Package 'shinytest2'

November 10, 2026

Title Testing for Shiny Applications

Version 0.6.0

Description Automated unit testing of Shiny applications through a headless 'Chromium' browser.

License MIT + file LICENSE

URL https://rstudio.github.io/shinytest2/, https://github.com/rstudio/shinytest2

BugReports https://github.com/rstudio/shinytest2/issues

Depends R (>= 4.1), testthat (>= 3.3.1)

Imports callr, checkmate (>= 2.0.0), chromote (>= 0.5.0), cli, fs, globals (>= 0.14.0), httr2 (>= 1.0.0), jsonlite, lifecycle (>= 1.0.3), pingr, pkgload, R6 (>= 2.4.0), rlang (>= 1.0.0), rmarkdown, shiny, withr

Suggests box, deSolve, diffobj, ggplot2, golem, httpuv, knitr, plotly, png, rhino, rstudioapi, shinytest (>= 1.5.1), shinyvalidate (>= 0.1.2), shinyWidgets, showimage, spelling, usethis, vdiffr (>= 1.0.0)

LinkingTo cpp11

VignetteBuilder knitr

Config/Needs/check rstudio/shiny, bslib

Config/Needs/shinytest2-testing decor

Config/Needs/website pkgdown, tidyverse/tidytemplate

Config/roxygen2/version 8.1.1

Config/testthat/edition 3

Config/usethis/last-upkeep 2026-11-09

Encoding UTF-8

Language en-US

Collate 'R6-helper.R' 'app-driver-chromote.R' 'app-driver-dir.R'
'app-driver-expect-download.R' 'app-driver-expect-js.R'
'app-driver-expect-screenshot.R'
'app-driver-expect-unique-names.R' 'app-driver-expect-values.R'
'app-driver-get-log.R' 'app-driver-initialize.R'
'app-driver-log-message.R' 'app-driver-message.R'
'app-driver-node.R' 'app-driver-set-inputs.R'
'app-driver-start.R' 'app-driver-stop.R' 'app-driver-timeout.R'
'app-driver-upload-file.R' 'app-driver-variant.R'
'app-driver-wait.R' 'app-driver-window.R' 'app-driver.R'
'chromote-methods.R' 'compare-screenshot-threshold.R' 'cpp11.R'
'expect-snapshot.R' 'expr-recurse.R' 'httr2.R' 'migrate.R'
'missing-value.R' 'pkg.R' 'utils.R' 'platform.R'
'record-test-unique-name.R' 'record-test.R' 'rstudio.R'
'save-app.R' 'shiny-browser.R' 'shinytest2-logs.R'
'shinytest2-package.R' 'test-app.R' 'use.R'

NeedsCompilation yes

Author Barret Schloerke [cre, aut] (ORCID:
<https://orcid.org/0000-0001-9986-114X>),
Posit Software, PBC [cph, fnd] (ROR: <https://ror.org/03wc8by49>),
Winston Chang [ctb] (Original author to rstudio/shinytest),
Gábor Csárdi [ctb] (Original author to rstudio/shinytest),
Hadley Wickham [ctb] (Original author to rstudio/shinytest),
Mango Solutions [cph, ccp] (Original author to rstudio/shinytest)

Maintainer Barret Schloerke <barret@posit.co>

Repository CRAN

Date/Publication 2026-11-10 08:10:08 UTC

Contents

AppDriver 2
compare_screenshot_threshold 53
local_app_support 56
migrate_from_shinytest 58
platform_variant 59
record_test 60
test_app 61
use_shinytest2 64

Index **66**

`AppDriver` *Drive a Shiny application*

Description

This class starts a Shiny app in a new R session, along with **chromote**'s headless browser that can be used to simulate user actions. This provides a full simulation of a Shiny app so that you can test user interactions with a live app.

Methods described below are ordered by perceived popularity. *Expect* methods are grouped next to their corresponding *get* methods.

Vignettes

Please see Testing in depth for more details about the different expectation methods.

Please see Robust testing for more details about the cost / benefits for each expectation method.

Test mode

To have your `AppDriver` retrieve values from your Shiny app, be sure to set `shiny::runApp(test.mode = TRUE)` when running your Shiny app.

If you are deploying your Shiny app where you do not have control over the call to `shiny::runApp()`, you can set `options(shiny.testmode = TRUE)` in a `.Rprofile` file within your Shiny app directory.

Start-up failure

If the app throws an error during initialization, the `AppDriver` will be stored in `rlang::last_error()$app`. This allows for the "failure to initialize" to be signaled while also allowing for the app to be retrieved after any initialization error has been thrown.

Exporting reactive values

Reactive values from within your Shiny application can be exported using the method: `shiny::exportTestValues()`. This underutilized method exposes internal values of your app without needing to create a corresponding input value or output value.

For example:

```
library(shiny)
shiny_app <- shinyApp(
  fluidPage(
    h1("Pythagorean theorem"),
    numericInput("A", "A", 3),
    numericInput("B", "B", 4),
    verbatimTextOutput("C"),
  ),
  function(input, output) {
    a_squared <- reactive({ req(input$A); input$A * input$A })
    b_squared <- reactive({ req(input$B); input$B * input$B })
    c_squared <- reactive({ a_squared() + b_squared() })
    c_value <- reactive({ sqrt(c_squared()) })
    output$C <- renderText({ c_value() })
```

```
    exportTestValues(
      a_squared = { a_squared() },
      b_squared = { b_squared() },
      c_squared = { c_squared() }
    )
  }
)

app <- AppDriver$new(shiny_app)

init_vals <- app$get_values()
str(init_vals)
#> List of 3
#> $ input :List of 2
#> ..$ A: int 3
#> ..$ B: int 4
#> $ output:List of 1
#> ..$ C: chr "5"
#> $ export:List of 3
#> ..$ a_squared: int 9
#> ..$ b_squared: int 16
#> ..$ c_squared: int 25
```

These exported test values are only exposed when `shiny::runApp(test.mode = TRUE)` is set. **shinytest2** sets this variable when running Shiny based app or document.

testthat wrappers

The two main expectation methods: `$expect_values()` and `$expect_screenshot()` eventually wrap `testthat::expect_snapshot_file()`.

Their underlying logic is similar to:

```
skip_on_cran() # AppDriver

## Expect values
tmpfile <- tempfile(fileext = ".json")
jsonlite::write_json(app$get_values(), tmpfile)
expect_snapshot_file(
  tmpfile,
  variant = app$get_variant(),
  compare = testthat::compare_file_text
)


## Expect screenshot
tmpfile <- tempfile(fileext = ".png")
app$get_screenshot(tmpfile)
```

```
expect_snapshot_file(
  tmpfile,
  variant = app$get_variant(),
  compare = testthat::compare_file_binary
)
```

To update the snapshot values, you will need to run a variation of `testthat::snapshot_review()`.

Methods

Public methods:

- `AppDriver$new()`
- `AppDriver$view()`
- `AppDriver$click()`
- `AppDriver$set_inputs()`
- `AppDriver$upload_file()`
- `AppDriver$expect_values()`
- `AppDriver$get_value()`
- `AppDriver$get_values()`
- `AppDriver$expect_download()`
- `AppDriver$get_download()`
- `AppDriver$expect_text()`
- `AppDriver$get_text()`
- `AppDriver$expect_html()`
- `AppDriver$get_html()`
- `AppDriver$expect_js()`
- `AppDriver$get_js()`
- `AppDriver$run_js()`
- `AppDriver$expect_screenshot()`
- `AppDriver$get_screenshot()`
- `AppDriver$wait_for_idle()`
- `AppDriver$wait_for_value()`
- `AppDriver$wait_for_js()`
- `AppDriver$expect_unique_names()`
- `AppDriver$get_dir()`
- `AppDriver$get_url()`
- `AppDriver$get_window_size()`
- `AppDriver$set_window_size()`
- `AppDriver$get_chromote_session()`
- `AppDriver$get_process()`
- `AppDriver$get_variant()`
- `AppDriver$get_logs()`
- `AppDriver$log_message()`

- AppDriver$stop()

`AppDriver$new()`: Initialize an `AppDriver` object

Usage:

```
AppDriver$new(
  app_dir = testthat::test_path("../../"),
  ...,
  name = NULL,
  variant = missing_arg(),
  seed = NULL,
  load_timeout = missing_arg(),
  timeout = missing_arg(),
  wait = TRUE,
  screenshot_args = missing_arg(),
  expect_values_screenshot_args = TRUE,
  check_names = TRUE,
  view = missing_arg(),
  height = NULL,
  width = NULL,
  clean_logs = TRUE,
  shiny_args = list(),
  render_args = NULL,
  options = list()
)
```

Arguments:

`app_dir` This value can be many different things:

- A directory containing your Shiny application or a run-time Shiny R Markdown document.
- A URL pointing to your shiny application. (Don't forget to set `testmode = TRUE` when running your application!)
- A Shiny application object which inherits from `"shiny.appobj"`.
- A function that runs your Shiny application or Shiny R Markdown or returns a Shiny app object. If being used within a package, be sure to call `library(<pkg>)` first within the provided function as the function will be executed within the background R process. When `library*()` is called, `pkgload::load_all()` will be automatically executed to load the package's source code.

By default, `app_dir` is set to `test_path("../../")` to work in both interactive and testing usage.

If a file path is not provided to `app_dir`, snapshots will be saved as if the root of the Shiny application was the current directory.

`...` Must be empty. Allows for parameter expansion.

`name` Prefix value to use when saving testthat snapshot files. Ex: `NAME-001.json`. Name **must** be unique when saving multiple snapshots from within the same testing file. Otherwise, two different `AppDriver` objects will be referencing the same files.

`variant` If not-`NULL`, results will be saved in `_snaps/{variant}/{test.md}`, so `variant` must be a single string of alphanumeric characters suitable for use as a directory name.

You can variants to deal with cases where the snapshot output varies and you want to capture and test the variations. Common use cases include variations for operating system, R version, or version of key dependency. For example usage, see `platform_variant()`.

`seed` An optional random seed to use before starting the application. For apps that use R's random number generator, this can make their behavior repeatable.

`load_timeout` How long to wait for the app to load, in ms. This includes the time to start R. Defaults to 15s.

If `load_timeout` is missing, the first numeric value found will be used:

- R option `options(shinytest2.load_timeout=)`
- System environment variable `SHINYTEST2_LOAD_TIMEOUT`
- `15 * 1000` (15 seconds)

`timeout` Default number of milliseconds for any `timeout` or `timeout_` parameter in the `AppDriver` class. Defaults to 4s.

If `timeout` is missing, the first numeric value found will be used:

- R option `options(shinytest2.timeout=)`
- System environment variable `SHINYTEST2_TIMEOUT`
- `4 * 1000` (4 seconds)

`wait` If `TRUE`, `$wait_for_idle(duration = 200, timeout = load_timeout)` will be called once the app has connected a new session, blocking until the Shiny app is idle for 200ms.

`screenshot_args` Default set of arguments to pass in to `chromote::ChromoteSession`'s `$get_screenshot()` method when taking screenshots within `$expect_screenshot()`. To disable screenshots by default, set to `FALSE`.

`expect_values_screenshot_args` The value for `screenshot_args` when producing a debug screenshot for `$expect_values()`. To disable debug screenshots by default, set to `FALSE`.

`check_names` Check if widget names are unique once the application initially loads? If duplicate names are found on initialization, a warning will be displayed.

`view` Opens the `chromote::ChromoteSession` in an interactive browser tab before attempting to navigate to the Shiny app.

`height, width` Window size to use when opening the `chromote::ChromoteSession`. Both `height` and `width` values must be non-null values to be used.

`clean_logs` Whether to remove the `stdout` and `stderr` Shiny app logs when the `AppDriver` object is garbage collected.

`shiny_args` A list of options to pass to `shiny::runApp()`. Ex: `list(port = 8080)`.

`render_args` Passed to `rmarkdown::run(render_args=)` for interactive `.Rmd`s. Ex: `list(quiet = TRUE)`

`options` A list of `base::options()` to set in the Shiny application's child R process. See `shiny::shinyOptions()` for inspiration. If `shiny.trace = TRUE`, then all WebSocket traffic will be captured by `chromote` and time-stamped for logging purposes.

Returns: An object with class `AppDriver` and the many methods described in this documentation.

Examples:

```
# Create an AppDriver from the Shiny app in the current directory
app <- AppDriver()

# Create an AppDriver object from a different Shiny app directory
```

```
example_app <- system.file("examples/01_hello", package = "shiny")
app <- AppDriver(example_app)

# Expect consistent inital values
app$expect_values()
```

`AppDriver$view()`: View the Shiny application

Calls `$view()` on the `chromote::ChromoteSession` object to *view* your Shiny application in a Chrome browser.

This method is very helpful for debugging while writing your tests.

Usage:

```
AppDriver$view()
```

Examples:

```
# Open app in Chrome
app$view()
```

`AppDriver$click()`: Click an element

Find a Shiny input/output value or DOM CSS selector and click it using the DOM method `TAG.click()`.

This method can be used to click input buttons and other elements that need to simulate a click action.

Usage:

```
AppDriver$click(
  input = missing_arg(),
  output = missing_arg(),
  selector = missing_arg(),
  ...
)
```

Arguments:

`input`, `output`, `selector` A name of an Shiny `input`/`output` value or a DOM CSS selector. Only one of these may be used.

`...` If `input` is used, all extra arguments are passed to `$set_inputs(!!input := "click", ...)`. This means that the `AppDriver` will wait until an output has been updated within the specified `timeout_`. When clicking any other content, `...` must be empty.

Examples:

```
app_path <- system.file("examples/07_widgets", package = "shiny")
app <- AppDriver$new(app_path)

tmpfile <- write.csv(cars, "cars.csv")
app$upload_file(file1 = tmpfile)
cat(app$get_text("#view"))
app$set_inputs(dataset = "cars", obs = 6)
app$click("update")
cat(app$get_text("#view"))
```

`AppDriver$set_inputs()`: Set input values

Set Shiny inputs by sending the value to the Chrome browser and programmatically updating the values. Given `wait_ = TRUE`, the method will not return until an output value has been updated.

Usage:

```
AppDriver$set_inputs(
  ...,
  wait_ = TRUE,
  timeout_ = missing_arg(),
  allow_no_input_binding_ = FALSE,
  priority_ = c("input", "event")
)
```

Arguments:

`...` Name-value pairs, `component_name_1 = value_1, component_name_2 = value_2` etc. Input with name `component_name_1` will be assigned value `value_1`.

