

# Package ‘shinygenui’

September 22, 2026

**Title** Generative UI for 'shiny'

**Version** 0.2.0

**Description** Build interactive user interfaces for 'shiny' applications through a conversation with a large language model (LLM). Developers choose a set of reusable components, and the model arranges and updates those components as the user describes what they need. Each component's inputs are checked before it is shown, and the model supplies data rather than executable code. Applications can also save and replay the sequence of interface changes without contacting a model. For background on generative user interfaces, see Leviathan et al. (2026) <[doi:10.48550/arXiv.2604.09577](https://doi.org/10.48550/arXiv.2604.09577)>.

**License** MIT + file LICENSE

**URL** <https://nanx.me/shinygenui/>,  
<https://github.com/nanxstats/shinygenui>

**BugReports** <https://github.com/nanxstats/shinygenui/issues>

**Encoding** UTF-8

**Depends** R (>= 4.1.0)

**Imports** bslib, cli, ellmer (>= 0.4.1), htmltools, jsonlite, R6, rlang, shiny, shinychat (>= 0.5.0), stats, utils, whisker

**Suggests** DT, ggplot2, knitr, rmarkdown, shinytest2, testthat (>= 3.0.0), withr

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**Config/roxygen2/version** 8.1.0

**NeedsCompilation** no

**Author** Nan Xiao [aut, cre, cph] (ORCID:  
<<https://orcid.org/0000-0002-0250-5673>>)

**Maintainer** Nan Xiao <[me@nanx.me](mailto:me@nanx.me)>

**Repository** CRAN

**Date/Publication** 2026-09-22 02:20:02 UTC

Contents

genui\_call . . . . . 2

genui\_canvas . . . . . 3

genui\_card\_row . . . . . 3

genui\_catalog . . . . . 4

genui\_component . . . . . 5

genui\_components\_bslib . . . . . 6

genui\_dispatch . . . . . 7

genui\_prompt . . . . . 8

genui\_replay . . . . . 9

genui\_server . . . . . 10

genui\_trace . . . . . 12

Index 14

---

|            |                               |
|------------|-------------------------------|
| genui_call | Create a normalized tool call |
|------------|-------------------------------|

---

Description

A `genui_call` is the normalized representation of one model-issued tool call: the tool name plus its arguments as plain data. `genui_dispatch()` consumes these. In production they are produced by the compiled ellmer tools; in tests and replays you can hand-build them.

Usage

```
genui_call(tool, args = list())
```

Arguments

|      |                                                                                                                                                                                  |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| tool | Tool name: a component name from the catalog, or one of the built-ins "update_component", "remove_component", "clear_canvas".                                                    |
| args | Named list of arguments as supplied by the model. For update_component, args must contain id and args (the partial arguments to merge, as a named list or a JSON object string). |

Value

A `genui_call` object.

Examples

```
genui_call("scatter_plot", list(x = "mpg", y = "hp"))
genui_call("update_component", list(id = "c1", args = list(x = "wt")))
genui_call("remove_component", list(id = "c1"))
```

---

|              |                                                  |
|--------------|--------------------------------------------------|
| genui_canvas | <i>Canvas container for generated components</i> |
|--------------|--------------------------------------------------|

---

## Description

Place this anywhere in your UI, typically as the main content next to a `shinychat::chat_ui()` sidebar. Components emitted by the model are inserted into it progressively as tool calls complete. Pair it with a `genui_server()` call using the same id.

## Usage

```
genui_canvas(id, ..., placeholder = "Components will appear here.")
```

## Arguments

|             |                                                                                |
|-------------|--------------------------------------------------------------------------------|
| id          | Module id, matching the id passed to <code>genui_server()</code> .             |
| ...         | Attributes and initial children added to the canvas <code>&lt;div&gt;</code> . |
| placeholder | Text shown while the canvas is empty.                                          |

## Value

An `htmltools::tag()` object.

## Examples

```
genui_canvas("canvas")
```

---

|                |                                                   |
|----------------|---------------------------------------------------|
| genui_card_row | <i>Container component: a titled row of cards</i> |
|----------------|---------------------------------------------------|

---

## Description

A container the model can place other components into: create the row, then create children with `parent_id` set to the row's instance id. Children lay out in a responsive grid inside the row's card. Removing the row removes its children. Add it to your `genui_catalog()` alongside `genui_components_bslib()` (or your own components) to let the model group related views.

