Package 'shinysnap'

September 28, 2026

Title Save and Restore the State of 'shiny' Applications

Version 0.1.0

Description Save the state of applications built with 'shiny', the web application framework by Chang et al. (2026) <doi:10.32614/CRAN.package.shiny>. Users can share their work and continue it in another session. Input values and selected values held by the server are saved in JSON (JavaScript Object Notation) files that can be read and edited by hand. Saved state can be restored without reloading the page or setting up bookmarking. Restoration waits for inputs that appear as the page changes and reports which values were restored, missing, or could not be applied.

License MIT + file LICENSE

URL https://nanx.me/shinysnap/, https://github.com/nanxstats/shinysnap

BugReports https://github.com/nanxstats/shinysnap/issues

Encoding UTF-8

Imports htmltools, jsonlite, promises, R6, shiny (>= 1.8.0), utils, zmij

Suggests bslib, chromote, httpuv, knitr, later, rmarkdown, shinyMatrix, shinytest2, testthat (>= 3.0.0), withr, zip

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>

Repository CRAN

Date/Publication 2026-09-28 09:10:07 UTC

Contents

`print.shinysnap_restore` *Print, format, and coerce snapshots and restore results*

Description

`print()` shows a compact summary: app name and version, creation time, and the number and names of inputs, values, and attachments. `format()` returns the same lines as a character vector. `as.list()` drops the class and returns the underlying list.

Usage

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

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

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

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

## S3 method for class 'shinysnap'
```

```
as.list(x, ...)

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

Arguments

x	A snapshot object, a snapshot comparison from snap_diff(), or a restore handle or report from snap_restore(). format() and as.list() accept snapshot objects only.
...	Passed to print.data.frame() when printing a snapshot comparison or restore report; ignored otherwise.

Details

For a snap_diff() result, print() shows the changed entries and their old and new values. For a snap_restore() handle, it shows the transaction id; for a completed restore report, it shows the input statuses and the elapsed time.

Value

print() returns x invisibly; format() a character vector; as.list() a plain list.

Examples

```
snap <- snap_unserialize('{
  "format": 1,
  "app": {"name": "myapp", "version": "2.4.1"},
  "created": "2026-09-16T18:22:03Z",
  "inputs": {"n": 100, "rate": 0.025}
}')
print(snap)
format(snap)
names(as.list(snap))
print(snap_diff(snap, list(inputs = list(n = 50, rate = 0.025))))

if (interactive()) {
  library(shiny)
  ui <- fluidPage(
    numericInput("n", "n", 0),
    actionButton("restore", "Restore state")
  )
  server <- function(input, output, session) {
    observeEvent(input$restore, {
      handle <- snap_restore(list(inputs = list(n = 100)), on_done = print)
      print(handle)
    })
  }
  shinyApp(ui, server)
}
```

snap_as_bookmark_url *Interoperate with bookmarks, tests, and bundles*

Description

- `snap_as_bookmark_url()` encodes a snapshot the way shiny's URL bookmarking (`enableBookmarking("url")`) does, so that the state can be opened by reloading the app at that URL. The app must have URL bookmarking enabled and a UI *function* for the URL to restore; the query string carries the same `_inputs_` and `_values_` keys that `session$doBookmark()` writes, with input ids in the order stored in the snapshot.
- `snap_as_test_inputs()` returns the input values as a named list for `shiny::testServer()`: `session$setInputs(!!!snap_as_test_inputs(x))`.
- `snap_attachment()` returns the local path(s) of the file(s) a `fileInput()` held when the snapshot was taken, if the snapshot came from a bundle (see `snap_write()`) or from the same session.

Usage

```
snap_as_bookmark_url(x, session = NULL, base_url = NULL)

snap_as_test_inputs(x)

snap_attachment(x, id)
```

Arguments

`x`	A snapshot object, a list of snapshot fields, a file path, or JSON text.
`session`	A Shiny session whose `clientData` provides the base URL (protocol, host, port, and path), or `NULL`.
`base_url`	The base URL to prepend, for example `"https://example.org/app/"`. When neither `session` nor `base_url` is given, only the query string (starting with ?) is returned.
`id`	An input id.