`wait_` Wait until all reactive updates have completed?

`timeout_` Amount of time to wait before giving up (milliseconds). Defaults to the resolved `timeout` value during the `AppDriver` initialization.

`allow_no_input_binding_` When setting the value of an input, allow it to set the value of an input even if that input does not have an input binding. This is useful to replicate behavior like hovering over a **plotly** plot.

`priority_` Sets the event priority. For expert use only: see https://shiny.rstudio.com/articles/communicating-with-js.html#values-vs-events for details.

Examples:

```
app_path <- system.file("examples/07_widgets", package = "shiny")
app <- AppDriver$new(app_path)

cat(app$get_text("#view"))
app$set_inputs(dataset = "cars", obs = 6)
app$click("update")
cat(app$get_text("#view"))
```

`AppDriver$upload_file()`: Upload a file

Uploads a file to the specified file input.

Usage:

```
AppDriver$upload_file(..., wait_ = TRUE, timeout_ = missing_arg())
```

Arguments:

`...` Name-path pair, e.g. `component_name = file_path`. The file located at `file_path` will be uploaded to file input with name `component_name`.

`wait_` Wait until all reactive updates have completed?

`timeout_` Amount of time to wait before giving up (milliseconds). Defaults to the resolved `timeout` value during the `AppDriver` initialization.

Examples:

```
app_path <- system.file("examples/09_upload", package = "shiny")
app <- AppDriver$new(app_path)
```

```
# Save example file
tmpfile <- tempfile(fileext = ".csv")
write.csv(cars, tmpfile, row.names = FALSE)

# Upload file to input named: file1
app$upload_file(file1 = tmpfile)
```

`AppDriver$expect_values()`: Expect `input`, `output`, and `export` values

A JSON snapshot is saved of given the results from the underlying call to `$get_values()`.

Note, values that contain environments or other values that will have trouble serializing may not work well. Instead, these objects should be manually inspected and have their components tested individually.

Please see Robust testing for more details.

Usage:

```
AppDriver$expect_values(
  ...,
  input = missing_arg(),
  output = missing_arg(),
  export = missing_arg(),
  screenshot_args = missing_arg(),
  name = NULL,
  transform = NULL,
  cran = deprecated()
)
```

Arguments:

`...` Must be empty. Allows for parameter expansion.

`input`, `output`, `export` Depending on which parameters are supplied, different return values can occur: * If `input`, `output`, and `export` are all missing, then all values are included in the snapshot. * If at least one `input`, `output`, or `export` is specified, then only the requested values are included in the snapshot.
The values supplied to each variable can be: * A character vector of specific names to only include in the snapshot. * `TRUE` to request that all values of that type are included in the snapshot. * Anything else (e.g. `NULL` or `FALSE`) will result in the parameter being ignored.

`screenshot_args` This value is passed along to `$expect_screenshot()` where the resulting snapshot expectation is ignored. If missing, the default value will be `$new(expect_values_screenshot_args=)`. The final value can either be:

- `TRUE`: A screenshot of the browser's scrollable area will be taken with no delay
- `FALSE`: No screenshot will be taken
- A named list of arguments. These arguments are passed directly to `chromote::ChromoteSession`'s `$get_screenshot()` method. The `selector` and `delay` will default to `"html"` and `0` respectively.

`name` The file name to be used for the snapshot. The file extension will be overwritten to `.json`. By default, the `name` supplied to `app` on initialization with a counter will be used (e.g. `"NAME-001.json"`).

`transform` Optionally, a function to scrub sensitive or stochastic text from the output. Should take a character vector of lines as input and return a modified character vector as output.

`cran` Deprecated. With `AppDriver` never testing on CRAN, this parameter no longer has any effect.

Returns: The result of the snapshot expectation

Examples:

```
library(shiny)
shiny_app <- shinyApp(
  fluidPage(
    h1("Pythagorean theorem"),
    numericInput("A", "A", 3),
    numericInput("B", "B", 4),
    verbatimTextOutput("C"),
  ),
  function(input, output) {
    a_squared <- reactive({ req(input$A); input$A * input$A })
    b_squared <- reactive({ req(input$B); input$B * input$B })
    c_squared <- reactive({ a_squared() + b_squared() })
    c_value <- reactive({ sqrt(c_squared()) })
    output$C <- renderText({ c_value() })

    exportTestValues(
      a_squared = { a_squared() },
      b_squared = { b_squared() },
      c_squared = { c_squared() }
    )
  }
)

app <- AppDriver$new(shiny_app)

# Snapshot all known values
app$expect_values()

# Snapshot only `export` values
app$expect_values(export = TRUE)

# Snapshot values `"A"` from `input` and `"C"` from `output`
app$expect_values(input = "A", output = "C")
```

`AppDriver$get_value()`**:** Get a single `input`, `output`, or `export` value

This is a helper function around `$get_values()` to retrieve a single `input`, `output`, or `export` value. Only a single `input`, `output`, or `export` value can be used.

Note, values that contain environments or other values that will have trouble serializing to RDS may not work well.

Usage:

```
AppDriver$get_value(
  ...,
  input = missing_arg(),
```

```
  output = missing_arg(),
  export = missing_arg(),
  hash_images = FALSE
)
```

Arguments:

`...` Must be empty. Allows for parameter expansion.

`input, output, export` One of these variable should contain a single string value. If more than one value is specified or no values are specified, an error will be thrown.

`hash_images` If `TRUE`, images will be hashed before being returned. Otherwise, all images will return their full data64 encoded value.

Returns: The requested `input`, `output`, or `export` value.

Examples:

```
app_path <- system.file("examples/04_mpg", package = "shiny")
app <- AppDriver$new(app_path)

# Retrieve a single value
app$get_value(output = "caption")
#> [1] "mpg ~ cyl"
# Equivalent code using `$get_values()`
app$get_values(output = "caption")$output$caption
#> [1] "mpg ~ cyl"
```

`AppDriver$get_values()`: Get `input`, `output`, and `export` values

Retrieves a list of all known `input`, `output`, or `export` values. This method is a core method when inspecting your Shiny app.

Note, values that contain environments or other values that will have trouble serializing may not work well.

Usage:

```
AppDriver$get_values(
  ...,
  input = missing_arg(),
  output = missing_arg(),
  export = missing_arg(),
  hash_images = FALSE
)
```

Arguments:

`...` Must be empty. Allows for parameter expansion.

`input, output, export` Depending on which parameters are supplied, different return values can occur: * If `input`, `output`, and `export` are all missing, then all values are included in the snapshot. * If at least one `input`, `output`, or `export` is specified, then only the requested values are included in the snapshot.

The values supplied to each variable can be: * A character vector of specific names to only include in the snapshot. * `TRUE` to request that all values of that type are included in the snapshot. * Anything else (e.g. `NULL` or `FALSE`) will result in the parameter being ignored.

`hash_images` If `TRUE`, images will be hashed before being returned. Otherwise, all images will return their full data64 encoded value.

Returns: A named list of all inputs, outputs, and export values.

Examples:

```
library(shiny)
shiny_app <- shinyApp(
  fluidPage(
    h1("Pythagorean theorem"),
    numericInput("A", "A", 3),
    numericInput("B", "B", 4),
    verbatimTextOutput("C"),
  ),
  function(input, output) {
    a_squared <- reactive({ req(input$A); input$A * input$A })
    b_squared <- reactive({ req(input$B); input$B * input$B })
    c_squared <- reactive({ a_squared() + b_squared() })
    c_value <- reactive({ sqrt(c_squared()) })
    output$C <- renderText({ c_value() })

    exportTestValues(
      a_squared = { a_squared() },
      b_squared = { b_squared() },
      c_squared = { c_squared() }
    )
  }
)

app <- AppDriver$new(shiny_app)

# Show all known values
str(app$get_values())
#> List of 3
#> $ input :List of 2
#> ..$ A: int 3
#> ..$ B: int 4
#> $ output:List of 1
#> ..$ C: chr "5"
#> $ export:List of 3
#> ..$ a_squared: int 9
#> ..$ b_squared: int 16
#> ..$ c_squared: int 25

# Get only `export` values
str(app$get_values(export = TRUE))
#> List of 1
#> $ export:List of 3
#> ..$ a_squared: int 9
#> ..$ b_squared: int 16
#> ..$ c_squared: int 25
```

```
# Get values `"A"` from `input` and `"C"` from `output`
str(app$get_values(input = "A", output = "C"))
#> List of 2
#> $ input :List of 1
#> ..$ A: int 3
#> $ output:List of 1
#> ..$ C: chr "5"
```

`AppDriver$expect_download()`: Expect a downloadable file

Given a `shiny::downloadButton()`/`shiny::downloadLink()` `output` ID, the corresponding file will be downloaded and saved as a snapshot file.

Usage:

```
AppDriver$expect_download(
  output,
  ...,
  compare = NULL,
  name = NULL,
  transform = NULL,
  cran = deprecated()
)
```

Arguments:

`output` output ID of `shiny::downloadButton()`/`shiny::downloadLink()`

`...` Must be empty. Allows for parameter expansion.

`compare` This value is passed through to `testthat::expect_snapshot_file()`. By default it is set to `NULL` which will default to `testthat::compare_file_text` if `name` has extension `.r`, `.R`, `.Rmd`, `.md`, or `.txt`, and otherwise uses `testthat::compare_file_binary`.

`name` File name to save file to (including file name extension). The default, `NULL`, generates an ascending sequence of names: `001.download`, `002.download`, etc.

`transform` Optionally, a function to scrub sensitive or stochastic text from the output. Should take a character vector of lines as input and return a modified character vector as output.

`cran` Deprecated. With `AppDriver` never testing on CRAN, this parameter no longer has any effect.

Examples:

```
app_path <- system.file("examples/10_download", package = "shiny")
app <- AppDriver$new(app_path)

# Save snapshot of rock.csv as 001.download
# Save snapshot value of `rock.csv` to capture default file name
app$expect_download("downloadData", compare = testthat::compare_file_text)
```

`AppDriver$get_download()`: Get downloadable file

Given a `shiny::downloadButton()`/`shiny::downloadLink()` `output` ID, the corresponding file will be downloaded and saved as a file.

Usage:

```
AppDriver$get_download(output, filename = NULL)
```

Arguments:

`output` output ID of `shiny::downloadButton()`/`shiny::downloadLink()`

`filename` File path to save the downloaded file to.

Returns: `$get_download()` will return the final save location of the file. This location can change depending on the value of `filename` and response headers.
Location logic:

- If `filename` is not NULL, `filename` will be returned.
- If a `content-disposition` `filename` is provided, then a temp file containing this `filename` will be returned.
- Otherwise, a temp file ending in `.download` will be returned.

Examples:

```
app_path <- system.file("examples/10_download", package = "shiny")
app <- AppDriver$new(app_path)

# Get rock.csv as a tempfile
app$get_download("downloadData")
#> [1] "/TEMP/PATH/rock.csv"

# Get rock.csv as a "./myfile.csv"
app$get_download("downloadData", filename = "./myfile.csv")
#> [1] "./myfile.csv"
```

`AppDriver$expect_text()`**:** Expect snapshot of UI text
`$expect_text()` will extract the text value of all matching elements via `TAG.textContent` and store them in a snapshot file. This method is more robust to internal package change as only the text values will be maintained. Note, this method will not retrieve any `<input />` value's text content, e.g. text inputs or text areas, as the input values are not stored in the live HTML.

When possible, use `$expect_text()` over `$expect_html()` to allow package authors room to alter their HTML structures. The resulting array of `TAG.textContent` values found will be stored in a snapshot file.

Please see Robust testing for more details.

Usage:

```
AppDriver$expect_text(selector, ..., cran = deprecated())
```

Arguments:

`selector` A DOM CSS selector to be passed into `document.querySelectorAll()`

`...` Must be empty. Allows for parameter expansion.

`cran` Deprecated. With `AppDriver` never testing on CRAN, this parameter no longer has any effect.

Examples:

```
hello_app <- system.file("examples/01_hello", package = "shiny")
app <- AppDriver$new(hello_app)

# Make a snapshot of `"Hello Shiny!"`
app$expect_text("h2")
```

`AppDriver$get_text()`: Get UI text

`$get_text()` will extract the text value of all matching elements via `TAG.textContent`. Note, this method will not retrieve any `<input />` value's text content, e.g. text inputs or text areas, as the input values are not stored in the live HTML.

Usage:

```
AppDriver$get_text(selector)
```

Arguments:

`selector` A DOM CSS selector to be passed into `document.querySelectorAll()`

Returns: A vector of character values

Examples:

```
hello_app <- system.file("examples/01_hello", package = "shiny")
app <- AppDriver$new(hello_app)

app$get_text("h2")
#> [1] "Hello Shiny!"
```

`AppDriver$expect_html()`: Expect snapshot of UI HTML

`$expect_html()` will extract the full DOM structures of each matching element and store them in a snapshot file. This method captures internal DOM structure which may be brittle to changes by external authors or dependencies.

Note, this method will not retrieve any `<input />` value's text content, e.g. text inputs or text areas, as the input values are not stored in the live HTML.

When possible, use `$expect_text()` over `$expect_html()` to allow package authors room to alter their HTML structures. The resulting array of `TAG.textContent` values found will be stored in a snapshot file.

Please see Robust testing for more details.

Usage:

```
AppDriver$expect_html(selector, ..., outer_html = TRUE, cran = deprecated())
```

Arguments:

`selector` A DOM selector to be passed into `document.querySelectorAll()`

`...` Must be empty. Allows for parameter expansion.

`outer_html` If `TRUE` (default), the full DOM structure will be returned (`TAG.outerHTML`). If `FALSE`, the full DOM structure of the child elements will be returned (`TAG.innerHTML`).

`cran` Deprecated. With `AppDriver` never testing on CRAN, this parameter no longer has any effect.

Examples:

```
app_path <- system.file("examples/04_mpg", package = "shiny")
app <- AppDriver$new(app_path)

# Save a snapshot of the `caption` output
app$expect_html("#caption")
```

`AppDriver$get_html()`: Get UI HTML

`$get()` will extract the full DOM structures of each matching element. This method captures internal DOM structure which may be brittle to changes by external authors or dependencies.

Note, this method will not retrieve any <input /> value's text content, e.g. text inputs or text areas, as the input values are not stored in the live HTML.

Please see Robust testing for more details.

Usage:

```
AppDriver$get_html(selector, ..., outer_html = TRUE)
```

Arguments:

`selector` A DOM selector to be passed into `document.querySelectorAll()`

`...` Must be empty. Allows for parameter expansion.