## Usage

```
genui_card_row()
```

## Value

A `genui_component()` object with `container = TRUE`.

## Examples

```
catalog <- genui_catalog(
  genui_card_row(),
  genui_component(
    name = "note_card",
    description = "A card showing a short note.",
    args = list(text = "The note text."),
    ui = function(id, args) htmltools::p(args$text)
  )
)
```

---

genui\_catalog

*Collect components into a catalog*


---

## Description

A catalog is the finite set of components the model is allowed to render. Each component compiles to one `ellmer::tool()`; the built-in lifecycle tools (`update_component`, `remove_component`, `clear_canvas`) are registered alongside it by `genui_server()`.

## Usage

```
genui_catalog(...)
```

## Arguments

... `genui_component()` objects, or lists of them (so component packs such as `genui_components_bslib()` can be passed directly).

## Value

A `genui_catalog` object: a named list of components, keyed by component name.

## Examples

```
note <- genui_component(
  name = "note_card",
  description = "A card showing a short note.",
  args = list(text = "The note text."),
  ui = function(id, args) htmltools::p(args$text)
)
catalog <- genui_catalog(note)
names(catalog)
```

---

|                 |                                         |
|-----------------|-----------------------------------------|
| genui_component | <i>Define a generative UI component</i> |
|-----------------|-----------------------------------------|

---

## Description

A component is one entry in the finite catalog that constrains what the model can render. It couples a model-facing tool schema (name, description, typed args) with developer-written rendering code: a ui function plus an optional server function, instantiated together as a dynamic Shiny module every time the model creates or updates an instance. The model only ever supplies data arguments validated against the declared types; it never emits code.

## Usage

```
genui_component(
  name,
  description,
  args = list(),
  ui,
  server = NULL,
  check = NULL,
  container = FALSE,
  width = c("auto", "wide", "full")
)
```

## Arguments

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| name        | Tool name the model sees. A snake_case string: lowercase letters, digits, and underscores, starting with a letter. Must be unique within a catalog and must not collide with the built-in lifecycle tools (update_component, remove_component, clear_canvas).                                                                                                                                                                                                                                                     |
| description | One to three sentences telling the model what the component shows and when to use it. This is the model-facing documentation for the component, so write it well.                                                                                                                                                                                                                                                                                                                                                 |
| args        | Named list of ellmer type specifications (for example <code>ellmer::type_string()</code> , <code>ellmer::type_enum()</code> ) describing the arguments the model may supply. As a shorthand, a plain string is promoted to <code>ellmer::type_string(&lt;string&gt;)</code> . Argument names <code>id</code> and <code>parent_id</code> are reserved by the package. Catalogs are often built inside the Shiny server function so enums can enumerate live facts, such as the column names of the active dataset. |
| ui          | function(id, args) returning an <code>htmltools::tag()</code> (or tag list). <code>id</code> is the fully namespaced module id for this instance; use <code>shiny::NS(id)</code> to namespace any embedded inputs and outputs. <code>args</code> is the validated argument list.                                                                                                                                                                                                                                  |
| server      | Optional function(id, args, data) that calls <code>shiny::moduleServer()</code> to wire outputs and embedded inputs. <code>data</code> is the reactive passed to <code>genui_server()</code> . If the module creates observers, include them in the module's return value (alone or inside a list) so the package can destroy them when the instance is updated                                                                                                                                                   |

|           |                                                                                                                                                                                                                                     |
|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|           | or removed. A reactivities element in the return value is stored in the instance registry keyed by instance id; nothing reads it in v0.1, it is the hook for cross-component reactivity in a later release.                         |
| check     | Optional function(args, data) for semantic validation beyond the JSON schema. Return NULL when args are acceptable, or a string describing the problem; the string is returned to the model as a tool error so it can self-correct. |
| container | If TRUE, this component can host child instances: its ui must render an element with id shiny::NS(id, "slot"), and other components may target it by passing the instance's id as parent_id.                                        |
| width     | Layout hint for the canvas grid: "auto" (default), "wide", or "full".                                                                                                                                                               |

### Value

A genui\_component object.