Value

`snap_as_bookmark_url()` returns a string. `snap_as_test_inputs()` returns a named list. `snap_attachment()` returns a character vector of file paths (one per uploaded file), or `NULL` when the snapshot has no attachment for `id`.

Examples

```
snap <- snap_unserialize('{
  "format": 1,
  "inputs": {"n": 100, "method": "b", "weights": [0.5, 0.75]},
  "values": {"note": "baseline"}
```

```
}')
snap_as_bookmark_url(snap, base_url = "https://example.org/app/")
str(snap_as_test_inputs(snap))
snap_attachment(snap, "upload")
```

---

snap_dependency *The client-side script as an HTML dependency*

Description

The `snap_*()` server functions inject this script into the page on first use, so including it in the UI is optional. Include it explicitly when you want the script to load with the page, before the server function runs.

Usage

```
snap_dependency()
```

Value

An `htmltools::htmlDependency()`.

Examples

```
snap_dependency()
```

---

snap_diff *Compare two snapshots*

Description

Lists the inputs, values, and metadata entries that differ between two snapshots. Values are flattened one level, so a tracked `reactiveValues` stored as `prefs` with a changed `digits` field shows up as `prefs$digits`.

Usage

```
snap_diff(a, b)
```

Arguments

a, b	Snapshot objects, lists of snapshot fields, file paths, or JSON text.

Value

A data frame of class `shinysnap_diff` with one row per difference and the columns `id`, `section` (`"inputs"`, `"values"`, or `"meta"`), `status` (`"added"`, `"removed"`, or `"changed"`, seen from `a` to `b`), and the list columns `old` and `new` holding the values (`NULL` for the side where the entry is absent).

Examples

```
a <- snap_unserialize('{"format": 1, "inputs": {"n": 1, "x": "old"},
  "values": {"prefs": {"digits": 3}}}')
b <- snap_unserialize('{"format": 1, "inputs": {"n": 1, "y": true},
  "values": {"prefs": {"digits": 4}}}')
snap_diff(a, b)
```

---

snap_download_button *Save-state button and its download handler*

Description

`snap_download_button()` is a `downloadButton()` with the client script attached. `snap_download_handler()` defines the matching download: it takes a snapshot with `snap_take()` when the button is clicked and writes it with `snap_write()`. Together they replace the usual `downloadHandler()` boilerplate.

Usage

```
snap_download_button(
  id,
  label = "Save state",
  icon = shiny::icon("download"),
  class = NULL,
  ...
)

snap_download_handler(
  id,
  filename = "state",
  format = c("json", "zip"),
  ...,
  snapshot = NULL,
  session = shiny::getDefaultReactiveDomain()
)
```

Arguments

`id`	The output id of the button. `snap_download_handler()` accepts a vector of ids, so that several buttons (for example one per tab) share one definition.
`label`, `icon`, `class`, `...`	Passed on to `shiny::downloadButton()`. In `snap_download_handler()`, `...` is passed on to `snap_take()`.
`filename`	The file name without extension: a string, a function, or a reactive (for example a text input where the user names the file). It is reduced to the characters `A-Z a-z 0-9 . _ -`, falls back to `"state"` when empty, and gets the format's extension appended.

format	The file format: "json" (the default) or "zip" for a bundle that includes uploaded files (see snap_write()).
snapshot	NULL to take a snapshot when the button is clicked (the default), a snapshot object, or a function or reactive returning one.
session	The Shiny session. Defaults to the current session.

Value

snap_download_button() returns a tag. snap_download_handler() returns the ids invisibly.

Examples

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

  ui <- fluidPage(
    sliderInput("n", "n", 1, 100, 50),
    textInput("filename", "File name", "state"),
    snap_download_button("save")
  )

  server <- function(input, output, session) {
    snap_enable(app = "demo", version = "1.0.0", exclude = "^filename$")
    snap_download_handler("save", filename = reactive(input$filename))
  }

  shinyApp(ui, server)
}
```

---

snap_enable *Configure shinysnap for a session*

Description

Sets the app identity and the defaults used by snap_take() and snap_restore() for the current session. Calling it is optional: every snap_*() server function creates the per-session state on first use with the defaults below. Call it again to change the settings.

Usage