`outer_html` If `TRUE`, the full DOM structure will be returned (`TAG.outerHTML`). If `FALSE`, the full DOM structure of the child elements will be returned (`TAG.innerHTML`).

Examples:

```
app_path <- system.file("examples/03_reactivity", package = "shiny")
app <- AppDriver$new(app_path, check_names = FALSE)

app$set_inputs(caption = "Custom value!")
cat(app$get_html(".shiny-input-container")[1])
#> <div class="form-group shiny-input-container">
#>  <label class="control-label" id="caption-label" for="caption">Caption:</label>
#>  <input id="caption" type="text" class="form-control shiny-bound-input" value="Data Summary">
#> </div>
## ^^ No update to the DOM of `caption`
```

`AppDriver$expect_js()`: Expect snapshot of JavaScript script output

This is a building block function that may be called by other functions. For example, `$expect_text()` and `$expect_html()` are thin wrappers around this function.

Once the `script` has executed, the JSON result will be saved to a snapshot file.

Usage:

```
AppDriver$expect_js(
  script = missing_arg(),
  ...,
  file = missing_arg(),
  timeout = missing_arg(),
  pre_snapshot = NULL,
  cran = deprecated()
)
```

Arguments:

`script` A string containing the JavaScript script to be executed.

`...` Must be empty. Allows for parameter expansion.

`file` A file containing JavaScript code to be read and used as the `script`. Only one of `script` or `file` can be specified.

`timeout` Amount of time to wait before giving up (milliseconds). Defaults to the resolved `timeout` value during the `AppDriver` initialization.

`pre_snapshot` A function to be called on the result of the script before taking the snapshot. `$expect_html()` and `$expect_text()` both use `unlist()`.

`cran` Deprecated. With `AppDriver` never testing on CRAN, this parameter no longer has any effect.

Examples:

```
app_path <- system.file("examples/07_widgets", package = "shiny")
app <- AppDriver$new(app_path)

# Track how many clicks are given to `#update` button
app$run_js("
  window.test_counter = 0;
  $('#update').click(() => window.test_counter++);
")
app$set_inputs(obs = 20)
# Click the update button, incrementing the counter
app$click("update")
# Save a snapshot of number of clicks (1)
app$expect_js("window.test_counter;")
```

`AppDriver$get_js()`**:** Execute JavaScript code in the browser and return the result

This function will block the local R session until the code has finished executing its *tick* in the browser. If a `Promise` is returned from the script, `$get_js()` will wait for the promise to resolve. To have JavaScript code execute asynchronously, wrap the code in a Promise object and have the script return an atomic value.

Arguments will have to be inserted into the script as there is not access to `arguments`. This can be done with commands like `paste()`. If using `glue::glue()`, be sure to use uncommon `.open` and `.close` values to avoid having to double all `{` and `}`.

Usage:

```
AppDriver$get_js(
  script = missing_arg(),
  ...,
  file = missing_arg(),
  timeout = missing_arg()
)
```

Arguments:

`script` JavaScript to execute. If a JavaScript Promise is returned, the R session will block until the promise has been resolved and return the value.

`...` Must be empty. Allows for parameter expansion.

`file` A (local) file containing JavaScript code to be read and used as the `script`. Only one of `script` or `file` can be specified.

`timeout` Amount of time to wait before giving up (milliseconds). Defaults to the resolved `timeout` value during the `AppDriver` initialization.

Returns: Result of the `script` (or `file` contents)

Examples:

```
library(shiny)
shiny_app <- shinyApp(h1("Empty App"), function(input, output) { })
app <- AppDriver$new(shiny_app)
```

```
# Execute JavaScript code in the app's browser
app$get_js("1 + 1;")
#> [1] 2

# Execute a JavaScript Promise. Return the resolved value.
app$get_js("
  new Promise((resolve) => {
    setTimeout(() => resolve(1 + 1), 1000)
  }).
  then((value) => value + 1);
")
#> [1] 3

# With escaped arguments
loc_field <- "hostname"
js_txt <- paste0("window.location[", jsonlite::toJSON(loc_field, auto_unbox = TRUE), "]")
app$get_js(js_txt)
#> [1] "127.0.0.1"

# With `glue::glue()`
js_txt <- glue::glue_data(
  lapply(
    list(x = 40, y = 2),
    jsonlite::toJSON,
    auto_unbox = TRUE
  ),
  .open = "<", .close = ">",
  "let answer = function(a, b) {\n",
  "  return a + b;\n",
  "};\n",
  "answer(<x>, <y>);\n"
)
app$get_js(js_txt)
#> [1] 42
```

`AppDriver$run_js()`**:** Execute JavaScript code in the browser

This function will block the local R session until the code has finished executing its *tick* in the browser.

The final result of the code will be ignored and not returned to the R session.

Usage:

```
AppDriver$run_js(
  script = missing_arg(),
  ...,
  file = missing_arg(),
  timeout = missing_arg()
)
```

Arguments:

`script` JavaScript to execute.

`...` Must be empty. Allows for parameter expansion.

`file` A (local) file containing JavaScript code to be read and used as the `script`. Only one of `script` or `file` can be specified.

`timeout` Amount of time to wait before giving up (milliseconds). Defaults to the resolved `timeout` value during the `AppDriver` initialization.

Examples:

```
library(shiny)
shiny_app <- shinyApp(h1("Empty App"), function(input, output) { })
app <- AppDriver$new(shiny_app)

# Get JavaScript answer from the app's browser
app$get_js("1 + 1")
#> [1] 2
# Execute JavaScript code in the app's browser
app$run_js("1 + 1")
# (Returns `app` invisibly)

# With escaped arguments
loc_field <- "hostname"
js_txt <- paste0("window.location[", jsonlite::toJSON(loc_field, auto_unbox = TRUE), "]")
app$run_js(js_txt)
app$get_js(js_txt)
#> [1] "127.0.0.1"
```

`AppDriver$expect_screenshot()`**:** Expect a screenshot of the Shiny application

This method takes a screenshot of the application (of only the `selector` area) and compares the image to the expected image.

Please be aware that this method is very brittle to changes outside of your Shiny application. These changes can include:

- running on a different R version
- running on a different in operating system
- using a different default system font
- using different package versions These differences are explicitly clear when working with plots.

Unless absolutely necessary for application consistency, it is strongly recommended to use other expectation methods.

Usage:

```
AppDriver$expect_screenshot(
  ...,
  threshold = getOption("shinytest2.compare_screenshot.threshold", NULL),
  kernel_size = getOption("shinytest2.compare_screenshot.kernel_size", 5),
  screenshot_args = missing_arg(),
  delay = missing_arg(),
  selector = missing_arg(),
  compare = missing_arg(),
```

```
  quiet = FALSE,
  name = NULL,
  cran = deprecated()
)
```

Arguments:

`...` Must be empty. Allows for parameter expansion.

`threshold` Parameter supplied to `compare_screenshot_threshold()` when using the default `compare` method. The default value can be set globally with the `shinytest2.compare_screenshot.threshold` option.
If the value of `threshold` is `NULL`, `compare_screenshot_threshold()` will act like `testthat::compare_file_bir`
However, if `threshold` is a positive number, it will be compared against the largest convolution value found if the two images fail a `testthat::compare_file_binary` comparison. Which value should I use? Threshold values values below 5 help deter false-positive screenshot comparisons (such as inconsistent rounded corners). Larger values in the 10s and 100s will help find *real* changes. However, not all values are one size fits all and you will need to play with a threshold that fits your needs.

`kernel_size` Parameter supplied to `compare_screenshot_threshold()` when using the default `compare` method. The `kernel_size` represents the height and width of the convolution kernel applied to the pixel differences. This integer-like value should be relatively small. The default value can be set globally with the `shinytest2.compare_screenshot.kernel_size` option.

`screenshot_args` This named list of arguments is passed along to `chromote::ChromoteSession`'s `$get_screenshot()` method. If missing, the value will default to `$new(screenshot_args=)`.
If `screenshot_args` is:

- `TRUE`: A screenshot of the browser's scrollable area will be taken with no delay
- A named list of arguments: Arguments passed directly to `chromote::ChromoteSession`'s `$get_screenshot()` method. The `delay` argument will default to `0` seconds. The `selector` argument can take two special values in addition to being a CSS DOM selector.
  - `"scrollable_area"` (default): The entire scrollable area will be captured. Typically this is your browser's viewport size, but it can be larger if the page is scrollable. This value works well with Apps that contain elements whose calculated dimensions may be different than their presented size.
  - `"viewport"`: This value will capture the browser's viewport in its current viewing location, height, and width. It will only capture what is currently being seen with `$view()`.

  In `v0.3.0`, the default `selector` value was changed from the HTML DOM selector (`"html"`) to entire scrollable area (`"scrollable_area"`).

`delay` The number of **seconds** to wait before taking the screenshot. This value can be supplied as `delay` or `screenshot_args$delay`, with the `delay` parameter having preference.

`selector` The selector is a CSS selector that will be used to select a portion of the page to be captured. This value can be supplied as `selector` or `screenshot_args$selector`, with the `selector` parameter having preference.
In `v0.3.0`, two special `selector` values were added:

- `"scrollable_area"` (default): The entire scrollable area will be captured. Typically this is your browser's viewport size, but it can be larger if the page is scrollable. This

value works well with Apps that contain elements whose calculated dimensions may be different than their presented size.

- `"viewport"`: This value will capture the browser's viewport in its current viewing location, height, and width. It will only capture what is currently being seen with `$view()`.

In `v0.3.0`, the default `selector` value was changed from the HTML DOM selector (`"html"`) to entire scrollable area (`"scrollable_area"`).

`compare` A function used to compare the screenshot snapshot files. The function should take two inputs, the paths to the `old` and `new` snapshot, and return either `TRUE` or `FALSE`. `compare` defaults to a function that wraps around `compare_screenshot_threshold(old, new, threshold = threshold, kernel_size = kernel_size, quiet = quiet)`. Note: if `threshold` is `NULL` (default), `compare` will behave as if `testthat::compare_file_binary()` was provided, comparing the two images byte-by-byte.

`quiet` Parameter supplied to `compare_screenshot_threshold()` when using the default `compare` method. If `FALSE`, diagnostic information will be presented when the computed value is larger than a non-`NULL` `threshold` value.

`name` The file name to be used for the snapshot. The file extension will overwritten to `.png`. By default, the `name` supplied to `app` on initialization with a counter will be used (e.g. `"NAME-001.png"`).

`cran` Deprecated. With `AppDriver` never testing on CRAN, this parameter no longer has any effect.

Examples:

```
# These example lines should be performed in a `./tests/testthat`
# test file so that snapshot files can be properly saved

app_path <- system.file("examples/01_hello", package = "shiny")
app <- AppDriver$new(app_path, variant = platform_variant())

# Expect a full size screenshot to be pixel perfect
app$expect_screenshot()

# Images are brittle when containing plots
app$expect_screenshot(selector = "#distPlot")

# Test with more threshold in pixel value differences
# Helps with rounded corners
app$expect_screenshot(threshold = 10)

# Equivalent expectations
app$expect_screenshot() # default
app$expect_screenshot(threshold = NULL)
app$expect_screenshot(compare = testthat::compare_file_binary)
expect_snapshot_file(
  app$get_screenshot(),
  variant = app$get_variant(),
  compare = testthat::compare_file_binary
)
```

```
# Equivalent expectations
app$expect_screenshot(threshold = 3, kernel_size = 5)
app$expect_screenshot(compare = function(old, new) {
  compare_screenshot_threshold(
    old, new,
    threshold = 3,
    kernel_size = 5
  )
})
expect_screenshot_file(
  app$get_screenshot(),
  variant = app$get_variant(),
  compare = function(old, new) {
    compare_screenshot_threshold(
      old, new,
      threshold = 3,
      kernel_size = 5
    )
  }
)

# Take a screenshot of the entire scrollable area
app$expect_screenshot()
app$expect_screenshot(selector = "scrollable_area")

## Take a screenshot of the current viewport
# Shrink the window to be smaller than the app
app$set_window_size(400, 500)
# Scroll the viewport just a bit
app$run_js("window.scroll(30, 70)")
# Take screenshot of browser viewport
app$expect_screenshot(selector = "viewport")
```

`AppDriver$get_screenshot()`**:** Take a screenshot

Take a screenshot of the Shiny application.

Usage:

```
AppDriver$get_screenshot(
  file = NULL,
  ...,
  screenshot_args = missing_arg(),
  delay = missing_arg(),
  selector = missing_arg()
)
```

Arguments:

`file` If `NULL`, then the image will be displayed to the current Graphics Device. If a file path, then the screenshot will be saved to that file.

`...` Must be empty. Allows for parameter expansion.

`screenshot_args` This named list of arguments is passed along to `chromote::ChromoteSession`'s `$get_screenshot()` method. If missing, the value will default to `$new(screenshot_args=)`. If `screenshot_args` is:

- `TRUE`: A screenshot of the browser's scrollable area will be taken with no delay
- A named list of arguments: Arguments passed directly to `chromote::ChromoteSession`'s `$get_screenshot()` method. The `delay` argument will default to `0` seconds. The `selector` argument can take two special values in addition to being a CSS DOM selector.
  - `"scrollable_area"` (default): The entire scrollable area will be captured. Typically this is your browser's viewport size, but it can be larger if the page is scrollable. This value works well with Apps that contain elements whose calculated dimensions may be different than their presented size.
  - `"viewport"`: This value will capture the browser's viewport in its current viewing location, height, and width. It will only capture what is currently being seen with `$view()`.

  In `v0.3.0`, the default `selector` value was changed from the HTML DOM selector (`"html"`) to entire scrollable area (`"scrollable_area"`).

If `screenshot_args=FALSE` is provided, the parameter will be ignored and a screenshot will be taken with default behavior.

`delay` The number of **seconds** to wait before taking the screenshot. This value can be supplied as `delay` or `screenshot_args$delay`, with the `delay` parameter having preference.

`selector` The selector is a CSS selector that will be used to select a portion of the page to be captured. This value can be supplied as `selector` or `screenshot_args$selector`, with the `selector` parameter having preference.

In `v0.3.0`, two special `selector` values were added:

- `"scrollable_area"` (default): The entire scrollable area will be captured. Typically this is your browser's viewport size, but it can be larger if the page is scrollable. This value works well with Apps that contain elements whose calculated dimensions may be different than their presented size.
- `"viewport"`: This value will capture the browser's viewport in its current viewing location, height, and width. It will only capture what is currently being seen with `$view()`.