### Examples

```
genui_component(
  name = "note_card",
  description = "A card showing a short markdown note. Use for narrative
    text that should live on the canvas rather than in the chat.",
  args = list(
    title = "A short title for the card.",
    text = ellmer::type_string("The markdown body text.")
  ),
  ui = function(id, args) {
    htmltools::div(
      htmltools::h5(args$title),
      htmltools::p(args$text)
    )
  }
)
```

---

genui\_components\_bslib

*Starter component pack built on bslib*

---

### Description

A small, general-purpose catalog: a value box, a Markdown card, a data table, a scatter plot, and a histogram with an embedded bin-count slider (the reference interactive component: dragging the slider re-renders at Shiny speed with no LLM round trip). Pass the result to [genui\\_catalog\(\)](#), optionally alongside your own components.

### Usage

```
genui_components_bslib(data = NULL)
```

## Arguments

**data** Optional data frame used to ground column arguments as enums at catalog build time. Typically the same data your `genui_server()` data reactive returns. With NULL, column arguments are free-form strings validated only by the `check()` hooks.

## Details

When data is supplied, column arguments become enums of the actual column names, so a hallucinated column is a schema violation the model must correct. Each component also validates columns against the live data through its `check()` hook at render time.

## Value

A list of `genui_component()` objects.

## Examples

```
catalog <- genui_catalog(genui_components_bslib(data = mtcars))
names(catalog)
```

---

|                |                                                   |
|----------------|---------------------------------------------------|
| genui_dispatch | <i>Validate one tool call and plan its effect</i> |
|----------------|---------------------------------------------------|

---

## Description

The pure core of shinygenui: given the catalog, one normalized call, and the current instance state, either return a plan describing what should happen (create, update, remove, or clear) or signal a classed error whose message is written for the model to read and correct. No Shiny session, registry mutation, or LLM is involved; the Shiny executor applies the returned plan.

## Usage

```
genui_dispatch(catalog, call, state, data = NULL)
```

## Arguments

**catalog** A `genui_catalog()`.

**call** A `genui_call()` (or a plain list with tool and args).

**state** Current instance state: a GenuiRegistry or a list with instances (a named list of `list(component, args, parent_id)` keyed by instance id) and `next_id` (integer).

**data** Current value of the app's data object, passed to component `check()` hooks. Typically `shiny::isolate(data())` at call time.

**Value**

A `genui_plan` object, a list with at least `action` (one of "create", "update", "remove", "clear") plus the fields the executor needs: `id`, `component`, `args` (full args to render), `delta` (validated partial args, updates only), `parent_id`, and `ids` (teardown order, removes and clears only).

**Examples**

```
catalog <- genui_catalog(
  genui_component(
    name = "note_card",
    description = "A card showing a short note.",
    args = list(text = "The note text."),
    ui = function(id, args) htmltools::p(args$text)
  )
)
state <- list(instances = list(), next_id = 1L)
genui_dispatch(catalog, genui_call("note_card", list(text = "hi")), state)
```

---

genui\_prompt

---

*Assemble the system prompt from a catalog*


---

**Description**

Renders the packaged whisker template (`inst/prompts/system.md`) with the catalog's components and an optional developer-supplied context string. The result instructs the model to narrate briefly in chat while placing visuals through tools, to reuse `update_component` when the user refines an existing view, and to prefer few, dense components.

**Usage**

```
genui_prompt(catalog, context = NULL, template = NULL)
```

**Arguments**

|                       |                                                                                                                                                                                                                                                                                    |
|-----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>catalog</code>  | A <code>genui_catalog()</code> .                                                                                                                                                                                                                                                   |
| <code>context</code>  | Optional string appended as an "App context" section: the data schema, a few sample rows, a business glossary; whatever the model needs to ground its answers. Raw data is never included unless you put it here yourself.                                                         |
| <code>template</code> | Optional path to an alternative whisker template, or a template string. It receives components (each with <code>name</code> , <code>description</code> , <code>container</code> , and formatted <code>args</code> lines), <code>has_containers</code> , and <code>context</code> . |

**Value**

A string: the assembled system prompt.

## Examples

```
catalog <- genui_catalog(
  genui_component(
    name = "note_card",
    description = "A card showing a short note.",
    args = list(text = "The note text."),
    ui = function(id, args) htmltools::p(args$text)
  )
)
cat(genui_prompt(catalog, context = "The data is mtcars."))
```

---

genui\_replay

Rebuild a canvas from a saved trace, no LLM required

---

## Description

Folds a trace (from [genui\\_trace\(\)](#), or the trace reactive returned by [genui\\_server\(\)](#)) through the same validate-and-execute pipeline the model's tool calls use, recreating every instance in a [genui\\_canvas\(\)](#) with id target. Instance ids come back identical because ids are assigned deterministically and never reused. Embedded inputs return at their default values: input state is ephemeral by design and not part of the trace.