```
snap_enable(
  app = NULL,
  version = NULL,
  exclude = NULL,
  include = NULL,
  live_only = TRUE,
  use_restore_context = TRUE,
  verbose = FALSE,
  session = shiny::getDefaultReactiveDomain()
)
```

Arguments

app, version	The name and version of the app, written into every snapshot file. They default to the options shinysnap.app and shinysnap.version, which packaged apps can set once in .onLoad().
exclude, include	Character vectors of regular expressions matched against fully namespaced input ids. Matching exclude patterns are never captured; when include is given, only matching ids are. Ids excluded with shiny's setBookmarkExclude() are always honoured too.
live_only	Capture only inputs that are currently on the page (as reported by the client script), dropping values of inputs whose UI has been removed. Set to FALSE to capture every value shiny remembers.
use_restore_context	Prime shiny's restoreInput() mechanism during a restore, so that dynamic UI re-rendered during the restore is built with the restored values. Turn off only to debug.
verbose	Print messages about what is captured, dropped, and restored.
session	The Shiny session. Defaults to the current session.

Value

The session's controller, an R6 object, invisibly. It holds the snapshot configuration and is intended for internal use.

Examples

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

  server <- function(input, output, session) {
    snap_enable(app = "myapp", version = "2.4.0", exclude = c("^btn_", "^nav_"))
  }
}
```

---

snap_exclude *Exclude or include inputs by pattern*

Description

snap_exclude() adds regular expressions; inputs whose fully namespaced id matches any of them are left out of snapshots and ignored on restore. snap_include() adds patterns that an id must match to be captured at all. Both accumulate over calls within a session and complement the exclude/include arguments of snap_enable() and snap_take().

Usage

```
snap_exclude(patterns, session = shiny::getDefaultReactiveDomain())

snap_include(patterns, session = shiny::getDefaultReactiveDomain())
```

Arguments

patterns	A character vector of regular expressions, matched against fully namespaced input ids (for example "^mod-btn_").
session	The Shiny session. Defaults to the current session.

Details

Ids excluded with shiny's setBookmarkExclude() are always honoured too, and action buttons, password inputs, and file inputs are never captured as input values.

Value

The complete vector of patterns registered so far, invisibly.

Examples

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

  server <- function(input, output, session) {
    snap_exclude(c("^btn_", "^nav_"))
    snap_include(c("^model_", "^prefs_"))
  }
}
```

---

snap_file_input *Restore-state upload and its handler*

Description

snap_file_input() is a fileInput() with the client script attached. snap_file_restore() observes it: when a file is uploaded it reads the snapshot with snap_read(), runs validate and migrate, and calls snap_restore(), reporting problems to the user.

Usage

```
snap_file_input(id, label = "Restore state", accept = c(".json", ".zip"), ...)

snap_file_restore(
  id,
  ...,
  validate = NULL,
```

```
  migrate = NULL,
  on_error = c("notify", "modal", "stop"),
  session = shiny::getDefaultReactiveDomain()
)
```

Arguments

id	The input id. `snap_file_restore()` accepts a vector of ids, so that several upload controls share one definition.
label, accept, ...	Passed on to `shiny::fileInput()`. In `snap_file_restore()`, `...` is passed on to `snap_restore()`.
validate	A function of the snapshot that should `stop()` with a user-facing message when the file is not acceptable. Runs first.
migrate	A function `function(snapshot, from_version)` returning a modified snapshot, for example filling in defaults for values introduced after `from_version` (the app version recorded in the file, or `NULL`). Runs after `validate`, before anything is touched.
on_error	How to report a file that cannot be restored: a `showNotification()` (the default), a `showModal()` dialog, or `stop()`.
session	The Shiny session. Defaults to the current session.

Value

`snap_file_input()` returns a tag. `snap_file_restore()` returns the ids invisibly.

Examples

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

  ui <- fluidPage(
    sliderInput("n", "n", 1, 100, 50),
    snap_download_button("save"),
    snap_file_input("restore")
  )

  server <- function(input, output, session) {
    snap_enable(app = "demo", version = "1.0.0")
    snap_download_handler("save")
    snap_file_restore("restore", validate = function(snap) {
      if (is.null(snap$inputs$n)) stop("This file does not contain `n`.")
    })
  }

  shinyApp(ui, server)
}
```

snap_inputs *Access the parts of a snapshot*

Description

`snap_inputs()` returns the captured input values (a named list keyed by fully namespaced input ids), `snap_values()` the server-side values, and `snap_meta()` the user-supplied metadata. See shinysnap-package for the layout of the object.