In `v0.3.0`, the default `selector` value was changed from the HTML DOM selector (`"html"`) to entire scrollable area (`"scrollable_area"`).

Examples:

```
app_path <- system.file("examples/01_hello", package = "shiny")
app <- AppDriver$new(app_path)

# Display in graphics device
app$get_screenshot()

# Update bins then display `"disPlot"` in graphics device
app$set_inputs(bins = 10)
app$get_screenshot(selector = "#distPlot")

# Save screenshot to file and view it
tmpfile <- tempfile(fileext = ".png")
```

```
app$get_screenshot(tmpfile)
showimage::show_image(tmpfile)
```

`AppDriver$wait_for_idle()`**:** Wait for Shiny to not be busy (idle) for a set amount of time

Waits until Shiny has not been busy for a set duration of time, e.g. no reactivity is updating or has occurred.

This is useful, for example, when waiting for your application to initialize or if you've resized the window with `$set_window_size()` and want to make sure all plot redrawing is complete before take a screenshot.

By default,

- `$new(wait = TRUE)` waits for Shiny to not be busy after initializing the application
- `$set_window_size(wait = TRUE)` waits for Shiny to not be busy after resizing the window.)

Usage:

```
AppDriver$wait_for_idle(duration = 500, timeout = missing_arg())
```

Arguments:

`duration` How long Shiny must be idle (in ms) before unblocking the R session.

`timeout` Amount of time to wait before giving up (milliseconds). Defaults to the resolved `timeout` value during the `AppDriver` initialization.

Returns: `invisible(self)` if Shiny stabilizes within the `timeout`. Otherwise an error will be thrown

Examples:

```
app_path <- system.file("examples/01_hello", package = "shiny")
app <- AppDriver$new(app_path)

pre_value <- app$get_value(output = "distPlot")
# Update bins value
app$set_inputs(bins = 10, wait_ = FALSE)
middle_value <- app$get_value(output = "distPlot")
app$wait_for_idle()
post_value <- app$get_value(output = "distPlot")

# No guarantee that these values are different
identical(pre_value, middle_value)
# Will not be equal
identical(pre_value, post_value)

# ----------------------
## Change the screen size to trigger a plot update
pre_value <- app$get_value(output = "distPlot")
app$set_window_size(height = 1080, width = 1920, wait = FALSE)
middle_value <- app$get_value(output = "distPlot")
app$wait_for_idle()
post_value <- app$get_value(output = "distPlot")

# No guarantee that these values are different
identical(pre_value, middle_value)
```

```
# Will not be equal
identical(pre_value, post_value)
```

`AppDriver$wait_for_value()`: Wait for a new Shiny value

Waits until an `input`, `output`, or `export` Shiny value is not one of `ignore`d values, or the `timeout` is reached.

Only a single `input`, `output`, or `export` value may be used.

This function can be useful in helping determine if an application has finished processing a complex reactive situation.

Usage:

```
AppDriver$wait_for_value(
  ...,
  input = missing_arg(),
  output = missing_arg(),
  export = missing_arg(),
  ignore = list(NULL, ""),
  timeout = missing_arg(),
  interval = 400
)
```

Arguments:

`...` Must be empty. Allows for parameter expansion.

`input`, `output`, `export` A name of an input, output, or export value. Only one of these parameters may be used.

`ignore` List of possible values to ignore when checking for updates.

`timeout` Amount of time to wait before giving up (milliseconds). Defaults to the resolved `timeout` value during the `AppDriver` initialization.

`interval` How often to check for the condition, in ms.

`timeout_` Amount of time to wait for a new `output` value before giving up (milliseconds). Defaults to the resolved `timeout` value during the `AppDriver` initialization.

Returns: Newly found value

Examples:

```
library(shiny)
shiny_app <- shinyApp(
  fluidPage(
    h1("Dynamic output"),
    actionButton("display", "Display UI"),
    uiOutput("dym1"),
  ),
  function(input, output) {
    output$dym1 <- renderUI({
      req(input$display)
      Sys.sleep(runif(1, max = 2)) # Artificial calculations
      tagList(
        sliderInput("slider1", "Slider #1", 0, 100, 25),
        uiOutput("dym2")
```

```
      )
    })
    output$dym2 <- renderUI({
      Sys.sleep(runif(1, max = 2)) # Artificial calculations
      tagList(
        sliderInput("slider2", "Slider #2", 0, 100, 50),
        "Total:", verbatimTextOutput("total")
      )
    })
    output$total <- renderText({
      req(input$slider1, input$slider2)
      input$slider1 + input$slider2
    })
  }
)

app <- AppDriver$new(shiny_app)

# Create UI / output values
app$click("display")
# Wait for total to be calculated (or have a non-NULL value)
new_total_value <- app$wait_for_value(output = "total")
#> [1] "75"
app$get_value(output = "total")
#> [1] "75"
```

`AppDriver$wait_for_js()`**:** Wait for a JavaScript expression to be true

Waits until a JavaScript `expression` evaluates to `true` or the `timeout` is exceeded.

Usage:

```
AppDriver$wait_for_js(script, timeout = missing_arg(), interval = 100)
```

Arguments:

`script` A string containing JavaScript code. This code must eventually return a truethy value or a timeout error will be thrown.

`timeout` How long the script has to return a `truethy` value (milliseconds). Defaults to the resolved `timeout` value during the `AppDriver` initialization.

`interval` How often to check for the condition (milliseconds).

Returns: `invisible(self)` if expression evaluates to `true` without error within the timeout. Otherwise an error will be thrown

Examples:

```
shiny_app <- shinyApp(h1("Empty App"), function(input, output) { })
app <- AppDriver$new(shiny_app)

# Contrived example:
# Wait until `Date.now()` returns a number that ends in a 5. (0 - 10 seconds)
system.time(
  app$wait_for_js("Math.floor((Date.now() / 1000) % 10) == 5;")
```

```
)

## A second example where we run the contents of a JavaScript file
## and use the result to wait for a condition
app$run_js(file = "complicated_file.js")
app$wait_for_js("complicated_condition();")
```

`AppDriver$expect_unique_names()`**:** Expect unique input and output names.

If the HTML has duplicate input or output elements with matching `id` values, this function will throw an error. It is similar to `AppDriver$new(check_names = TRUE)`, but asserts that no warnings are displayed.

This method will not throw if a single input and a single output have the same name.

Usage:

```
AppDriver$expect_unique_names()
```

Examples:

```
shiny_app <- shinyApp(
  ui = fluidPage(
    # Duplicate input IDs: `"text"`
    textInput("text", "Text 1"),
    textInput("text", "Text 2")
  ),
  server = function(input, output) {
    # empty
  }
)
# Initial checking for unique names (default behavior)
app <- AppDriver$new(shiny_app, check_names = TRUE)
#> Warning:
#> ! Shiny inputs should have unique HTML id values.
#> i The following HTML id values are not unique:
#> * text
app$stop()

# Manually assert that all names are unique
app <- AppDriver$new(shiny_app, check_names = FALSE)
app$expect_unique_names()
#> Error: `app_check_unique_names(self, private)` threw an unexpected warning.
#> Message: ! Shiny inputs should have unique HTML id values.
#> i The following HTML id values are not unique:
#>   * text
#> Class:   rlang_warning/warning/condition
app$stop()
```

`AppDriver$get_dir()`**:** Retrieve the Shiny app path

Usage:

```
AppDriver$get_dir()
```

Returns: The directory containing the Shiny application or Shiny runtime document. If a URL was provided to `app_dir` during initialization, the current directory will be returned.

Examples:

```
app_path <- system.file("examples/01_hello", package = "shiny")
app <- AppDriver$new(app_path)

identical(app$get_dir(), app_path)
#> [1] TRUE
```

`AppDriver$get_url()`**:** Retrieve the Shiny app URL

Usage:

```
AppDriver$get_url()
```

Returns: URL where the Shiny app is being hosted

Examples:

```
app_path <- system.file("examples/01_hello", package = "shiny")
app <- AppDriver$new(app_path)

browseURL(app$get_url())
```

`AppDriver$get_window_size()`**:** Get window size

Get current size of the browser window, as list of numeric scalars named `width` and `height`.

Usage:

```
AppDriver$get_window_size()
```

Examples:

```
app_path <- system.file("examples/01_hello", package = "shiny")
app <- AppDriver$new(app_path)

app$get_window_size()
#> $width
#> [1] 992
#>
#> $height
#> [1] 1323
```

`AppDriver$set_window_size()`**:** Sets size of the browser window.

Usage:

```
AppDriver$set_window_size(width, height, wait = TRUE)
```

Arguments:

`width`, `height` Height and width of browser, in pixels.

`wait` If `TRUE`, `$wait_for_idle()` will be called after setting the window size. This will block until any width specific items (such as plots) that need to be re-rendered.

Examples:

```
app_path <- system.file("examples/01_hello", package = "shiny")
# Set init window size
app <- AppDriver$new(app_path, height = 1400, width = 1000)

app$get_window_size()
#> $width
#> [1] 1000
#>
#> $height
#> [1] 1400

# Manually set the window size
app$set_window_size(height = 1080, width = 1920)
app$get_window_size()
#> $width
#> [1] 1920
#>
#> $height
#> [1] 1080
```

`AppDriver$get_chromote_session()`**:** Get Chromote Session

Get the `chromote::ChromoteSession` object from the **chromote** package.

Usage:

```
AppDriver$get_chromote_session()
```

Returns: `chromote::ChromoteSession` R6 object

Examples:

```
app_path <- system.file("examples/01_hello", package = "shiny")
app <- AppDriver$new(app_path)

b <- app$get_chromote_session()
b$Runtime$evaluate("1 + 1")
#> $result
#> $result$type
#> [1] "number"
#>
#> $result$value
#> [1] 2
#>
#> $result$description
#> [1] "2"
```

`AppDriver$get_process()`**:** Get the Shiny process

Get the `callr::r_bg()` process that runs the Shiny app in the background. Use it to check if the app is still running, or to wait for an app that calls `shiny::stopApp()` to exit before calling `$stop()`.

Usage:

```
AppDriver$get_process()
```

Returns: callr::r_process R6 object, or NULL if the app is not hosted locally (e.g. app_dir is a URL)

Examples:

```
app_path <- system.file("examples/01_hello", package = "shiny")
app <- AppDriver$new(app_path)

app$get_process()$is_alive()
#> [1] TRUE
app$stop()
app$get_process()$is_alive()
#> [1] FALSE
```

`AppDriver$get_variant()`**:** Get the variant

Get the variant supplied during initialization

Usage:

```
AppDriver$get_variant()
```

Returns: The variant value supplied during initialization or NULL if no value was supplied.

Examples:

```
app_path <- system.file("examples/01_hello", package = "shiny")

app <- AppDriver$new(app_path)

app$get_variant()
#> NULL

app <- AppDriver$new(app_path, variant = platform_variant())
app$get_variant()
#> [1] "mac-4.1"
```

`AppDriver$get_logs()`**:** Get all logs

Retrieve all of the debug logs that have been recorded.

Usage:

```
AppDriver$get_logs()
```

Returns: A data.frame with the following columns:

- workerid: The shiny worker ID found within the browser
- timestamp: POSIXct timestamp of the message
- location: The location of the message was found. One of three values:
  - "shinytest2": Occurs when $log_message() is called
  - "shiny": stdin and stdout messages from the Shiny server. Note message() output is sent to stdout.
  - "chromote": Captured by the **chromote** event handlers. See console API, exception thrown, websocket sent, and websocket received for more details
- level: For a given location, there are different types of log levels.
  - "shinytest2": "info"; Only log messages are captured.

- "shiny": "stdout" or "stderr"; Note, message() output is sent to stderr.
- "chromote": Correspond to any level of a JavaScript console.LEVEL() function call. Typically, these are "log"and"error"but can include"info", "debug", and "warn". If options(shiny.trace = TRUE), then the level will recorded as "websocket"‘.

