```

---

mascot_studio *Launch the interactive mascot studio*

Description

Explore the package's functions, try their controls with live feedback, and copy the working R code into your own app. The studio needs the suggested packages `bslib` (0.7.0 or later) and `knitr`; the `"theme"` view needs only `bslib`. The studio is experimental and may change in a later release. What you choose in it stays in the returned app; it changes no global options.

Usage

```
mascot_studio(view = "studio", ..., fn = NULL, character = "tessa", run = TRUE)
```

Arguments

`view`	Which view to open: `"studio"` (the default) for the studio, one page per function, `"examples"` for the gallery of bundled examples, or `"theme"` to theme one character.
`...`	Must be empty. Forces the arguments after it to be named.
`fn`	A function name in the studio's Guide, without parentheses. `NULL` starts on the canvas with `mascot_mood()` selected; a name opens that function's Guide. Used in the studio view only.
`character`	A built-in character id string from the studio's character selector, or a `mascot_character()` helper for a built-in character. Defaults to the default character of `mascot()`. A checked character object or a helper carrying one is refused with advice to use `mascot_preview()` or `mascot()` to display it. Checked objects come from `mascot_read_character()` or `mascot_check_character()`.
`run`	A logical flag. `TRUE` launches in an interactive R session; `FALSE` returns the app without launching it.

Value

With `run = FALSE`, a `shiny.appobj` object. With `run = TRUE`, the value returned by `shiny::runApp()` after the app stops.

Experimental interface

[Experimental]

See Also

`mascot_example()`

Other mascot examples: `mascot_example()`, `mascot_examples()`

Examples

```
app <- mascot_studio(fn = "mascot_theme", run = FALSE)
inherits(app, "shiny.appobj")
```

---

mascot_styling *Styling*

Description

Style a mascot's panels and restore button with CSS. Use `mascot_theme()` for the character's drawn colours.

Public classes

These classes are styling hooks: `.mascot` (the whole mascot), `.mascot-status` (status text), `.mascot-bubble` (speech panel), `.mascot-bubble-region` (the space for the speech panel), `.mascot-menu` (menu panel) and `.mascot-tooltip` (tooltip panel). Other classes and all `--_mascot-*` properties are private and may change. Use `mascot_size()` and `mascot_placement()` for size and position.

`.mascot-bubble-region` is the space kept for the speech bubble: above an inline mascot (so a bubble never moves the page) and next to a floating one. It is kept even while no bubble shows. On a page that never shows a bubble, hide it with `.mascot-bubble-region { display: none; }`, so an inline mascot keeps no empty space above it. Hidden, it also stops screen readers from announcing bubbles. Leave its position, size and pointer events to the package.

Public custom properties

Set these on `.mascot`, an individual mascot's id, or an ancestor such as `:root`. A colour you set wins over the page colours mascot copies and over Bootstrap variables, also in a `bslib` app. The package sets none of these colour properties itself. When you set none, panels follow the page and the restore button follows the character. When a mascot floats, it copies the values it inherited from its original container; values set on the mascot itself win.

- `--mascot-panel-bg`: bubble, menu and tooltip backgrounds and tails. Default: page background; otherwise `#fff` in light mode, `#212529` in dark.
- `--mascot-panel-color`: panel text and minimise control. Default: page text; otherwise `#212529` in light mode, `#f8f9fa` in dark.
- `--mascot-panel-border`: panel and tail borders, bubble close border. Default: page border; otherwise `#dee2e6` in light mode, `#495057` in dark.
- `--mascot-restore-bg`: round restore button background. Default: character body colour, or its first colour if absent.
- `--mascot-restore-fg`: restore button text, border and focus outline. Default: character outline colour, or its first colour if absent.
- `--mascot-restore-glow`: restore button glow on hover or keyboard focus. Default: the character's accent colour (its secondary brand colour, or `rim`), then its main colour, then its first colour.
- `--mascot-z-index`: floating mascot's stacking order. Default: `1031`, above Bootstrap fixed navigation bars and below overlay panels and modals. A supplied value wins over this default.

CSS example

Add this CSS to your app or site's CSS file. The panel colour properties style all three panel kinds; the restore properties style the restore button.