Usage

```
snap_inputs(x)

snap_values(x)

snap_meta(x)
```

Arguments

x	A snapshot object.

Value

A named list, possibly empty.

Examples

```
snap <- snap_unserialize('{
  "format": 1,
  "inputs": {"n": 100, "rate": 0.025},
  "values": {"prefs": {"digits": 3, "scientific": false}},
  "meta": {"note": "baseline scenario"}
}')
snap_inputs(snap)
snap_values(snap)
snap_meta(snap)
```

snap_is_restoring *Is a restore in flight?*

Description

`TRUE` from the moment `snap_restore()` is called until the browser reports that the restore has settled (or timed out), `FALSE` otherwise. Inside a reactive context the result is reactive, so observers and outputs can react to the end of a restore; outside one it is a plain value.

Usage

```
snap_is_restoring(session = shiny::getDefaultReactiveDomain())
```

Arguments

session	The Shiny session. Defaults to the current session.

Details

Use it to keep expensive observers from running on every intermediate value during a restore, and to run something once the state is complete.

Value

A logical scalar.

Examples

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

  server <- function(input, output, session) {
    observeEvent(input$n, {
      if (snap_is_restoring()) {
        return()
      }
      message("user changed n to ", input$n)
    })
  }
}
```

---

snap_on_restore *Register hooks that run around a restore*

Description

`snap_on_restore()` registers a function that `snap_restore()` calls after the tracked values have been written back and before the input values are sent to the browser. `snap_on_restored()` registers a function that runs once the browser reports that the restore has settled. Both receive a `state` list with `inputs` (the named list of input values being restored), `values` (the `values` section of the file), `snapshot` (the whole snapshot), and `txn` (the transaction id); the `restored` hook also receives the restore report.

Usage

```
snap_on_restore(fn, session = shiny::getDefaultReactiveDomain())

snap_on_restored(fn, session = shiny::getDefaultReactiveDomain())
```

Arguments

fn	A function taking state (and, for snap_on_restored(), report).
session	The Shiny session. Defaults to the current session.

Details

Inside a module, the hooks see only the module's inputs and values, with the namespace prefix removed, and only the module's rows of the report.

Errors raised by a hook abort the restore and reject its promise.

Value

A function with no arguments that removes the hook when called.

Examples

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

  server <- function(input, output, session) {
    snap_on_restore(function(state) {
      message("restoring ", length(state$inputs), " inputs")
    })
    snap_on_restored(function(state, report) {
      print(report)
    })
  }
}
```

---

snap_on_save *Register a hook that runs when a snapshot is taken*

Description

snap_on_save() registers a function that snap_take() calls after the inputs and tracked values have been collected. It receives a state object: state$inputs is the named list of captured input values and state$values is an environment. Anything the hook assigns into state$values is stored in the snapshot's values section, so several hooks (and several modules) can contribute without overwriting each other, just like shiny's onBookmark().

Usage

```
snap_on_save(fn, session = shiny::getDefaultReactiveDomain())
```

Arguments

fn	A function taking one argument, state.
session	The Shiny session. Defaults to the current session.

Details

Inside a module, the hook sees only the module's inputs (with the namespace prefix removed) and its values are stored under namespaced names automatically.

Value

A function with no arguments that removes the hook when called.

Examples

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

  server <- function(input, output, session) {
    snap_on_save(function(state) {
      state$values$fitted_at <- format(Sys.time())
    })
  }
}
```

---

snap_restore *Restore a snapshot into the running app*

Description

Writes the snapshot's tracked values back into the registered `reactiveValues`, runs the restore hooks, and sends the input values to the browser, where the client script applies each one as soon as its input is on the page. Inputs inside dynamic UI that appears (or re-renders) during the restore receive their values without any timing configuration; while the restore is in flight, shiny's `restoreInput()` mechanism is primed so that dynamic UI is built with the restored values directly.

Usage