Examples:

```
app1 <- AppDriver$new(system.file("examples/01_hello", package = "shiny"))

app1$get_logs()
#> {shinytest2} R  info   10:00:28.86 Start AppDriver initialization
#> {shinytest2} R  info   10:00:28.86 Starting Shiny app
#> {shinytest2} R  info   10:00:29.76 Creating new ChromoteSession
#> {shinytest2} R  info   10:00:30.56 Navigating to Shiny app
#> {shinytest2} R  info   10:00:30.70 Injecting shiny-tracer.js
#> {chromote}   JS info   10:00:30.75 shinytest2; jQuery found
#> {chromote}  JS info  10:00:30.77 shinytest2; Waiting for shiny session to connect
#> {chromote}   JS info   10:00:30.77 shinytest2; Loaded
#> {shinytest2} R  info   10:00:30.77 Waiting for Shiny to become ready
#> {chromote}   JS info   10:00:30.90 shinytest2; Connected
#> {chromote}   JS info   10:00:30.95 shinytest2; shiny:busy
#> {shinytest2} R  info  10:00:30.98 Waiting for Shiny to become idle for 200ms within 15000ms
#> {chromote}  JS info  10:00:30.98 shinytest2; Waiting for Shiny to be stable
#> {chromote}   JS info   10:00:31.37 shinytest2; shiny:idle
#> {chromote}   JS info   10:00:31.38 shinytest2; shiny:value distPlot
#> {chromote}  JS info  10:00:31.57 shinytest2; Shiny has been idle for 200ms
#> {shinytest2} R  info   10:00:31.57 Shiny app started
#> {shiny}      R  stderr ----------- Loading required package: shiny
#> {shiny}      R  stderr ----------- Running application in test mode.
#> {shiny}      R  stderr -----------
#> {shiny}      R  stderr ----------- Listening on http://127.0.0.1:4679


# To capture all websocket traffic, set `options = list(shiny.trace = TRUE)`
app2 <- AppDriver$new(
  system.file("examples/01_hello", package = "shiny"),
  options = list(shiny.trace = TRUE)
)

app2$get_logs()
## (All WebSocket messages have been replaced with `WEBSOCKET_MSG` in example below)
#> {shinytest2} R  info      10:01:57.49 Start AppDriver initialization
#> {shinytest2} R  info      10:01:57.50 Starting Shiny app
#> {shinytest2} R  info      10:01:58.20 Creating new ChromoteSession
#> {shinytest2} R  info      10:01:58.35 Navigating to Shiny app
#> {shinytest2} R  info      10:01:58.47 Injecting shiny-tracer.js
#> {chromote}   JS info      10:01:58.49 shinytest2; jQuery not found
#> {chromote}   JS info      10:01:58.49 shinytest2; Loaded
#> {shinytest2} R  info      10:01:58.50 Waiting for Shiny to become ready
#> {chromote}   JS info      10:01:58.55 shinytest2; jQuery found
```

```
#> {chromote}  JS info    10:01:58.55 shinytest2; Waiting for shiny session to connect
#> {chromote}   JS websocket 10:01:58.64 send WEBSOCKET_MSG
#> {chromote}   JS websocket 10:01:58.67 recv WEBSOCKET_MSG
#> {chromote}   JS info      10:01:58.67 shinytest2; Connected
#> {chromote}   JS websocket 10:01:58.71 recv WEBSOCKET_MSG
#> {chromote}   JS websocket 10:01:58.72 recv WEBSOCKET_MSG
#> {chromote}   JS info      10:01:58.72 shinytest2; shiny:busy
#> {chromote}   JS websocket 10:01:58.73 recv WEBSOCKET_MSG
#> {chromote}   JS websocket 10:01:58.73 recv WEBSOCKET_MSG
#> {shinytest2} R  info     10:01:58.75 Waiting for Shiny to become idle for 200ms within 15000ms
#> {chromote}  JS info    10:01:58.75 shinytest2; Waiting for Shiny to be stable
#> {chromote}   JS websocket 10:01:58.81 recv WEBSOCKET_MSG
#> {chromote}   JS websocket 10:01:58.81 recv WEBSOCKET_MSG
#> {chromote}   JS info      10:01:58.81 shinytest2; shiny:idle
#> {chromote}   JS websocket 10:01:58.82 recv WEBSOCKET_MSG
#> {chromote}   JS info      10:01:58.82 shinytest2; shiny:value distPlot
#> {chromote}  JS info    10:01:59.01 shinytest2; Shiny has been idle for 200ms
#> {shinytest2} R  info      10:01:59.01 Shiny app started
#> {shiny}      R  stderr    ----------- Loading required package: shiny
#> {shiny}      R  stderr    ----------- Running application in test mode.
#> {shiny}      R  stderr    -----------
#> {shiny}      R  stderr    ----------- Listening on http://127.0.0.1:4560
#> {shiny}    R  stderr   ----------- SEND {"config":{"workerId":"","sessionId"}}|truncated
#> {shiny}    R  stderr   ----------- RECV {"method":"init","data":{"bins":30,}}|truncated
#> {shiny}    R  stderr   ----------- SEND {"custom":{"showcase-src":{"srcref":}}}|truncated
#> {shiny}      R  stderr    ----------- SEND {"busy":"busy"}
#> {shiny}    R  stderr   ----------- SEND {"custom":{"showcase-src":{"srcref":}}}|truncated
#> {shiny}    R  stderr   ----------- SEND {"recalculating":{"name":"distPlot",}}|truncated
#> {shiny}    R  stderr   ----------- SEND {"recalculating":{"name":"distPlot",}}|truncated
#> {shiny}      R  stderr    ----------- SEND {"busy":"idle"}
#> {shiny}    R  stderr   ----------- SEND {"errors":{},"values":{"distPlot":}}|truncated

# The log that is returned is a `data.frame()`.
log <- app2$get_logs()
tibble::glimpse(log)
#> Rows: 43
#> Columns: 5
#> $ workerid  <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
#> $ timestamp <dttm> 2022-09-19 10:01:57, 2022-09-19 10:01:57, 2022-09-19 10:01:58, 2022...
#> $ location  <chr> "shinytest2", "shinytest2", "shinytest2", "shinytest2", "shinytest2"...
#> $ level    <chr> "info", "info", "info", "info", "info", "info", "info", "info", "inf...
#> $ message   <chr> "Start AppDriver initialization", "Starting Shiny app", "Creating ne...
#> $ workerid  <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
#> $ timestamp <dttm> 2022-03-16 11:09:57, 2022-03-16 11:09:57, 2022-03-16 11:09:...
#> $ location  <chr> "shinytest2", "shinytest2", "shinytest2", "shinytest2", "shi...
#> $ level    <chr> "info", "info", "info", "info", "info", "info", "info", "inf...
#> $ message   <chr> "Start AppDriver initialization", "Starting Shiny app", "Cre...
```

```
# It may be filtered to find desired logs
subset(log, level == "websocket")
## (All WebSocket messages have been replaced with `WEBSOCKET_MSG` in example below)
#> {chromote} JS websocket 10:01:58.64 send WEBSOCKET_MSG
#> {chromote} JS websocket 10:01:58.67 recv WEBSOCKET_MSG
#> {chromote} JS websocket 10:01:58.71 recv WEBSOCKET_MSG
#> {chromote} JS websocket 10:01:58.72 recv WEBSOCKET_MSG
#> {chromote} JS websocket 10:01:58.73 recv WEBSOCKET_MSG
#> {chromote} JS websocket 10:01:58.73 recv WEBSOCKET_MSG
#> {chromote} JS websocket 10:01:58.81 recv WEBSOCKET_MSG
#> {chromote} JS websocket 10:01:58.81 recv WEBSOCKET_MSG
#> {chromote} JS websocket 10:01:58.82 recv WEBSOCKET_MSG
```

`AppDriver$log_message()`**:** Add a message to the **shinytest2** log.

Usage:

```
AppDriver$log_message(message)
```

Arguments:

`message` Single message to store in log

Examples:

```
app_path <- system.file("examples/01_hello", package = "shiny")
app <- AppDriver$new(app_path)

app$log_message("Setting bins to smaller value")
app$set_inputs(bins = 10)
app$get_logs()
```

`AppDriver$stop()`**:** Stop the Shiny application driver

This method stops all known processes:

- The Shiny application in the background R process,
- the background R process hosting the Shiny application, and
- the Chromote Session instance.

To stop your shiny application and return a value from `$stop()`, see `shiny::stopApp()`. This is useful in testing to return context information.

Typically, this can be paired with a button that when clicked will call `shiny::stopApp(info)` to return `info` from the test app back to the main R session.

Usage:

```
AppDriver$stop(signal_timeout = missing_arg())
```

Arguments:

`signal_timeout` Milliseconds to wait between sending a `SIGINT`, `SIGTERM`, and `SIGKILL` to the Shiny process. Defaults to 500ms and does not utilize the resolved value from `AppDriver$new(timeout=)`. However, if **covr** is currently executing, then the `timeout` is set to 20,000ms to allow for the coverage report to be generated.

Returns: The result of the background process if the Shiny application has already been terminated.

Examples:

```
rlang::check_installed("reactlog")

library(shiny)
shiny_app <- shinyApp(
  ui = fluidPage(
    actionButton("button", "Stop app and return Reactlog"),
    "Click count:", textOutput("count")
  ),
  server = function(input, output) {
    output$count <- renderText({ input$button })
    observe({
      req(input$button)
      stopApp(shiny::reactlog())
    })
  }
)

app <- AppDriver$new(
  shiny_app,
  # Enable reactlog in background R session
  options = list(shiny.reactlog = TRUE)
)

app$click("button")
rlog <- app$stop()
str(head(rlog, 2))
#> List of 2
#> $ :List of 7
#> ..$ action : chr "define"
#> ..$ reactId: chr "r3"
#> ..$ label  : chr "Theme Counter"
#> ..$ type   : chr "reactiveVal"
#> ..$ value  : chr " num 0"
#> ..$ session: chr "bdc7417f2fc8c84fc05c9518e36fdc44"
#> ..$ time   : num 1.65e+09
#> $ :List of 7
#> ..$ action : chr "define"
#> ..$ reactId: chr "r4"
#> ..$ label  : chr "output$count"
#> .. ..- attr(*, "srcref")= int [1:6] 7 32 7 45 32 45
#> .. ..- attr(*, "srcfile")= chr ""
#> ..$ type   : chr "observer"
#> ..$ value  : chr " NULL"
#> ..$ session: chr "bdc7417f2fc8c84fc05c9518e36fdc44"
#> ..$ time   : num 1.65e+09
```

See Also

platform_variant(), use_shinytest2_test()