## Usage

```
genui_replay(
  trace,
  catalog,
  target,
  data = NULL,
  session = shiny::getDefaultReactiveDomain()
)
```

## Arguments

|         |                                                                                                                                                      |
|---------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| trace   | A trace list, as returned by <a href="#">genui_trace()</a> (already evaluated, not the reactive) or restored via <a href="#">readRDS()</a> .         |
| catalog | The <a href="#">genui_catalog()</a> to render with; component names in the trace must exist in it.                                                   |
| target  | Module id of the <a href="#">genui_canvas()</a> to rebuild into. Use a canvas of its own, not one already driven by <a href="#">genui_server()</a> . |
| data    | Optional reactive (or function) returning the data object, passed to component servers and check() hooks, as in <a href="#">genui_server()</a> .     |
| session | The Shiny session (defaults to the current reactive domain).                                                                                         |

## Details

Entries that no longer validate (for example after the catalog changed) are skipped with a warning; the rest of the trace still replays. The target canvas is emptied first, so replaying twice is idempotent.

**Value**

(Invisibly) a list with reactivities trace and instances describing the rebuilt canvas, as in `genui_server()`.

**Examples**

```
note <- genui_component(
  name = "note_card",
  description = "A card showing a short note.",
  args = list(text = "The note text."),
  ui = function(id, args) htmltools::p(args$text)
)
catalog <- genui_catalog(note)
saved_trace <- list(list(
  op = "create",
  id = "c1",
  component = "note_card",
  args = list(text = "Hello")
))
shiny::testServer(
  function(input, output, session) {
    suppressWarnings(genui_replay(saved_trace, catalog, target = "canvas"))
  },
  {
    stopifnot(identical(names(session$returned$instances()), "c1"))
  }
)
```

---

genui\_server

---

*Server logic for a generative UI canvas*


---

**Description**

Wires a `genui_catalog()` to an ellmer Chat: compiles every component into a schema-validated tool, registers the built-in lifecycle tools (`update_component`, `remove_component`, `clear_canvas`), installs the assembled system prompt, and executes validated tool calls against the `genui_canvas()` with the matching id. Components stream onto the canvas progressively as the model emits tool calls; validation and rendering failures are returned to the model as tool errors and never crash the session.

**Usage**

```
genui_server(
  id,
  catalog,
  chat,
  data = NULL,
  chat_id = NULL,
  greeting = NULL,
  system_prompt = NULL
)
```

## Arguments

|               |                                                                                                                                                                                                              |
|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| id            | Module id, matching the <code>genui_canvas()</code> id.                                                                                                                                                      |
| catalog       | A <code>genui_catalog()</code> .                                                                                                                                                                             |
| chat          | An ellmer Chat object (any provider). Its system prompt is replaced with <code>system_prompt</code> .                                                                                                        |
| data          | A reactive (or function) returning the app's current data object. It is passed to component <code>server()</code> functions and, isolated, to <code>check()</code> hooks at validation time.                 |
| chat_id       | Id of a <code>shinychat::chat_ui()</code> placed in the UI at the same namespace level as <code>genui_canvas()</code> , or NULL (default) to manage the chat loop yourself.                                  |
| greeting      | Optional markdown welcome message, shown while the chat is empty and dismissed after the first user message (only used when <code>chat_id</code> is set). It is display-only and never sent to the model.    |
| system_prompt | System prompt to install on chat. Defaults to <code>genui_prompt()</code> of the catalog; use <code>genui_prompt(catalog, context = ...)</code> to add app context, or pass any string to take full control. |

## Details

With `chat_id`, the package connects a `shinychat::chat_ui()` you placed in the UI using `shinychat::chat_server()`. It handles asynchronous streaming, cancellation, and attachments. Set `allow_attachments = FALSE` in `chat_ui()` to disable uploads; attachments provide context to the model and do not replace the component data. Conversation history is disabled because restoring a conversation also requires restoring its canvas and registry. With `chat_id = NULL` you run your own loop on chat; the registered tools work all the same.

Create the Chat object inside your server function, one per session. Sharing a single Chat across sessions would cross-wire the tool closures and leak conversation history between users.