```
snap_restore(
  x,
  session = shiny::getDefaultReactiveDomain(),
  ...,
  inputs = TRUE,
  values = TRUE,
  include = NULL,
  exclude = NULL,
  validate = NULL,
  migrate = NULL,
  check_app = TRUE,
  unknown = c("warn", "skip", "error"),
  timeout = 10,
  settle = 0.3,
  use_restore_context = NULL,
```

```
  on_done = NULL
)
```

Arguments

x	A snapshot object, a file path, or JSON text.
session	The Shiny session. Defaults to the current session.
...	Not used; arguments after session must be named.
inputs, values	Restore the inputs / the tracked values? Set one to FALSE to restore only the other.
include, exclude	Regular expressions matched against fully namespaced ids, in addition to those configured with snap_enable().
validate	A function of the snapshot that should stop() with a user-facing message when the file is not acceptable. Runs first.
migrate	A function function(snapshot, from_version) returning a modified snapshot, for example filling in defaults for values introduced after from_version (the app version recorded in the file, or NULL). Runs after validate, before anything is touched.
check_app	Refuse files whose app name differs from the one configured with snap_enable() (only when both are known).
unknown	What to do when inputs never appeared or failed: "warn" (one consolidated warning, the default), "skip" (nothing), or "error" (reject the promise).
timeout	Seconds after which the browser stops waiting for inputs that have not appeared.
settle	Seconds of quiet (no busy state, no new UI) after which the browser considers the restore complete.
use_restore_context	Prime shiny's restoreInput() mechanism during the restore. Defaults to the value configured with snap_enable().
on_done	A function of the report, called when the restore settles; a convenience for code that does not want to work with promises.

Details

The function never blocks. It returns a handle containing a promise that resolves to a *restore report* once the browser reports that the page has settled (no activity for settle seconds) or timeout seconds have passed. The report is a data frame with one row per input and the columns id, status, binding, and detail, plus the attributes txn, elapsed (seconds), settled, and timed_out. Statuses:

- applied: sent to an input that was on the page.
- constructed: the input appeared during the restore already carrying the value, thanks to restoreInput().
- reapplied: the input re-rendered during the restore and received the value again.
- missing: the input never appeared before the restore settled.

- `failed`: the input's binding raised an error.
- `mismatched`: applied, but the input reports a different value afterwards (for example a select whose choices do not contain it).
- `skipped`: excluded, or its restorer chose not to restore it.

Preparation errors (an unreadable file, a failing `validate` or `migrate` hook, a file from a different app) are raised immediately. Problems during the transaction reject the promise: a second `snap_restore()` cancels the first (condition class `shinysnap_cancelled`), a failing hook, and, with `unknown = "error"`, inputs that never appeared or failed.

Value

A handle of class `shinysnap_restore`, invisibly: a list with the transaction id in `txn` and a `promises::promise()` that resolves to the restore report in `promise`. The handle is deliberately not a promise itself: shiny waits for a promise returned from an observer before it flushes, and the report can only arrive after the browser has finished applying the restore, so returning the promise from an observer would stall the restore. Use `on_done`, `snap_on_restored()`, or `promises::then(handle$promise, ...)` to work with the report.

Examples

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

  server <- function(input, output, session) {
    observeEvent(input$restore_from_text, {
      snap_restore(input$json, on_done = function(report) print(report))
    })
  }
}
```

`snap_restorer` *Register how an input is restored*

Description

A *restorer* turns the value stored in a snapshot into the message that the input's client-side binding understands, that is, what the matching `update*Input()` function would send. shinysnap ships restorers for the inputs of shiny, bslib, and shinyMatrix (see `snap_restorers()`); the default for everything else is `list(value = value)`. Register your own for an input id or for a binding name (for example `"shinyWidgets.pickerInput"`).

Usage

```
snap_restorer(x, fn, session = NULL)

snap_restorers(session = shiny::getDefaultReactiveDomain())
```

Arguments

x	An input id or a binding name, as reported in the snapshot's bindings section.
fn	The restorer function, or NULL to remove a registration.
session	A Shiny session to register the restorer for that session only, or NULL (the default) to register it globally.

Details

fn is called as fn(id, value, binding, session) and must return the message as a list, or NULL to skip the input (reported as skipped). It must not call update*() functions itself: those go through session$sendInputMessage(), which silently drops messages for inputs that are not on the page yet. To find the right payload, read the update*() function's source and keep the part that carries the value.