Examples

```
## ------------------------------------------------
## Method `AppDriver$new()`
## ------------------------------------------------

## Not run:
# Create an AppDriver from the Shiny app in the current directory
app <- AppDriver()

# Create an AppDriver object from a different Shiny app directory
example_app <- system.file("examples/01_hello", package = "shiny")
app <- AppDriver(example_app)

# Expect consistent inital values
app$expect_values()

## End(Not run)

## ------------------------------------------------
## Method `AppDriver$view()`
## ------------------------------------------------

## Not run:
# Open app in Chrome
app$view()

## End(Not run)

## ------------------------------------------------
## Method `AppDriver$click()`
## ------------------------------------------------

## Not run:
app_path <- system.file("examples/07_widgets", package = "shiny")
app <- AppDriver$new(app_path)

tmpfile <- write.csv(cars, "cars.csv")
app$upload_file(file1 = tmpfile)
cat(app$get_text("#view"))
app$set_inputs(dataset = "cars", obs = 6)
app$click("update")
cat(app$get_text("#view"))

## End(Not run)

## ------------------------------------------------
## Method `AppDriver$set_inputs()`
## ------------------------------------------------
```

```
## Not run:
app_path <- system.file("examples/07_widgets", package = "shiny")
app <- AppDriver$new(app_path)

cat(app$get_text("#view"))
app$set_inputs(dataset = "cars", obs = 6)
app$click("update")
cat(app$get_text("#view"))

## End(Not run)

## ------------------------------------------------
## Method `AppDriver$upload_file()`
## ------------------------------------------------

## Not run:
app_path <- system.file("examples/09_upload", package = "shiny")
app <- AppDriver$new(app_path)

# Save example file
tmpfile <- tempfile(fileext = ".csv")
write.csv(cars, tmpfile, row.names = FALSE)

# Upload file to input named: file1
app$upload_file(file1 = tmpfile)

## End(Not run)

## ------------------------------------------------
## Method `AppDriver$expect_values()`
## ------------------------------------------------

## Not run:
library(shiny)
shiny_app <- shinyApp(
  fluidPage(
    h1("Pythagorean theorem"),
    numericInput("A", "A", 3),
    numericInput("B", "B", 4),
    verbatimTextOutput("C"),
  ),
  function(input, output) {
    a_squared <- reactive({ req(input$A); input$A * input$A })
    b_squared <- reactive({ req(input$B); input$B * input$B })
    c_squared <- reactive({ a_squared() + b_squared() })
    c_value <- reactive({ sqrt(c_squared()) })
    output$C <- renderText({ c_value() })

    exportTestValues(
      a_squared = { a_squared() },
      b_squared = { b_squared() },
      c_squared = { c_squared() }
```

```
    )
  }
)

app <- AppDriver$new(shiny_app)

# Snapshot all known values
app$expect_values()

# Snapshot only `export` values
app$expect_values(export = TRUE)

# Snapshot values `"A"` from `input` and `"C"` from `output`
app$expect_values(input = "A", output = "C")

## End(Not run)

## ------------------------------------------------
## Method `AppDriver$get_value()`
## ------------------------------------------------

## Not run:
app_path <- system.file("examples/04_mpg", package = "shiny")
app <- AppDriver$new(app_path)

# Retrieve a single value
app$get_value(output = "caption")
#> [1] "mpg ~ cyl"
# Equivalent code using `$get_values()`
app$get_values(output = "caption")$output$caption
#> [1] "mpg ~ cyl"

## End(Not run)

## ------------------------------------------------
## Method `AppDriver$get_values()`
## ------------------------------------------------

## Not run:
library(shiny)
shiny_app <- shinyApp(
  fluidPage(
    h1("Pythagorean theorem"),
    numericInput("A", "A", 3),
    numericInput("B", "B", 4),
    verbatimTextOutput("C"),
  ),
  function(input, output) {
    a_squared <- reactive({ req(input$A); input$A * input$A })
    b_squared <- reactive({ req(input$B); input$B * input$B })
    c_squared <- reactive({ a_squared() + b_squared() })
    c_value <- reactive({ sqrt(c_squared()) })
    output$C <- renderText({ c_value() })
```

```
    exportTestValues(
      a_squared = { a_squared() },
      b_squared = { b_squared() },
      c_squared = { c_squared() }
    )
  }
)

app <- AppDriver$new(shiny_app)

# Show all known values
str(app$get_values())
#> List of 3
#> $ input :List of 2
#> ..$ A: int 3
#> ..$ B: int 4
#> $ output:List of 1
#> ..$ C: chr "5"
#> $ export:List of 3
#> ..$ a_squared: int 9
#> ..$ b_squared: int 16
#> ..$ c_squared: int 25

# Get only `export` values
str(app$get_values(export = TRUE))
#> List of 1
#> $ export:List of 3
#> ..$ a_squared: int 9
#> ..$ b_squared: int 16
#> ..$ c_squared: int 25

# Get values `"A"` from `input` and `"C"` from `output`
str(app$get_values(input = "A", output = "C"))
#> List of 2
#> $ input :List of 1
#> ..$ A: int 3
#> $ output:List of 1
#> ..$ C: chr "5"

## End(Not run)

## ------------------------------------------------
## Method `AppDriver$expect_download()`
## ------------------------------------------------

## Not run:
app_path <- system.file("examples/10_download", package = "shiny")
app <- AppDriver$new(app_path)

# Save snapshot of rock.csv as 001.download
# Save snapshot value of `rock.csv` to capture default file name
app$expect_download("downloadData", compare = testthat::compare_file_text)
```

```
## End(Not run)

## ------------------------------------------------
## Method `AppDriver$get_download()`
## ------------------------------------------------

## Not run:
app_path <- system.file("examples/10_download", package = "shiny")
app <- AppDriver$new(app_path)

# Get rock.csv as a tempfile
app$get_download("downloadData")
#> [1] "/TEMP/PATH/rock.csv"

# Get rock.csv as a "./myfile.csv"
app$get_download("downloadData", filename = "./myfile.csv")
#> [1] "./myfile.csv"

## End(Not run)

## ------------------------------------------------
## Method `AppDriver$expect_text()`
## ------------------------------------------------

## Not run:
hello_app <- system.file("examples/01_hello", package = "shiny")
app <- AppDriver$new(hello_app)

# Make a snapshot of `"Hello Shiny!"`
app$expect_text("h2")

## End(Not run)

## ------------------------------------------------
## Method `AppDriver$get_text()`
## ------------------------------------------------

## Not run:
hello_app <- system.file("examples/01_hello", package = "shiny")
app <- AppDriver$new(hello_app)

app$get_text("h2")
#> [1] "Hello Shiny!"

## End(Not run)

## ------------------------------------------------
## Method `AppDriver$expect_html()`
## ------------------------------------------------

## Not run:
app_path <- system.file("examples/04_mpg", package = "shiny")
```

```
app <- AppDriver$new(app_path)

# Save a snapshot of the `caption` output
app$expect_html("#caption")

## End(Not run)

## ------------------------------------------------
## Method `AppDriver$get_html()`
## ------------------------------------------------

## Not run:
app_path <- system.file("examples/03_reactivity", package = "shiny")
app <- AppDriver$new(app_path, check_names = FALSE)

app$set_inputs(caption = "Custom value!")
cat(app$get_html(".shiny-input-container")[1])
#> <div class="form-group shiny-input-container">
#>   <label class="control-label" id="caption-label" for="caption">Caption:</label>
#>  <input id="caption" type="text" class="form-control shiny-bound-input" value="Data Summary">
#> </div>
## ^^ No update to the DOM of `caption`

## End(Not run)

## ------------------------------------------------
## Method `AppDriver$expect_js()`
## ------------------------------------------------

## Not run:
app_path <- system.file("examples/07_widgets", package = "shiny")
app <- AppDriver$new(app_path)

# Track how many clicks are given to `#update` button
app$run_js("
  window.test_counter = 0;
  $('#update').click(() => window.test_counter++);
")
app$set_inputs(obs = 20)
# Click the update button, incrementing the counter
app$click("update")
# Save a snapshot of number of clicks (1)
app$expect_js("window.test_counter;")

## End(Not run)

## ------------------------------------------------
## Method `AppDriver$get_js()`
## ------------------------------------------------

## Not run:
library(shiny)
shiny_app <- shinyApp(h1("Empty App"), function(input, output) { })
```

```
app <- AppDriver$new(shiny_app)

# Execute JavaScript code in the app's browser
app$get_js("1 + 1;")
#> [1] 2

# Execute a JavaScript Promise. Return the resolved value.
app$get_js("
  new Promise((resolve) => {
    setTimeout(() => resolve(1 + 1), 1000)
  }).
  then((value) => value + 1);
")
#> [1] 3

# With escaped arguments
loc_field <- "hostname"
js_txt <- paste0("window.location[", jsonlite::toJSON(loc_field, auto_unbox = TRUE), "]")
app$get_js(js_txt)
#> [1] "127.0.0.1"

# With `glue::glue()`
js_txt <- glue::glue_data(
  lapply(
    list(x = 40, y = 2),
    jsonlite::toJSON,
    auto_unbox = TRUE
  ),
  .open = "<", .close = ">",
  "let answer = function(a, b) {\n",
  "  return a + b;\n",
  "};\n",
  "answer(<x>, <y>);\n"
)
app$get_js(js_txt)
#> [1] 42

## End(Not run)

## ------------------------------------------------
## Method `AppDriver$run_js()`
## ------------------------------------------------

## Not run:
library(shiny)
shiny_app <- shinyApp(h1("Empty App"), function(input, output) { })
app <- AppDriver$new(shiny_app)

# Get JavaScript answer from the app's browser
app$get_js("1 + 1")
#> [1] 2
# Execute JavaScript code in the app's browser
app$run_js("1 + 1")
```

```
# (Returns `app` invisibly)

# With escaped arguments
loc_field <- "hostname"
js_txt <- paste0("window.location[", jsonlite::toJSON(loc_field, auto_unbox = TRUE), "]")
app$run_js(js_txt)
app$get_js(js_txt)
#> [1] "127.0.0.1"

## End(Not run)

## ------------------------------------------------
## Method `AppDriver$expect_screenshot()`
## ------------------------------------------------

## Not run:
# These example lines should be performed in a `./tests/testthat`
# test file so that snapshot files can be properly saved

app_path <- system.file("examples/01_hello", package = "shiny")
app <- AppDriver$new(app_path, variant = platform_variant())

# Expect a full size screenshot to be pixel perfect
app$expect_screenshot()

# Images are brittle when containing plots
app$expect_screenshot(selector = "#distPlot")

# Test with more threshold in pixel value differences
# Helps with rounded corners
app$expect_screenshot(threshold = 10)

# Equivalent expectations
app$expect_screenshot() # default
app$expect_screenshot(threshold = NULL)
app$expect_screenshot(compare = testthat::compare_file_binary)
expect_snapshot_file(
  app$get_screenshot(),
  variant = app$get_variant(),
  compare = testthat::compare_file_binary
)

# Equivalent expectations
app$expect_screenshot(threshold = 3, kernel_size = 5)
app$expect_screenshot(compare = function(old, new) {
  compare_screenshot_threshold(
    old, new,
    threshold = 3,
    kernel_size = 5
  )
})
expect_screenshot_file(
  app$get_screenshot(),
```

```
  variant = app$get_variant(),
  compare = function(old, new) {
    compare_screenshot_threshold(
      old, new,
      threshold = 3,
      kernel_size = 5
    )
  }
)

# Take a screenshot of the entire scrollable area
app$expect_screenshot()
app$expect_screenshot(selector = "scrollable_area")

## Take a screenshot of the current viewport
# Shrink the window to be smaller than the app
app$set_window_size(400, 500)
# Scroll the viewport just a bit
app$run_js("window.scroll(30, 70)")
# Take screenshot of browser viewport
app$expect_screenshot(selector = "viewport")

## End(Not run)

## ------------------------------------------------
## Method `AppDriver$get_screenshot()`
## ------------------------------------------------

## Not run:
app_path <- system.file("examples/01_hello", package = "shiny")
app <- AppDriver$new(app_path)

# Display in graphics device
app$get_screenshot()

# Update bins then display `"disPlot"` in graphics device
app$set_inputs(bins = 10)
app$get_screenshot(selector = "#distPlot")

# Save screenshot to file and view it
tmpfile <- tempfile(fileext = ".png")
app$get_screenshot(tmpfile)
showimage::show_image(tmpfile)

## End(Not run)

## ------------------------------------------------
## Method `AppDriver$wait_for_idle()`
## ------------------------------------------------

## Not run:
app_path <- system.file("examples/01_hello", package = "shiny")
app <- AppDriver$new(app_path)
```

```
pre_value <- app$get_value(output = "distPlot")
# Update bins value
app$set_inputs(bins = 10, wait_ = FALSE)
middle_value <- app$get_value(output = "distPlot")
app$wait_for_idle()
post_value <- app$get_value(output = "distPlot")

# No guarantee that these values are different
identical(pre_value, middle_value)
# Will not be equal
identical(pre_value, post_value)

# ---------------------
## Change the screen size to trigger a plot update
pre_value <- app$get_value(output = "distPlot")
app$set_window_size(height = 1080, width = 1920, wait = FALSE)
middle_value <- app$get_value(output = "distPlot")
app$wait_for_idle()
post_value <- app$get_value(output = "distPlot")

# No guarantee that these values are different
identical(pre_value, middle_value)
# Will not be equal
identical(pre_value, post_value)

## End(Not run)

## ------------------------------------------------
## Method `AppDriver$wait_for_value()`
## ------------------------------------------------

## Not run:
library(shiny)
shiny_app <- shinyApp(
  fluidPage(
    h1("Dynamic output"),
    actionButton("display", "Display UI"),
    uiOutput("dym1"),
  ),
  function(input, output) {
    output$dym1 <- renderUI({
      req(input$display)
      Sys.sleep(runif(1, max = 2)) # Artificial calculations
      tagList(
        sliderInput("slider1", "Slider #1", 0, 100, 25),
        uiOutput("dym2")
      )
    })
    output$dym2 <- renderUI({
      Sys.sleep(runif(1, max = 2)) # Artificial calculations
      tagList(
        sliderInput("slider2", "Slider #2", 0, 100, 50),
```

```
        "Total:", verbatimTextOutput("total")
      )
    })
    output$total <- renderText({
      req(input$slider1, input$slider2)
      input$slider1 + input$slider2
    })
  }
)

app <- AppDriver$new(shiny_app)

# Create UI / output values
app$click("display")
# Wait for total to be calculated (or have a non-NULL value)
new_total_value <- app$wait_for_value(output = "total")
#> [1] "75"
app$get_value(output = "total")
#> [1] "75"

## End(Not run)

## ------------------------------------------------
## Method `AppDriver$wait_for_js()`
## ------------------------------------------------

## Not run:
shiny_app <- shinyApp(h1("Empty App"), function(input, output) { })
app <- AppDriver$new(shiny_app)

# Contrived example:
# Wait until `Date.now()` returns a number that ends in a 5. (0 - 10 seconds)
system.time(
  app$wait_for_js("Math.floor((Date.now() / 1000) % 10) == 5;")
)

## A second example where we run the contents of a JavaScript file
## and use the result to wait for a condition
app$run_js(file = "complicated_file.js")
app$wait_for_js("complicated_condition();")

## End(Not run)

## ------------------------------------------------
## Method `AppDriver$expect_unique_names()`
## ------------------------------------------------

## Not run:
shiny_app <- shinyApp(
  ui = fluidPage(
    # Duplicate input IDs: `"text"`
    textInput("text", "Text 1"),
    textInput("text", "Text 2")
```

```
  ),
  server = function(input, output) {
    # empty
  }
)
# Initial checking for unique names (default behavior)
app <- AppDriver$new(shiny_app, check_names = TRUE)
#> Warning:
#> ! Shiny inputs should have unique HTML id values.
#> i The following HTML id values are not unique:
#> * text
app$stop()

# Manually assert that all names are unique
app <- AppDriver$new(shiny_app, check_names = FALSE)
app$expect_unique_names()
#> Error: `app_check_unique_names(self, private)` threw an unexpected warning.
#> Message: ! Shiny inputs should have unique HTML id values.
#> i The following HTML id values are not unique:
#>   * text
#> Class:   rlang_warning/warning/condition
app$stop()

## End(Not run)

## ------------------------------------------------
## Method `AppDriver$get_dir()`
## ------------------------------------------------

## Not run:
app_path <- system.file("examples/01_hello", package = "shiny")
app <- AppDriver$new(app_path)

identical(app$get_dir(), app_path)
#> [1] TRUE

## End(Not run)

## ------------------------------------------------
## Method `AppDriver$get_url()`
## ------------------------------------------------

## Not run:
app_path <- system.file("examples/01_hello", package = "shiny")
app <- AppDriver$new(app_path)

browseURL(app$get_url())

## End(Not run)

## ------------------------------------------------
## Method `AppDriver$get_window_size()`
## ------------------------------------------------
```

```
## Not run:
app_path <- system.file("examples/01_hello", package = "shiny")
app <- AppDriver$new(app_path)

app$get_window_size()
#> $width
#> [1] 992
#>
#> $height
#> [1] 1323

## End(Not run)

## ------------------------------------------------
## Method `AppDriver$set_window_size()`
## ------------------------------------------------

## Not run:
app_path <- system.file("examples/01_hello", package = "shiny")
# Set init window size
app <- AppDriver$new(app_path, height = 1400, width = 1000)

app$get_window_size()
#> $width
#> [1] 1000
#>
#> $height
#> [1] 1400

# Manually set the window size
app$set_window_size(height = 1080, width = 1920)
app$get_window_size()
#> $width
#> [1] 1920
#>
#> $height
#> [1] 1080

## End(Not run)

## ------------------------------------------------
## Method `AppDriver$get_chromote_session()`
## ------------------------------------------------

## Not run:
app_path <- system.file("examples/01_hello", package = "shiny")
app <- AppDriver$new(app_path)

b <- app$get_chromote_session()
b$Runtime$evaluate("1 + 1")
#> $result
#> $result$type
```

```
#> [1] "number"
#>
#> $result$value
#> [1] 2
#>
#> $result$description
#> [1] "2"

## End(Not run)

## ------------------------------------------------
## Method `AppDriver$get_process()`
## ------------------------------------------------

## Not run:
app_path <- system.file("examples/01_hello", package = "shiny")
app <- AppDriver$new(app_path)

app$get_process()$is_alive()
#> [1] TRUE
app$stop()
app$get_process()$is_alive()
#> [1] FALSE

## End(Not run)

## ------------------------------------------------
## Method `AppDriver$get_variant()`
## ------------------------------------------------

## Not run:
app_path <- system.file("examples/01_hello", package = "shiny")

app <- AppDriver$new(app_path)

app$get_variant()
#> NULL

app <- AppDriver$new(app_path, variant = platform_variant())
app$get_variant()
#> [1] "mac-4.1"

## End(Not run)

## ------------------------------------------------
## Method `AppDriver$get_logs()`
## ------------------------------------------------

## Not run:
app1 <- AppDriver$new(system.file("examples/01_hello", package = "shiny"))

app1$get_logs()
#> {shinytest2} R  info   10:00:28.86 Start AppDriver initialization
```

```
#> {shinytest2} R  info   10:00:28.86 Starting Shiny app
#> {shinytest2} R  info   10:00:29.76 Creating new ChromoteSession
#> {shinytest2} R  info   10:00:30.56 Navigating to Shiny app
#> {shinytest2} R  info   10:00:30.70 Injecting shiny-tracer.js
#> {chromote}   JS info   10:00:30.75 shinytest2; jQuery found
#> {chromote}   JS info   10:00:30.77 shinytest2; Waiting for shiny session to connect
#> {chromote}   JS info   10:00:30.77 shinytest2; Loaded
#> {shinytest2} R  info   10:00:30.77 Waiting for Shiny to become ready
#> {chromote}   JS info   10:00:30.90 shinytest2; Connected
#> {chromote}   JS info   10:00:30.95 shinytest2; shiny:busy
#> {shinytest2} R  info   10:00:30.98 Waiting for Shiny to become idle for 200ms within 15000ms
#> {chromote}   JS info   10:00:30.98 shinytest2; Waiting for Shiny to be stable
#> {chromote}   JS info   10:00:31.37 shinytest2; shiny:idle
#> {chromote}   JS info   10:00:31.38 shinytest2; shiny:value distPlot
#> {chromote}   JS info   10:00:31.57 shinytest2; Shiny has been idle for 200ms
#> {shinytest2} R  info   10:00:31.57 Shiny app started
#> {shiny}      R  stderr ----------- Loading required package: shiny
#> {shiny}      R  stderr ----------- Running application in test mode.
#> {shiny}      R  stderr -----------
#> {shiny}      R  stderr ----------- Listening on http://127.0.0.1:4679


# To capture all websocket traffic, set `options = list(shiny.trace = TRUE)`
app2 <- AppDriver$new(
  system.file("examples/01_hello", package = "shiny"),
  options = list(shiny.trace = TRUE)
)

app2$get_logs()
## (All WebSocket messages have been replaced with `WEBSOCKET_MSG` in example below)
#> {shinytest2} R  info      10:01:57.49 Start AppDriver initialization
#> {shinytest2} R  info      10:01:57.50 Starting Shiny app
#> {shinytest2} R  info      10:01:58.20 Creating new ChromoteSession
#> {shinytest2} R  info      10:01:58.35 Navigating to Shiny app
#> {shinytest2} R  info      10:01:58.47 Injecting shiny-tracer.js
#> {chromote}   JS info      10:01:58.49 shinytest2; jQuery not found
#> {chromote}   JS info      10:01:58.49 shinytest2; Loaded
#> {shinytest2} R  info      10:01:58.50 Waiting for Shiny to become ready
#> {chromote}   JS info      10:01:58.55 shinytest2; jQuery found
#> {chromote}   JS info     10:01:58.55 shinytest2; Waiting for shiny session to connect
#> {chromote}   JS websocket 10:01:58.64 send WEBSOCKET_MSG
#> {chromote}   JS websocket 10:01:58.67 recv WEBSOCKET_MSG
#> {chromote}   JS info      10:01:58.67 shinytest2; Connected
#> {chromote}   JS websocket 10:01:58.71 recv WEBSOCKET_MSG
#> {chromote}   JS websocket 10:01:58.72 recv WEBSOCKET_MSG
#> {chromote}   JS info      10:01:58.72 shinytest2; shiny:busy
#> {chromote}   JS websocket 10:01:58.73 recv WEBSOCKET_MSG
#> {chromote}   JS websocket 10:01:58.73 recv WEBSOCKET_MSG
#> {shinytest2} R  info    10:01:58.75 Waiting for Shiny to become idle for 200ms within 15000ms
#> {chromote}   JS info      10:01:58.75 shinytest2; Waiting for Shiny to be stable
#> {chromote}   JS websocket 10:01:58.81 recv WEBSOCKET_MSG
#> {chromote}   JS websocket 10:01:58.81 recv WEBSOCKET_MSG
#> {chromote}   JS info      10:01:58.81 shinytest2; shiny:idle
```

```
#> {chromote}   JS websocket 10:01:58.82 recv WEBSOCKET_MSG
#> {chromote}   JS info      10:01:58.82 shinytest2; shiny:value distPlot
#> {chromote}   JS info      10:01:59.01 shinytest2; Shiny has been idle for 200ms
#> {shinytest2} R  info      10:01:59.01 Shiny app started
#> {shiny}      R  stderr    ----------- Loading required package: shiny
#> {shiny}      R  stderr    ----------- Running application in test mode.
#> {shiny}      R  stderr    -----------
#> {shiny}      R  stderr    ----------- Listening on http://127.0.0.1:4560
#> {shiny}    R stderr   ----------- SEND {"config":{"workerId":"","sessionId"}}|truncated
#> {shiny}    R stderr   ----------- RECV {"method":"init","data":{"bins":30,}}|truncated
#> {shiny}    R stderr   ----------- SEND {"custom":{"showcase-src":{"srcref":}}}|truncated
#> {shiny}      R  stderr    ----------- SEND {"busy":"busy"}
#> {shiny}    R stderr   ----------- SEND {"custom":{"showcase-src":{"srcref":}}}|truncated
#> {shiny}    R stderr   ----------- SEND {"recalculating":{"name":"distPlot",}}|truncated
#> {shiny}    R stderr   ----------- SEND {"recalculating":{"name":"distPlot",}}|truncated
#> {shiny}      R  stderr    ----------- SEND {"busy":"idle"}
#> {shiny}    R stderr   ----------- SEND {"errors":{},"values":{"distPlot":}}|truncated

# The log that is returned is a `data.frame()`.
log <- app2$get_logs()
tibble::glimpse(log)
#> Rows: 43
#> Columns: 5
#> $ workerid  <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
#> $ timestamp <dttm> 2022-09-19 10:01:57, 2022-09-19 10:01:57, 2022-09-19 10:01:58, 2022...
#> $ location  <chr> "shinytest2", "shinytest2", "shinytest2", "shinytest2", "shinytest2"...
#> $ level    <chr> "info", "info", "info", "info", "info", "info", "info", "info", "inf...
#> $ message  <chr> "Start AppDriver initialization", "Starting Shiny app", "Creating ne...
#> $ workerid  <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
#> $ timestamp <dttm> 2022-03-16 11:09:57, 2022-03-16 11:09:57, 2022-03-16 11:09:...
#> $ location  <chr> "shinytest2", "shinytest2", "shinytest2", "shinytest2", "shi...
#> $ level     <chr> "info", "info", "info", "info", "info", "info", "info", "inf...
#> $ message   <chr> "Start AppDriver initialization", "Starting Shiny app", "Cre...

# It may be filtered to find desired logs
subset(log, level == "websocket")
## (All WebSocket messages have been replaced with `WEBSOCKET_MSG` in example below)
#> {chromote} JS websocket 10:01:58.64 send WEBSOCKET_MSG
#> {chromote} JS websocket 10:01:58.67 recv WEBSOCKET_MSG
#> {chromote} JS websocket 10:01:58.71 recv WEBSOCKET_MSG
#> {chromote} JS websocket 10:01:58.72 recv WEBSOCKET_MSG
#> {chromote} JS websocket 10:01:58.73 recv WEBSOCKET_MSG
#> {chromote} JS websocket 10:01:58.73 recv WEBSOCKET_MSG
#> {chromote} JS websocket 10:01:58.81 recv WEBSOCKET_MSG
#> {chromote} JS websocket 10:01:58.81 recv WEBSOCKET_MSG
#> {chromote} JS websocket 10:01:58.82 recv WEBSOCKET_MSG

## End(Not run)

## ------------------------------------------------
## Method `AppDriver$log_message()`
## ------------------------------------------------
```

```
## Not run:
app_path <- system.file("examples/01_hello", package = "shiny")
app <- AppDriver$new(app_path)

app$log_message("Setting bins to smaller value")
app$set_inputs(bins = 10)
app$get_logs()

## End(Not run)

## ------------------------------------------------
## Method `AppDriver$stop()`
## ------------------------------------------------

## Not run:
rlang::check_installed("reactlog")

library(shiny)
shiny_app <- shinyApp(
  ui = fluidPage(
    actionButton("button", "Stop app and return Reactlog"),
    "Click count:", textOutput("count")
  ),
  server = function(input, output) {
    output$count <- renderText({ input$button })
    observe({
      req(input$button)
      stopApp(shiny::reactlog())
    })
  }
)

app <- AppDriver$new(
  shiny_app,
  # Enable reactlog in background R session
  options = list(shiny.reactlog = TRUE)
)

app$click("button")
rlog <- app$stop()
str(head(rlog, 2))
#> List of 2
#> $ :List of 7
#> ..$ action : chr "define"
#> ..$ reactId: chr "r3"
#> ..$ label  : chr "Theme Counter"
#> ..$ type   : chr "reactiveVal"
#> ..$ value  : chr " num 0"
#> ..$ session: chr "bdc7417f2fc8c84fc05c9518e36fdc44"
#> ..$ time   : num 1.65e+09
#> $ :List of 7
#> ..$ action : chr "define"
#> ..$ reactId: chr "r4"
```

```
#> ..$ label  : chr "output$count"
#> .. ..- attr(*, "srcref")= int [1:6] 7 32 7 45 32 45
#> .. ..- attr(*, "srcfile")= chr ""
#> ..$ type   : chr "observer"
#> ..$ value  : chr " NULL"
#> ..$ session: chr "bdc7417f2fc8c84fc05c9518e36fdc44"
#> ..$ time   : num 1.65e+09

## End(Not run)
```