```
.mascot {
  --mascot-panel-bg: #153448;
  --mascot-panel-color: #ffffff;
  --mascot-panel-border: #8fc7e8;
  --mascot-restore-bg: #153448;
  --mascot-restore-fg: #ffffff;
  --mascot-restore-glow: #8fc7e8;
  --mascot-z-index: 1040;
}
```

Choose readable colours for your page's light and dark modes. CSS hooks do not change the character's drawn colours or check colour contrast.

See Also

`mascot_theme()`, `mascot_size()`, `mascot_placement()`

Examples

```
# In an app, the same CSS usually lives in a file under www/.
css <- "
.mascot {
  --mascot-panel-bg: #153448;
  --mascot-panel-color: #ffffff;
  --mascot-panel-border: #8fc7e8;
  --mascot-restore-bg: #153448;
  --mascot-restore-fg: #ffffff;
```

```
}"
ui <- htmltools::tagList(
  htmltools::tags$style(css),
  mascot("helper", bubble = "Hello!")
)
```

`mascot_theme` *Set a mascot's colours and light or dark mode*

Description

Customise the character's colours and light or dark mode. Each character names some roles as leaders (for example `body`), in order of priority. A role you do not set follows the first of those leaders that has been given a colour; if none has, the role keeps its original colour. Call `mascot_colour_roles()` to find the roles for a character. A plain named list passed to `mascot()` or `update_mascot()` uses one set of colours in every mode. A `bslib::bs_theme()` fills the character's declared brand leaders with its primary and secondary colours; `_brand.yml` can be supplied through `bs_theme(brand = )`. A `bs_theme()` always uses `mode = "auto"`. Which parts of a character a `bs_theme()` colours is experimental and may change in a later release.

Usage

```
mascot_theme(
  ...,
  palette = NULL,
  light = NULL,
  dark = NULL,
  mode = "auto",
  css = NULL
)
```

Arguments

`...`	Must be empty. Forces the arguments after it to be named.
`palette`	A palette id from `mascot_palettes()`, or `NULL` (the default) for the character's own colours. Checked against the chosen character when used. An unknown palette id is an error in `mascot()` (the UI is being built) but only a warning in `update_mascot()` (a live session, so the mascot keeps running unchanged). `"original"` also chooses the character's own colours. A palette is applied first, then `light` and `dark` roles, then `css`. Without a `dark` map, a palette uses its `light` colours in both modes. A `bs_theme()` cannot be combined with a palette.
`light, dark`	Named lists of colour roles, or `NULL` (the default) to keep the palette's colours (or the character's own colours) in that mode. Values can be colour names, hex strings, `grDevices::rgb()` output, CSS `rgb()` or `rgba()` strings, or three RGB numbers (red, green and blue) in 0–255. Single numeric palette indices are not accepted. Roles use snake case; `light`, `dark` and `default` are reserved. Names follow the character format: a lowercase letter, then lowercase letters

or digits separated by single underscores, with an optional final underscore. `mascot_theme()` checks the form of the names; `mascot()`, `update_mascot()` and `mascot_preview()` check that the chosen character has those roles. Built-in characters keep one outline colour in both modes; their light outer band keeps them readable on any dark page. To use your own dark-mode outline, set `dark = list(outline = "#FFC93C")`.

`mode` Light or dark mode: `"auto"` (the default), `"light"` or `"dark"`. `"auto"` follows the nearest valid `data-bs-theme` attribute around the place where the mascot was written in the page, then Quarto's body classes. With no page signal it uses light mode. An explicit light or dark mode always wins. `NULL` is not accepted.

`css` A named list mapping colour roles to CSS variable names such as `list(body = "--team-blue")`, or `NULL` (the default) for no CSS source. This source is experimental. CSS entries override light and dark entries for the same role. Missing or invalid variables use the character's own colour and warn in the browser console; low contrast also warns there and keeps the colour. Variables are read where the mascot was written in the page, also when the mascot floats.

Details

After a character swap, roles you did not set follow their leaders again, and roles the new character does not have are ignored. A palette is kept if the new character has a palette with the same id. If not, R warns, the new character gets its own colours, and role colours you set explicitly stay. An empty `mascot_theme()` restores the character's own colours. A colour you choose with less than 3:1 contrast gives a warning and is kept (see "Contrast warnings" below). Colours you supply are used exactly as given. Shades worked out from them keep the hue and shift lightness and saturation by tuned amounts. Brand colours are the same in both modes.

Value

A `mascot_theme` object describing colours, CSS sources and mode. The returned object's fields are internal; pass the object on rather than reading its fields.

Contrast warnings

A warning appears when a colour chosen by the developer (light, dark, or a `bslib::bs_theme()` brand colour) has contrast below 3:1 against the page background (white in light mode, `#212529` in dark mode, or the `bslib` theme's backgrounds). Contrast is checked when `mascot()` builds the UI and when `update_mascot()` applies colours with `theme = ` ; `mascot_preview()` collects the results in its contrast table. A character's own colours and the built-in palette colours never warn, and the colour is always kept.

R warnings appear only when `shiny::in_devmode()` is `TRUE`, and always in `mascot_preview()` and the studio. Set `options(mascot.contrast_warnings = TRUE)` or `FALSE` to force them on or off everywhere. A non-flag value warns once per R session and uses the default. Visitors never see R warnings. Colours from CSS variables (`css =`) are checked in the browser and warn in its console.

These warnings have the class `mascot_contrast`. To silence only these warnings:

```
withCallingHandlers(
```

```
  mascot("helper", theme = list(rim = "#f5e6c8")),
  mascot_contrast = function(w) invokeRestart("muffleWarning")
)
```

`suppressWarnings()` also works, but it hides every warning.

A `bs_theme()` or explicit colours

A `bslib::bs_theme()` cannot be combined with `palette`, `light`, `dark` or `css`: `theme` takes either the theme or a `mascot_theme()`, and a `bs_theme()` passed to `mascot_theme()` is an error. It fills only the character's brand roles (the `brand` column of `mascot_colour_roles()`: `body` for `primary` and `rim` for `secondary` on `"tessa"`), and the roles that follow them change with them. A character with no brand roles, such as `"neer"`, keeps its own colours. The colour is the same in light and dark mode. A brand colour that is dark enough on a white page can fail 3:1 on the dark background. In development mode (or with `options(mascot.contrast_warnings = TRUE)`; see "Contrast warnings") a warning of class `mascot_contrast` then names the slot and the mode, and the colour is kept. To change it, give those roles for each mode yourself:

```
mascot("helper", theme = mascot_theme(
  light = list(body = "#1b3a6b", rim = "#5c6770"),
  dark = list(body = "#9ec5fe", rim = "#adb5bd")
))
```

The example uses the brand roles of `"tessa"`; other characters name theirs in `mascot_colour_roles()`. Roles you leave out follow the ones you set, as described above.

Experimental CSS source

[Experimental] CSS variables are worked out by the browser, so R cannot check their values.

Experimental colour mapping for `bs_theme()`

[Experimental] `theme = bslib::bs_theme()` is accepted, but which parts of a character it colours may change in a later release.

See Also

`mascot_colour_roles()`, `mascot()`, `update_mascot()`, mascot_styling for panel and button CSS hooks, `mascot_motion_settings()` for a complete Shiny settings example

Other mascot helpers: `mascot_action()`, `mascot_anchors()`, `mascot_bubble()`, `mascot_character()`, `mascot_characters()`, `mascot_check_character()`, `mascot_colour_roles()`, `mascot_families()`, `mascot_gesture()`, `mascot_gestures()`, `mascot_mood()`, `mascot_moods()`, `mascot_motion_settings()`, `mascot_palettes()`, `mascot_placement()`, `mascot_popup_menu()`, `mascot_preview()`, `mascot_read_character()`, `mascot_size()`, `mascot_status()`

Examples

```
mascot_theme()
mascot_theme(palette = "coral_navy", light = list(rim = "#b93664"))
mascot_theme(light = list(body = "#1f6feb"), dark = list(body = "#9ec5fe"), mode = "auto")
mascot_theme(css = list(body = "--team-blue"))
ui <- mascot("helper", theme = list(body = "#1f6feb"))

ui <- mascot("brand", theme = bslib::bs_theme(primary = "#1f6feb"))
```

`update_mascot` *Update a mascot in a Shiny app*

Description

Update any combination of character, mood, status, bubble, size, placement, theme, motion, click and gesture from your Shiny server. Arguments left as `NULL` stay unchanged. Invalid values warn and leave the mascot unchanged. In Shiny development mode they also show a worried mascot with the problem as its status; the next valid update clears it. Updates sent before a mascot appears are kept until it appears.

Usage