Optionally, the returned list may carry an attribute expect holding the value the binding's getValue() is expected to return after the message was applied; the client uses it to report mismatched when the widget shows something else. By default the message's value is used.

Resolution order when restoring an input: a session restorer for the id, a session restorer for its binding, a global restorer for the id or the binding, the built-in restorer for the binding, the default.

Value

snap_restorer() returns fn invisibly. snap_restorers() returns a data frame with the columns name and scope ("builtin", "global", or "session").

Examples

```
# A restorer for a hypothetical widget that expects a selected field:
snap_restorer("mypkg.myInput", function(id, value, binding, session) {
  list(selected = value)
})
snap_restorers()
snap_restorer("mypkg.myInput", NULL)
```

---

snap_serialize *Convert a snapshot to and from JSON text*

Description

snap_serialize() writes a snapshot as JSON text in the canonical shinysnap format; snap_unserialize() reads it back. These are the in-memory counterparts of snap_write() and snap_read().

Usage

```
snap_serialize(
  x,
  format = "json",
  pretty = TRUE,
  unsupported = c("error", "rds"),
  verbose = FALSE
)

snap_unserialize(
  text,
  format = "json",
  trust = FALSE,
  unknown_types = c("error", "keep")
)
```

Arguments

x	A snapshot object, as returned by snap_take() or snap_unserialize(), or a list with (some of) a snapshot's fields, for example list(inputs = list(n = 5)).
format	The text format. Only "json" is available.
pretty	Pretty-print with two-space indentation (the default) or emit compact JSON on one line.
unsupported	What to do with values the JSON format cannot describe: "error" (the default) or "rds" to embed them as serialized R objects.
verbose	Print a message about what was converted or dropped.
text	JSON text: a single string or a character vector of lines.
trust	Decode embedded serialized R objects ("$type": "rds")? Only set this to TRUE for files from a source you trust.
unknown_types	What to do with a "$type" the reader does not know: "error" (the default) or "keep" to keep the raw parsed value.

Details

The JSON format is designed to be read and edited by people: doubles are written with the fewest digits that read back to the same value, integers as plain digit runs, and everything JSON cannot express directly (the type of an empty vector, NA, Inf, names, dates, matrices, factors, data frames) as a small object with a "$type" key.

Values that the format cannot describe (environments, functions, S4 and R6 objects, unknown classes) are an error by default. With unsupported = "rds" they are embedded as base64 serialized R objects instead; because unserializing arbitrary data is unsafe, reading them back requires trust = TRUE, otherwise they decode to NULL with a warning.

Value

`snap_serialize()` returns a single string. `snap_unserialize()` returns a snapshot object of class `shinysnap`.

Examples

```
text <- '{
  "format": 1,
  "app": {"name": "myapp", "version": "2.4.1"},
  "inputs": {
    "dates": {"$type": "Date", "value": ["2024-01-01", "2024-03-01"]},
    "method": "b",
    "n": 100,
    "rate": 0.025,
    "weights": [0.5, 0.75]
  }
}'
snap <- snap_unserialize(text)
snap
str(snap_inputs(snap))
cat(snap_serialize(snap))
```

---

snap_take *Take a snapshot of the running app*

Description

Captures the current input values and the registered server-side values of the session into a snapshot object, ready for `snap_write()` or `snap_serialize()`. Nothing is written to disk.

Usage

```
snap_take(
  session = shiny::getDefaultReactiveDomain(),
  ...,
  include = NULL,
  exclude = NULL,
  live_only = NULL,
  values = TRUE,
  scope = c("root", "module"),
  meta = list()
)
```

Arguments

`session`	The Shiny session. Defaults to the current session.
`...`	Not used; arguments after `session` must be named.

include, exclude
: Regular expressions matched against fully namespaced ids, in addition to those configured with snap_enable().

live_only
: Keep only inputs currently on the page. Defaults to the value configured with snap_enable() (TRUE).

values
: Capture tracked values and run the save hooks? FALSE captures inputs only.

scope
: "root" (the default) captures the whole app with full ids, even when called inside a module; "module" keeps only ids under the calling module's namespace (still as full ids).

meta
: A named list of free-form metadata stored in the snapshot.