`compare_screenshot_threshold` *Compare screenshots given threshold value*

Description

chromote can sometimes produce screenshot images with non-deterministic (yet close) color values. This can happen in locations such as rounded corners of `div`s or `textarea`s.

Usage

```
compare_screenshot_threshold(
  old,
  new,
  ...,
  threshold = getOption("shinytest2.compare_screenshot.threshold", NULL),
  kernel_size = getOption("shinytest2.compare_screenshot.kernel_size", 5),
  quiet = FALSE
)

screenshot_max_difference(
  old,
  new = missing_arg(),
  ...,
  kernel_size = getOption("shinytest2.compare_screenshot.kernel_size", 5)
)
```

Arguments

`old`	Current screenshot file path
`new`	New screenshot file path
`...`	Must be empty. Allows for parameter expansion.
`threshold`	The threshold for maximum allowed image difference (see below for details). The default is `NULL` or can be set globally via the `shinytest2.compare_screenshot.threshold` option. If the value of `threshold` is `NULL`, `compare_screenshot_threshold()` will act like `testthat::compare_file_binary`. However, if `threshold` is a positive

number, it will be compared against the largest convolution value found if the two images fail a testthat::compare_file_binary comparison. The max value that can be found is 4 * kernel_size ^ 2.

Threshold values values below 5 help deter false-positive screenshot comparisons (such as inconsistent rounded corners). Larger values in the 10s and 100s will help find *real* changes. However, not all values are one size fits all and you will need to play with a threshold that fits your needs.

To find the current difference between two images, use screenshot_max_difference().

kernel_size — The kernel_size represents the height and width of the convolution kernel applied to the matrix of pixel differences. This integer-like value should be relatively small, such as 5. The default value can be set via the shinytest2.compare_screenshot.kernel_s option.

quiet — If FALSE and the value is larger than threshold, then a message is printed to the console. This is helpful when getting a failing image and being informed about how different the new image is from the old image.

Details

These differences make comparing screenshots impractical using traditional expectation methods as false-positives are produced often over time. To mitigate this, we can use a *fuzzy matching* algorithm that can tolerate small regional differences throughout the image. If the local changes found are larger than the threshold, then the images are determined to be different. Both the screenshot difference threshold and the size of the kernel (kernel_size) can be set to tune the false positive rate.

Functions

- compare_screenshot_threshold(): Compares two images and allows for a threshold difference of *so many* units in each RGBA color channel.

  It is suggested to use this method with AppDriver$expect_screenshot(threshold=, kernel_size=) to make expectations on screenshots given particular threshold and kernel_size values.

- screenshot_max_difference(): Finds the difference between two screenshots.

  This value can be used in compare_screenshot_threshold(threshold=). It is recommended that the value used for compare_screenshot_threshold(threshold=) is larger than the immediate max difference found. This allows for random fluctuations when rounding sub pixels.

  If new is missing, it will use the file value of old (FILE.png) and default to FILE.new.png

Algorithm for the difference between two screenshots

1. First the two images are compared using testthat::compare_file_binary(). If the files are identical, return TRUE that the screenshot images are the same.
2. If threshold is NULL, return FALSE as the convolution will not occur.
3. Prepare the screenshot difference matrix by reading the RGBA channels of each image and find their respective absolute differences
4. Sum the screenshot difference matrix channels at each pixel location

5. Perform a convolution using a small square kernel matrix that is `kernel_size` big and filled with 1s.
6. Find the largest value in the resulting convolution matrix.
7. If this max convolution value is larger than `threshold`, return `FALSE`, images are different.
8. Otherwise, return `TRUE`, images are the same.

Examples

```
img_folder <- system.file("example/imgs/", package = "shinytest2")
slider_old <- fs::path(img_folder, "slider-old.png")
slider_new <- fs::path(img_folder, "slider-new.png")

# Can you see the differences between these two images?
showimage::show_image(slider_old)
showimage::show_image(slider_new)

# You might have caught the difference between the two images!
slider_diff <- fs::path(img_folder, "slider-diff.png")
showimage::show_image(slider_diff)

# Let's find the difference between the two images
screenshot_max_difference(slider_old, slider_new)
# ~ 28

# Using different threshold values...
compare_screenshot_threshold(slider_old, slider_new, threshold = NULL)
#> FALSE # Images are not identical
compare_screenshot_threshold(slider_old, slider_new, threshold = 25)
#> FALSE # Images are more different than 25 units
compare_screenshot_threshold(slider_old, slider_new, threshold = 30)
#> TRUE # Images are not as different as 30 units

#########################

# Now let's look at two fairly similar images
bookmark_old <- fs::path(img_folder, "bookmark-old.png")
bookmark_new <- fs::path(img_folder, "bookmark-new.png")

# Can you see the difference between these two images?
# (Hint: Focused corners)
showimage::show_image(bookmark_old)
showimage::show_image(bookmark_new)

# Can you find the red pixels showing the differences?
# Hint: Look in the corners of the focused text
bookmark_diff <- fs::path(img_folder, "bookmark-diff.png")
showimage::show_image(bookmark_diff)

# Let's find the difference between the two images
screenshot_max_difference(bookmark_old, bookmark_new)
# ~ 0.25
```

```
# Using different threshold values...
compare_screenshot_threshold(bookmark_old, bookmark_new, threshold = NULL)
#> FALSE # Images are not identical
compare_screenshot_threshold(bookmark_old, bookmark_new, threshold = 5)
#> TRUE # Images are not as different than 5 units
```

`local_app_support` *Attach the Shiny application's support environment*

Description

Executes all `./R` files and `global.R` from a Shiny application directory into an environment that is attached appropriately. This is useful when you need access to helper functions, modules, or values created in the `./R` folder for testing purposes.

Usage

```
local_app_support(app_dir, envir = rlang::caller_env())

with_app_support(app_dir, expr, envir = rlang::caller_env())

load_app_support(app_dir, envir = rlang::caller_env())
```

Arguments

`app_dir`	The base directory for the Shiny application.
`envir`	The environment in which the App support should be made available.
`expr`	An expression to evaluate within the support environment.

Functions

- `local_app_support()`: Temporarily attach the Shiny application's support environment into the current environment.
- `with_app_support()`: For the provided `expr`, attach the Shiny application's support environment into the current environment.
- `load_app_support()`: Loads all support files into the current environment. No cleanup actions are ever performed.

When to use these functions

For inline apps (app objects passed directly to AppDriver): You do **NOT** need these functions. Since your test file already has access to your package functions (via `library(yourpackage)` or `devtools::load_all()`), you can reference them directly when creating the app object.