```
update_mascot(
  id,
  ...,
  character = NULL,
  mood = NULL,
  status = NULL,
  bubble = NULL,
  size = NULL,
  placement = NULL,
  theme = NULL,
  motion = NULL,
  click = NULL,
  gesture = NULL,
  session = shiny::getDefaultReactiveDomain()
)
```

Arguments

`id`	A required non-empty string without whitespace, unique on the page. In a Shiny module use `ns("helper")` in the UI and `"helper"` in the server; `update_mascot()` adds the module namespace once.
`...`	Must be empty. Forces the arguments after it to be named.
`character`	A character id such as `"tessa"`, or a `mascot_character()` helper. `NULL` (the default) leaves the character unchanged.

`mood` A mood name or `mascot_mood()` object. `NULL` (the default) leaves the mood unchanged.

`status` A non-empty string or `mascot_status()` object. `NULL` (the default) leaves the status unchanged; `mascot_status()` hands the status back to the mood. Text is always escaped.

`bubble` A non-empty string, an `htmltools` tag or tag list, or a `mascot_bubble()` object. `NULL` (the default) leaves the bubble unchanged; `mascot_bubble()` clears it. Plain text is escaped; `htmltools::HTML()` or tags supply HTML. Tags with dependencies or head content are not supported.

`size` A pixel number, percentage string or `mascot_size()` object, as in `mascot()`. `NULL` (the default) leaves the size unchanged.

`placement` A shortcut, named list or `mascot_placement()` object, as in `mascot()`. `NULL` (the default) leaves placement unchanged.

- Size and placement changes keep the running mascot, its mood and its character.
- A new placement replaces any position the visitor dragged the mascot to.
- Floating placement shows a minimise control by default, and dragging is off.
- Inline placement ignores `minimise` and `drag`. If you set either one yourself, a message says so; the defaults give no message. The update is still sent.
- With dragging enabled, the minimised restore button can also be dragged. A click restores it; a drag leaves it minimised in its new position.

`theme` A theme in the forms accepted by `mascot()`. `NULL` (the default) leaves the theme unchanged; `"original"` or `mascot_theme()` restores the character's own colours. A palette id string chooses a named palette. After a character swap, the theme is applied again to the new character's parts. The palette carries over if the new character has one of that name. If it has not, you get a warning with class `mascot_palette_dropped`, and colours you set explicitly stay for the roles both characters share.

`motion` A logical flag or `mascot_motion_settings()` object, as in `mascot()`. `NULL` (the default) leaves motion unchanged. Changes apply to the live mascot without restarting it; the visitor's reduced-motion preference still applies.

`click` A label, a `mascot_action()` object or a `mascot_popup_menu()` object, as in `mascot()`. `NULL` (the default) leaves the click unchanged; an empty helper, `mascot_action()` or `mascot_popup_menu()`, removes it, so the mascot is a picture again. A new click replaces the old one completely and closes an open menu. Clicks and menu choices arrive as `input$<id>_click` and `input$<id>_menu` in the module that owns the mascot; the click count continues across changes.

`gesture` A gesture name or `mascot_gesture()` object. `NULL` (the default) sends no gesture. Use `"nod"` to acknowledge an action or `"wave"` to greet. Must be named. See `mascot_gestures()` for descriptions. With `gestures = FALSE` in `mascot_motion_settings()`, the gesture is skipped; the mood, bubble and status in the same update still apply. In Shiny development mode, the browser asks Shiny to show a warning that says why.

`session` A Shiny session, defaulting to the current reactive domain. Must be named when supplied. `NULL` outside a server or module is an error.

Details

In `mascot()` and its helpers, `NULL` (where accepted) means none or the default. In `update_mascot()`, `NULL` leaves a setting unchanged. To reset a setting, send an empty helper:

Setting	Reset sent to `update_mascot()`
Character	`character = mascot_character()` (back to `"tessa"`)
Mood	`mood = mascot_mood()` (back to `"idle"`)
Size	`size = mascot_size()` (160, `small_screen = "auto"`)
Custom status	`status = mascot_status()` follows the mood
Bubble	`bubble = mascot_bubble()` removes it
Theme	`theme = mascot_theme()` restores character colours
Click	`click = mascot_action()` (or `mascot_popup_menu()`) removes it
Placement	`placement = mascot_placement()` (bottom-right); `mascot_placement("inline")` for inline
Motion	`motion = mascot_motion_settings()` allows movement unless the visitor prefers reduced motion

Each non-`NULL` argument replaces that whole setting, not part of it. For example, `update_mascot("h", size = 200)` also resets a custom `small_screen` to `"auto"`.

`mascot()` and the helpers raise an error on an invalid value. `update_mascot()` warns instead, with class `mascot_update_ignored`, and leaves the mascot unchanged. That also covers a helper that raises a `mascot_error` while an argument is evaluated inside the call, such as `size = mascot_size(input$px)`; the whole update is then ignored. An invalid `id`, unknown arguments in `...` or a missing session still raise errors. See mascot-package for all condition classes.

An unknown palette id is an error in `mascot()` but only a warning in `update_mascot()`. `mascot()` runs while the UI is built, where an error shows at once. `update_mascot()` runs in a live session, where an error would stop the app for the visitor, so it warns and the mascot keeps running unchanged. The message names any other characters that have a palette of that name. (A palette carried over from before a character swap is dropped instead, with a warning of class `mascot_palette_dropped`; see the `theme` argument.)

Status text and bubbles. Use the always visible status line for a short description of the current state. Use a bubble for a message about something that happened; it may time out. Use `mascot_bubble(..., duration = Inf)` for a message the user must act on, such as an invalid form, because a timed message can vanish before it is read. Bubbles survive mood changes and swaps; a queued bubble's timer starts when it appears. See `mascot_bubble()` for timing and closing behaviour.

Shiny modules. In a module, use `mascot(ns("helper"))` in the UI and `update_mascot("helper")` in its server. A `session$ns("helper")` id is also accepted, including in nested modules; the namespace is not added twice. An invalid `id` type, missing session or extra argument is an error. Calling with all update arguments `NULL` does nothing.

Gestures. A character or mood change applies before the gesture. During a character swap the gesture is skipped, because the wave within the swap replaces it. Reduced motion skips movement. The latest gesture sent before the mascot appears plays once on appearance, after its queued character and mood are applied.

React to your app's state. Call `update_mascot()` from an observer. Shiny sends updates when the observer finishes, so for a mood to show while a long computation runs, run it as a shiny::ExtendedTask and follow its status:

```
task <- ExtendedTask$new(function(x) promises::future_promise(fit_model(x)))
observeEvent(input$run, {
  update_mascot("helper", mood = "thinking", status = "Fitting the model")
  task$invoke(data())
})
observe({
  switch(task$status(),
    success = update_mascot("helper", mood = "happy", status = "Model ready"),
   error = update_mascot("helper", mood = "worried", status = "The model failed")
  )
})
```

For form validation, observe whether the inputs are valid (yourself, or with a validation package such as `shinyvalidate`) and set the mood from it:

```
observe({
  valid <- grepl("@", input$email, fixed = TRUE)
  update_mascot(
    "helper",
    mood = if (valid) "happy" else "worried",
    status = if (valid) "Ready to submit" else "Check the email address"
  )
})
```

Queued updates. Updates sent before a mascot appears are kept for the session, also when the first mascot is created by `shiny::renderUI()` or `shiny::insertUI()`. Each call is checked as a whole before its latest values are accepted. Until the character is known, a buffer of limited size holds whole calls, and they are checked against the first character that appears. A call that fails the check adds nothing, not even its bubble or status. A mascot that replaces one with the same id gets the latest accepted state, and only the latest pending gesture plays, once. A bubble the visitor dismissed stays dismissed in the replacement; sending a bubble again starts a new one, even with the same text.

Several calls in one flush. A flush is one round of Shiny reacting to inputs. Shiny sends the mascot one message when it ends, built from every call made in it, setting by setting: a call changes only the settings it names, and when two calls name the same setting the later call wins. If a reaction to a click sets `mood = "happy"` and a state observer sets `mood = "worried"` in the same flush, the observer that runs last decides the face and the other mood is never shown. `status` and `bubble` work the same way, and only the latest `gesture` plays. Give each setting one owner, so no call can overwrite another's:

```
# The event reaction owns the bubble and gesture, the observer owns the mood.
observeEvent(input$save, {
  update_mascot("helper", bubble = "Saved", gesture = "nod")
})
observe({
  update_mascot("helper", mood = if (valid()) "idle" else "worried")
})
```

After a custom status. A custom status stays until you replace it or hand it back with `status = mascot_status()`. A call that sets only `mood` changes the face and leaves the status text as it was, so "Fitting the model" can sit beside a happy face. To let the status follow the mood again, send `update_mascot("helper", mood = "happy", status = mascot_status())`.

What a re-created mascot takes from its UI call. A mascot that `shiny::renderUI()` draws again, or that is removed and added back, receives the latest accepted state (see "Queued updates"). For each setting an earlier accepted call to `update_mascot()` has set, the value in the `mascot()` call is ignored; the settings the server never set come from the `mascot()` call. A `mood`, `status` or `size` in the UI call therefore only matters until the server first sets that setting, and after that the UI call can use the defaults. A call that changes `character` also sets the mood (adapted to the new character), so after it the UI call's `mood` is ignored too.

A mascot in a modal. The mascot is off the page while the modal is closed. Updates sent then are kept, as under "Queued updates", and applied when the modal opens: the last value of each setting, a bubble sent in the meantime, and a gesture, which plays once as the mascot appears. So it can open with the mood, status or an unfinished bubble from the last visit, or with whatever was sent while the modal was closed. To open it clean, reset those settings as the modal opens (a reset is an update, so it also replaces the UI call's values for them) and send no gesture while it is closed. A reset cannot cancel a gesture already sent. Sending updates only while the modal is open avoids what was sent while it was closed, but not the state left from the last visit.

```
observeEvent(input$help, {
  update_mascot(
    "helper",
    mood = mascot_mood(),
    status = mascot_status(),
    bubble = mascot_bubble()
  )
  showModal(modalDialog(mascot("helper"), title = "Help"))
})
```

Value

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

See Also

`mascot()`, `mascot_motion_settings()` for a Shiny example that switches idle movement and plays a nod.

Other mascots in Shiny: `mascot()`

Examples

```
shiny::shinyApp(
  ui = shiny::fluidPage(
    mascot("helper"),
    shiny::actionButton("work", "Think"),
    shiny::actionButton("done", "Finish")
  ),
  server = function(input, output, session) {
```