## Value

(Invisibly) a list with `chat` (the wired Chat), and the reactivities `trace` (the ordered call trace, see `genui_trace()`) and `instances` (the live instance state, a named list keyed by id).

## Update semantics

`update_component` re-instantiates the component's module with the merged arguments inside the instance's stable shell: same canvas position, same ids, no page flicker. Because the module restarts, embedded input state (like the starter histogram's bin slider) resets to its defaults on update; snapshotting and restoring embedded input values across updates is explicitly future work.

## Error feedback

Any failure while handling a tool call — schema validation, a component's `check()` hook, or a rendering error — is signaled as a regular R condition. ellmer catches it and returns `conditionMessage()` to the model as the tool error, so the model can correct its arguments and retry; the Shiny session itself never crashes, and a failed call never leaves a half-rendered component behind. Failures are always logged to the app's server log; set `options(shinygenui.verbose = TRUE)` to also log successful canvas operations.

## Examples

```
note <- genui_component(
  name = "note_card",
  description = "A card showing a short note.",
  args = list(text = "The note text."),
  ui = function(id, args) htmltools::p(args$text)
)
catalog <- genui_catalog(note)
chat <- ellmer::chat_openai(
  model = "gpt-5.6-sol",
  credentials = function() list(api_key = "not-used")
)
shiny::testServer(
  genui_server,
  args = list(id = "canvas", catalog = catalog, chat = chat),
  {
    stopifnot("note_card" %in% names(chat$get_tools()))
  }
)
```

---

genui\_trace

*Read the ordered call trace of a canvas*


---

## Description

The trace is the spec of record for a canvas: every validated call the model made (create, update, remove, clear), in order, as plain JSON-friendly lists. [genui\\_replay\(\)](#) can rebuild the canvas from it with no LLM configured. Embedded input state is intentionally ephemeral and never recorded.

## Usage

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

## Arguments

|         |                                                                                                                                    |
|---------|------------------------------------------------------------------------------------------------------------------------------------|
| session | The Shiny session (defaults to the current reactive domain).                                                                       |
| id      | The <a href="#">genui_server()</a> module id to read, relative to session. May be omitted when the session has exactly one canvas. |

## Value

A reactive expression returning the trace: a list of entries of the form `list(op = "create", id = "c1", component = "...", args = list(...))` (plus `parent_id` for children; update entries carry the validated argument delta). Persist it with [saveRDS\(\)](#) to replay in a later session.

**Examples**

```
note <- genui_component(  
  name = "note_card",  
  description = "A card showing a short note.",  
  args = list(text = "The note text."),  
  ui = function(id, args) htmltools::p(args$text)  
)  
catalog <- genui_catalog(note)  
chat <- ellmer::chat_openai(  
  model = "gpt-5.6-sol",  
  credentials = function() list(api_key = "not-used")  
)  
shiny::testServer(  
  genui_server,  
  args = list(id = "canvas", catalog = catalog, chat = chat),  
  {  
    trace <- genui_trace(session)  
    stopifnot(identical(trace(), list()))  
  }  
)
```

# Index

`ellmer::tool()`, [4](#)  
`ellmer::type_enum()`, [5](#)  
`ellmer::type_string()`, [5](#)

`genui_call`, [2](#)  
`genui_call()`, [7](#)  
`genui_canvas`, [3](#)  
`genui_canvas()`, [9–11](#)  
`genui_card_row`, [3](#)  
`genui_catalog`, [4](#)  
`genui_catalog()`, [3, 6–11](#)  
`genui_component`, [5](#)  
`genui_component()`, [3, 4, 7](#)  
`genui_components_bslib`, [6](#)  
`genui_components_bslib()`, [3, 4](#)  
`genui_dispatch`, [7](#)  
`genui_dispatch()`, [2](#)  
`genui_prompt`, [8](#)  
`genui_prompt()`, [11](#)  
`genui_replay`, [9](#)  
`genui_replay()`, [12](#)  
`genui_server`, [10](#)  
`genui_server()`, [3–5, 7, 9, 10, 12](#)  
`genui_trace`, [12](#)  
`genui_trace()`, [9, 11](#)

`htmltools::tag()`, [3, 5](#)

`readRDS()`, [9](#)

`saveRDS()`, [12](#)  
`shiny::moduleServer()`, [5](#)  
`shinychat::chat_server()`, [11](#)  
`shinychat::chat_ui()`, [3, 11](#)