Details

What is captured:

- Inputs, by fully namespaced id, exactly as input$id returns them. With live_only, only inputs that are currently on the page are kept, so values of inputs whose dynamic UI has been removed are not carried along. Action buttons, password inputs, and values that shiny's own serializers mark as unserializable are never captured; ids excluded with setBookmarkExclude(), snap_exclude(), or the exclude patterns are dropped. File inputs are moved to the snapshot's attachments.
- The name of the client-side input binding of each captured input, in bindings.
- Values: the fields of every reactiveValues registered with snap_track(), then whatever the snap_on_save() hooks add.

The function isolates every read, so it never creates reactive dependencies. Inputs with a rate policy (text inputs debounce, sliders throttle) may lag the browser by a few hundred milliseconds; a snapshot taken from a download handler runs after the click has reached the server, which in practice is later than that.

Value

A snapshot object of class shinysnap.

Examples

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

  server <- function(input, output, session) {
    observeEvent(input$show, {
      print(snap_take())
    })
  }
}
```

snap_track *Track server-side values*

Description

Registers a `reactiveValues` object so that `snap_take()` captures its fields (all of them, or the ones named in `fields`) under `name` in the snapshot's `values` section, and so that a restore can write them back.

Usage

```
snap_track(
  values,
  fields = NULL,
  name = NULL,
  session = shiny::getDefaultReactiveDomain()
)
```

Arguments

`values`	A `reactiveValues` object.
`fields`	Names of the fields to capture, or `NULL` for all fields.
`name`	The name to store the values under. Defaults to the name of the variable passed as `values`.
`session`	The Shiny session. Defaults to the current session.

Details

Inside a module, `name` is namespaced automatically.

Value

The (namespaced) name the values are stored under, invisibly.

Examples

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

  server <- function(input, output, session) {
    prefs <- reactiveValues(digits = 3L, scientific = FALSE)
    snap_track(prefs)
  }
}
```

snap_write *Write a snapshot to a file and read it back*

Description

`snap_write()` saves a snapshot; `snap_read()` loads one. The canonical format is JSON (see `snap_serialize()`): plain text, readable, diffable, and safe to open. The `"zip"` format is a *bundle*: a zip archive holding the same JSON as `manifest.json` plus the files of any `fileInput()` uploads (under `attachments/`) and, with `unsupported = "rds"`, opaque R objects (under `objects/`); it needs the zip package. The `"rds"` format stores the R object with `saveRDS()`; it is neither readable nor safe across versions, and reading it requires `trust = TRUE` because unserializing a file runs arbitrary code paths.

Usage

```
snap_write(
  x,
  path,
  format = c("auto", "json", "zip", "rds"),
  pretty = TRUE,
  ...
)

snap_read(
  path,
  format = c("auto", "json", "zip", "rds"),
  trust = FALSE,
  unknown_types = c("error", "keep"),
  ...
)
```

Arguments

`x`	A snapshot object, or a list with (some of) a snapshot's fields (see `snap_serialize()`).
`path`	The file path.
`format`	`"auto"` picks the format from the extension (`.json`, `.zip`, or `.rds`); otherwise the format to use regardless of the extension.
`pretty`	Pretty-print JSON (the default) or write one compact line.
`...`	Passed on to `snap_serialize()` (`unsupported`, `verbose`).
`trust`	Decode embedded serialized R objects and allow the `"rds"` format? Only set this to `TRUE` for files from a source you trust.
`unknown_types`	What to do with a `"$type"` the reader does not know: `"error"` (the default) or `"keep"` to keep the raw parsed value.

Details

Reading a bundle checks the archive for path traversal and caps its uncompressed size at `getOption("shinysnap.max_bundl 100 * 1024^2)` bytes, then extracts it into a fresh temporary directory; `snap_attachment()` returns the local paths of the extracted uploads and bundled objects are decoded only with `trust = TRUE`.

Value

`snap_write()` returns `path` invisibly; `snap_read()` returns a snapshot object.

Examples

```
snap <- snap_unserialize('{"format": 1, "inputs": {"n": 100, "rate": 0.025}}')
path <- tempfile(fileext = ".json")
snap_write(snap, path)
cat(readLines(path), sep = "\n")
identical(snap_inputs(snap_read(path)), snap_inputs(snap))
```

Index