For apps with separate support files: Use these functions when your app is stored in a directory (e.g., `inst/myapps/app1/`) and has supporting R files in an `app.R` / `server.R`, `global.R, and ./R‘` subfolder that you need to access in your tests.

Choosing which function to use

- `local_app_support()`: For R packages or when you want automatic cleanup. The support environment is attached for the duration of the current environment, then automatically removed. This prevents pollution of the testing environment across multiple tests that test multiple apps.
- `with_app_support()`: Similar to `local_app_support()`, but uses an explicit code block. The support environment is only available within the provided expression and is cleaned up immediately after.
- `load_app_support()`: For standalone Shiny apps (non-package). Loads support files into the testing environment permanently with no automatic cleanup. Best used in `setup-shinytest2.R` for non-package apps.

Benefits

- **Isolation**: `local_app_support()` and `with_app_support()` prevent helper functions from one app test from interfering with another.
- **Access to app internals**: Test helper functions, modules, and utilities defined in your app's `./R` folder without manually sourcing files.
- **Package testing**: Enables testing apps within packages where the app's support files are separate from the package's exported functions.

Examples

```
## Not run:
# Example 1: Basic usage with local_app_support()
# File structure:
#   ./tests/testthat/apps/myapp/R/utils.R  (contains: n <- 42)
#   ./tests/testthat/test-utils.R

test_that("Can access support environment", {
  expect_false(exists("n"))
  shinytest2::local_app_support(test_path("apps/myapp"))
  expect_equal(n, 42)
})
# After test completes, n is automatically removed

# Example 2: Using with_app_support() for explicit scoping
test_that("Can access support environment", {
  expect_false(exists("n"))
  shinytest2::with_app_support(test_path("apps/myapp"), {
    expect_equal(n, 42)
  })
  expect_false(exists("n"))  # n is no longer available
})

# Example 3: Testing an app with its support files
# File structure:
#   ./inst/myapps/app1/app.R
#   ./inst/myapps/app1/R/modules.R  (contains: my_module_ui <- function() {...})
```

```
#   ./tests/testthat/test-app1.R

test_that("app1 loads with module support", {
  expect_false(exists("my_module_ui"))

  app_dir <- system.file("myapps/app1", package = "mypkg")
  local_app_support(app_dir)

  # Now we can access module functions from app1/R/
  expect_true(exists("my_module_ui"))

  # And test the app
  app <- AppDriver$new(app_dir)
  app$expect_values()
})

## End(Not run)
```

`migrate_from_shinytest` *Migrate* **shinytest** *tests*

Description

This function will migrate standard shinytest test files to the new **shinytest2** + **testthat** ed 3 snapshot format.

Usage

```
migrate_from_shinytest(
  app_dir,
  ...,
  clean = TRUE,
  include_expect_screenshot = missing_arg(),
  quiet = FALSE
)
```

Arguments

app_dir	Directory containing the Shiny application or Shiny Rmd file
...	Must be empty. Allows for parameter expansion.
clean	If TRUE, then the shinytest test directory and runner will be deleted after the migration to use **shinytest2**.
include_expect_screenshot	If TRUE, ShinyDriver$snapshot() will turn into both AppDriver$expect_values() and AppDriver$expect_screenshot(). If FALSE, ShinyDriver$snapshot() will only turn into AppDriver$expect_values(). If missing, include_expect_screenshot will behave as FALSE if shinytest::testApp(compareImages = FALSE) or ShinyDriver$snapshotIni = FALSE) is called.

quiet	Logical that determines if migration information and steps should be printed to the console.

Details

shinytest file contents will be traversed and converted to the new **shinytest2** format. If the **shinytest** code can not be directly seen in the code, then it will not be converted.

Value

Invisible TRUE

---

platform_variant *Platform specific variant*

Description

Returns a string to be used within **testthat**'s' snapshot testing. Currently, the Operating System and R Version (major, minor, no patch version) are returned.

Usage

```
platform_variant(..., os_name = TRUE, r_version = TRUE)
```

Arguments

...	Must be empty. Allows for parameter expansion.
os_name	if TRUE, include the OS name in the output
r_version	if TRUE, include the major and minor version of the R version, no patch version

Details

If more information is needed in the future to distinguish standard testing environments, this function will be updated accordingly.

See Also

testthat::test_dir()

record_test *Launch test event recorder for a Shiny app*

Description

Once a recording is completed, it will create or append a new **shinytest2** test to the **testthat** `test_file`.

Usage

```
record_test(
  app = ".",
  ...,
  name = NULL,
  seed = NULL,
  load_timeout = NULL,
  shiny_args = list(),
  test_file = NULL,
  open_test_file = rlang::is_interactive(),
  allow_no_input_binding = NULL,
  record_screen_size = TRUE,
  run_test = TRUE,
  record_in_package = TRUE
)
```

Arguments

app	A `AppDriver` object, path to a Shiny application, a Shiny application object, or a function that returns a Shiny application.
...	Must be empty. Allows for parameter expansion.
name	Name provided to `AppDriver`. This value should be unique between all tests within a test file. If it is not unique, different expect methods may overwrite each other.
seed	A random seed to set before running the app. This seed will also be used in the test script.
load_timeout	Maximum time to wait for the Shiny application to load, in milliseconds. If a value is provided, it will be saved in the test script.
shiny_args	A list of options to pass to `runApp()`. If a value is provided, it will be saved in the test script.
test_file	Base file name of the **testthat** test file. If `NULL`, a default name will be used. If recording within a package, the test file will be saved to the package's `tests/testthat/` directory. If not within a package, the test file will be saved to the app's `tests/testthat/` directory. If within a package, the default name is `test-app-<appdir>.R`, where `<appdir>` is the name of the app directory. If not within a package, the default name is `test-shinytest2.R`.

`open_test_file` If `TRUE`, the test file will be opened in an editor via `file.edit()` before executing.

`allow_no_input_binding`
This value controls if events without input bindings are recorded.

- If `TRUE`, events without input bindings are recorded.
- If `FALSE`, events without input bindings are not recorded.
- If `NULL` (default), if an updated input does not have a corresponding input, a modal dialog will be shown asking if unbound input events should be recorded.

See `AppDriver$set_inputs()` for more information.

`record_screen_size`
If `TRUE`, the screen size will be recorded when initialized and changed.

`run_test` If `TRUE`, `test_file` will be executed after saving the recording.

`record_in_package`
If `TRUE` and if the current working directory is within a package, the test file will be saved to the package's `tests/testthat/` directory. If `FALSE`, the test file will be saved to the app's `tests/testthat/` directory.

Uploading files

Files that are uploaded to your Shiny app must be located somewhere within the `tests/testthat` subdirectory or available via `system.file()`.

Files that are uploaded during recording that do not have a valid path will have a warning inserted into the code. Please fix the file path by moving the file to the `tests/testthat` subdirectory or by using `system.file()`. After fixing the path, remove the line of warning code.

See Also

`test_app()`

Examples

```
## Not run:
record_test("path/to/app")

## End(Not run)
```

`test_app` *Test Shiny applications with* **testthat**

Description

This is a helper method that wraps around `testthat::test_dir()` to test your Shiny application or Shiny runtime document. This is similar to how `testthat::test_check()` tests your R package but for your app.

Usage

```
test_app(
  app_dir = missing_arg(),
  ...,
  name = missing_arg(),
  reporter = testthat::get_reporter(),
  stop_on_failure = missing_arg(),
  check_setup = deprecated(),
  quiet = FALSE
)
```

Arguments

app_dir	The base directory for the Shiny application. • If app_dir is missing and test_app() is called within the ./tests/testthat.R file, the parent directory ("../") is used. • Otherwise, the default path of "." is used.
...	Parameters passed to testthat::test_dir()
name	Name to display in the middle of the test name. This value is only used when calling test_app() inside of **testthat** test. The final testing context will have the format of "{test_context} - {name} - {app_test_context}".
reporter	Reporter to pass through to testthat::test_dir().
stop_on_failure	If missing, the default value of TRUE will be used. However, if missing and currently testing, FALSE will be used to seamlessly integrate the app reporter to reporter.
check_setup	Deprecated. Parameter ignored.
quiet	If TRUE, suppresses deprecation warnings when called within **testthat** tests.

Details

Example usage:

```
## Interactive usage
# Test Shiny app in current working directory
shinytest2::test_app()

# Test Shiny app in another directory
path_to_app <- "path/to/app"
shinytest2::test_app(path_to_app)

## File: ./tests/testthat.R
# Will find Shiny app in "../"
shinytest2::test_app()

## File: ./tests/testthat/test-shinytest2.R
```

```
# Test a shiny application within your own {testthat} code
test_that("Testing a Shiny app in a package", {
  app <- shinytest2::AppDriver$new(path_to_app)
  # Perform tests with `app`...
})
```

When testing within a package, it is recommended to not call `test_app()`, but instead test your applications within your own **testthat** tests. This allows for more flexibility and control over how your applications are tested while your current package's testthat infrastructure. See the Use Package vignette for more details.

Uploading files

When testing an application, all non-temp files that are uploaded should be located in the `./tests/testthat` directory. This allows for tests to be more portable and self contained.

When recording a test with `record_test()`, for every uploaded file that is located outside of `./tests/testthat`, a warning will be thrown. Once the file path has be fixed, you may remove the warning statement.

Different ways to test

`test_app()` is an opinionated testing function that will only execute **testthat** tests in the `./tests/testthat` folder. If (for some rare reason) you have other non-**testthat** tests to execute, you can call `shiny::runTests()`. This method will generically run all test runners and their associated tests.

```
# Execute a single Shiny app's {testthat} file such as `./tests/testthat/test-shinytest2.R`
test_app(filter = "shinytest2")

# Execute all {testthat} tests
test_app()

# Execute all tests for all test runners
shiny::runTests()
```

See Also

- `record_test()` to create tests to record against your Shiny application.
- `testthat::snapshot_review()` and `testthat::snapshot_accept()` if you want to compare or update snapshots after testing.
- `local_app_support()` / `with_app_support()` to load the Shiny application's helper files into respective environments. These methods are useful for within package testing as they have fine tune control over when the support environment is loaded.
- `load_app_support()` to load the Shiny application's helper files into the calling environment. This method is useful for non-package based Shiny applications where the support environment should be available in every test file.

use_shinytest2 *Use* **shinytest2** *with your Shiny application*

Description

Use **shinytest2** with your Shiny application

Usage

```
use_shinytest2(
  app_dir = ".",
  runner = missing_arg(),
  setup = missing_arg(),
  ignore = missing_arg(),
  package = missing_arg(),
  ...,
  quiet = FALSE,
  overwrite = FALSE
)

use_shinytest2_test(
  app_dir = ".",
  open = rlang::is_interactive(),
  quiet = FALSE,
  overwrite = FALSE
)
```

Arguments

app_dir	The base directory for the Shiny application
runner	If TRUE, creates a **shinytest2** test runner at ./tests/testthat.R
setup	If TRUE, creates a setup file called ./tests/testthat/setup-shinytest2.R containing a call to load_app_support(). If you would like fine grain control over when the environment is loaded, please look at local_app_support() and with_app_support().
ignore	If TRUE, adds entries to .Rbuildignore and .gitignore to ignore new debug screenshots. (*_.new.png)
package	If TRUE, adds **shinytest2** to Suggests in the DESCRIPTION file.
...	Must be empty. Allows for parameter expansion.
quiet	If TRUE, console output will be suppressed.
overwrite	If TRUE, the test file or test runner will be overwritten.
open	If TRUE, the test file will be opened in an editor via file.edit() after saving.

Functions

- `use_shinytest2()`: This **usethis**-style method initializes many different useful features when using **shinytest2**:
  - `runner`: Creates a **shinytest2** test runner at `./tests/testthat.R`. This file will contain a call to `test_app()`.
  - `setup`: Creates `./tests/testthat/setup-shinytest2.R` to add your Shiny `./R` objects and functions into the testing environment. This file will run before testing begins.
  - `ignore`: Add an entry to `./Rbuildignore` (if it exists) and `.gitignore` to ignore new debug screenshots. (`*_.new.png`)
  - `package`: Adds `shinytest` to the `Suggests` packages in the `DESCRIPTION` file (if it exists).

  When all values are missing and currently in a package working directory, the defaults are all TRUE. When the current working directory is a package root directory, `runner`/`setup` are `FALSE` and `ignore`/`package` are `TRUE`.

  If any of these values are *not* missing, the remaining missing values will be set to `FALSE`. This allows `use_shinytest2()` to add more flags in future versions without opting into all changes inadvertently.
- `use_shinytest2_test()`: Creates a test file called `./tests/testthat/test-shinytest2.R`. By default, this file's template test will initialize your Shiny application and expect the initial values.

  This method will also set up a test runner if it does not exist.

Examples

```
# Set up shinytest2 testing configs
## Not run: use_shinytest2()
# Set up a shinytest2 test
## Not run: use_shinytest2_test()
```

Index

AppDriver, 2, *54*, *60, 61*

base::options(), *7*

callr::r_bg(), *30*
callr::r_process, *31*
chromote::ChromoteSession, *7, 8*, *10*, *21*, *24*, *30*
compare_screenshot_threshold, 53
compare_screenshot_threshold(), *21, 22*

Deprecated, *62*

file.edit(), *61*, *64*

load_app_support *(*local_app_support*)*, 56
load_app_support(), *63, 64*
local_app_support, 56
local_app_support(), *63, 64*

migrate_from_shinytest, 58

platform_variant, 59
platform_variant(), *7*, *36*

record_test, 60
record_test(), *63*

screenshot_max_difference *(*compare_screenshot_threshold*)*, 53
shiny::downloadButton(), *14, 15*
shiny::downloadLink(), *14, 15*
shiny::runApp(), *7*
shiny::runTests(), *63*
shiny::shinyOptions(), *7*
shiny::stopApp(), *30*, *34*

test_app, 61
test_app(), *61*, *65*
testthat::compare_file_binary, *21*, *53, 54*
testthat::compare_file_binary(), *22*, *54*
testthat::expect_snapshot_file(), *4*, *14*
testthat::snapshot_accept(), *63*
testthat::snapshot_review(), *5*, *63*
testthat::test_check(), *61*
testthat::test_dir(), *59*, *61, 62*

unlist(), *17*
use_shinytest2, 64
use_shinytest2_test *(*use_shinytest2*)*, 64
use_shinytest2_test(), *36*

with_app_support *(*local_app_support*)*, 56
with_app_support(), *63, 64*