```
    shiny::observeEvent(input$work, {
      update_mascot("helper", mood = "thinking", status = "Working")
    })
    shiny::observeEvent(input$done, {
      update_mascot("helper", mood = "happy", status = mascot_status(),
        bubble = "Finished", gesture = "nod")
    })
  }
)
```

Index

∗ **mascot examples**
mascot_example, 16
mascot_examples, 17
mascot_studio, 34
∗ **mascot helpers**
mascot_action, 6
mascot_anchors, 8
mascot_bubble, 9
mascot_character, 10
mascot_characters, 13
mascot_check_character, 14
mascot_colour_roles, 15
mascot_families, 18
mascot_gesture, 18
mascot_gestures, 19
mascot_mood, 20
mascot_moods, 21
mascot_motion_settings, 22
mascot_palettes, 24
mascot_placement, 25
mascot_popup_menu, 27
mascot_preview, 29
mascot_read_character, 31
mascot_size, 32
mascot_status, 33
mascot_theme, 37
∗ **mascots in Shiny**
mascot, 3
update_mascot, 40

bslib::bs_theme(), *4*, *30*, *37–39*

grDevices::rgb(), *37*

htmltools::HTML(), *3*, *9*, *30*, *33*, *41*
htmltools::htmlDependency(), *5*

mascot, 3
mascot(), *4*, *6*, *7*, *10*, *11*, *13*, *16*, *20–23*, *26*, *27*, *29–31*, *33*, *34*, *37–39*, *41*, *42*, *44*
mascot-package, *4*, *42*
mascot_action, 6
mascot_action(), *4*, *6*, *8*, *10*, *11*, *14–16*, *18–24*, *26–29*, *31–34*, *39*, *41*
mascot_anchors, 8
mascot_anchors(), *7*, *10*, *11*, *14–16*, *18–24*, *26*, *29*, *31–34*, *39*
mascot_bubble, 9
mascot_bubble(), *3*, *7*, *8*, *11*, *14–16*, *18–24*, *26*, *28–34*, *39*, *41*, *42*
mascot_character, 10
mascot_character(), *3*, *7*, *8*, *10*, *14–16*, *18–24*, *26*, *29–34*, *39*, *40*
mascot_character_format, 11, *15*, *32*
mascot_characters, 13
mascot_characters(), *7*, *8*, *10*, *11*, *15*, *16*, *18–24*, *26*, *29*, *31–34*, *39*
mascot_check_character, 14
mascot_check_character(), *7*, *8*, *10*, *11*, *13*, *14*, *16*, *18–24*, *26*, *29*, *31–34*, *39*
mascot_colour_roles, 15
mascot_colour_roles(), *7*, *8*, *10*, *11*, *14*, *15*, *18–24*, *26*, *29*, *31–34*, *37*, *39*
mascot_example, 16
mascot_example(), *17*, *35*
mascot_examples, 17
mascot_examples(), *16*, *17*, *35*
mascot_families, 18
mascot_families(), *7*, *8*, *10*, *11*, *13–16*, *19–24*, *26*, *29*, *31–34*, *39*
mascot_gesture, 18
mascot_gesture(), *7*, *8*, *10*, *11*, *14–16*, *18*, *20–24*, *26*, *29*, *31–34*, *39*, *41*
mascot_gestures, 19
mascot_gestures(), *7*, *8*, *10*, *11*, *14–16*, *18*, *19*, *21–24*, *26*, *29*, *31–34*, *39*, *41*
mascot_mood, 20
mascot_mood(), *3*, *7*, *8*, *10*, *11*, *14–16*, *18–24*, *26*, *29–34*, *39*, *41*

mascot_moods, 21
mascot_moods(), *3*, *7, 8*, *10, 11*, *13–16*, *18–21*, *23, 24*, *26*, *29–34*, *39*
mascot_motion_settings, 22
mascot_motion_settings(), *4*, *7, 8*, *10, 11*, *14–16*, *18–22*, *24*, *26*, *29–34*, *39*, *41*, *44*
mascot_palettes, 24
mascot_palettes(), *4*, *7, 8*, *10, 11*, *13–16*, *18–23*, *26*, *29–34*, *37*, *39*
mascot_placement, 25
mascot_placement(), *4*, *7, 8*, *10, 11*, *14–16*, *18–24*, *29*, *31–36*, *39*, *41*
mascot_popup_menu, 27
mascot_popup_menu(), *4*, *6–11*, *14–16*, *18–24*, *26*, *31–34*, *39*, *41*
mascot_preview, 29
mascot_preview(), *7, 8*, *10, 11*, *14–16*, *18–24*, *26*, *29*, *32–34*, *38, 39*
mascot_read_character, 31
mascot_read_character(), *7, 8*, *10, 11*, *13–16*, *18–24*, *26*, *29*, *31*, *33, 34*, *39*
mascot_size, 32
mascot_size(), *3, 4*, *7, 8*, *10, 11*, *14–16*, *18–24*, *26*, *29–32*, *34–36*, *39*, *41*
mascot_status, 33
mascot_status(), *3*, *7, 8*, *10, 11*, *14–16*, *18–24*, *26*, *29–33*, *39*, *41*
mascot_studio, 34
mascot_studio(), *16, 17*
mascot_styling, 35, *39*
mascot_theme, 37
mascot_theme(), *4*, *7, 8*, *10, 11*, *14–16*, *18–24*, *26*, *29–36*

shiny::ExtendedTask, *42*
shiny::in_devmode(), *38*
shiny::insertUI(), *43*
shiny::moduleServer(), *5*
shiny::renderUI(), *5*, *43, 44*
shiny::runApp(), *16, 17*, *35*
shiny::runExample(), *16*
shiny::uiOutput(), *5*

update_mascot, 40
update_mascot(), *3, 4*, *6, 7*, *10*, *18–21*, *23*, *26, 27*, *29*, *33, 34*, *37–40*, *42*