Package ‘shinytest’

May 30, 2024

Title Test Shiny Apps
Version 1.6.1
Description Please see the 'shinytest' to 'shinytest2' migration guide at <https://rstudio.github.io/shinytest2/articles/z-migration.html>.
License MIT + file LICENSE
URL https://github.com/rstudio/shinytest
BugReports https://github.com/rstudio/shinytest/issues
Imports assertthat, callr (>= 2.0.3), crayon, debugme, digest, htmlwidgets, httpuv, httr, jsonlite, parsedate, pingr, R6, rematch, rlang, rstudioapi (>= 0.8.0.9002), shiny (>= 1.3.2), testthat (>= 1.0.0), utils, webdriver (>= 1.0.6), withr
Suggests flexdashboard, globals, rmarkdown
Encoding UTF-8
RoxygenNote 7.3.1
SystemRequirements PhantomJS (http://phantomjs.org/)
NeedsCompilation no
Author Winston Chang [aut, cre],
Gábor Csárdi [aut],
Hadley Wickham [aut],
Posit Software, PBC [cph, fnd],
Ascent Digital Services [cph, ccp]
Maintainer Winston Chang <winston@posit.co>
Repository CRAN
Date/Publication 2024-05-30 19:10:02 UTC

R topics documented:

dependenciesInstalled .............................................. 2
effect_pass ............................................................ 2
migrateShinytestDir ..................................................... 3
dependenciesInstalled  

**Description**

dependenciesInstalled() that all the required system dependency, PhantomJS, is installed, and installDependencies() installs it if needed. For more information about where PhantomJS will be installed see webdriver::install_phantomjs().

**Usage**

dependenciesInstalled()

installDependencies()

**Value**

TRUE when all dependencies are fulfilled; otherwise, FALSE.

---

expect_pass  

**Description**

This returns an testthat expectation object.

**Usage**

expect_pass(object, info = NULL)

**Arguments**

- **object**: The results returned by testApp().
- **info**: Extra information to be included in the message (useful when writing tests in loops).
migrateShinytestDir

Examples

```r
## Not run:
expect_pass(testApp("path/to/app/"))

## End(Not run)
```

migrateShinytestDir  Migrate legacy `shinytest` files to new test directory structure

Description

This function migrates the old-style directory structure used by `shinytest` (versions 1.3.1 and below) to new test directory structure used in `shinytest` 1.4.0 and above.

Usage

```r
migrateShinytestDir(appdir, dryrun = FALSE)
```

Arguments

- **appdir**: A directory containing a Shiny application.
- **dryrun**: If TRUE, print out the changes that would be made, but don’t actually do them.

Details

Before `shinytest` 1.4.0, the shinytest scripts and results were put in a subdirectory of the application named `tests/`. As of `shinytest` 1.4.0, the tests are put in `tests/shinytest/`, so that it works with the `runTests()` function `shiny` package (added in `shiny` 1.5.0).

With `shinytest` 1.3.1 and below, the tests/ subdirectory of the application was used specifically for `shinytest`, and could not be used for other types of tests. So the directory structure would look like this:

```
appdir/  
 | `-- tests
  |    `-- mytest.R
```

In Shiny 1.5.0, the `shiny::runTests()` function was added, and it will run test scripts tests/ subdirectory of the application. This makes it possible to use other testing systems in addition to shinytest. `shinytest` 1.4.0 is designed to work with this new directory structure. The directory structure looks something like this:

```
appdir/  
 | `-- R
 |    `-- tests
    |    `-- shinytest.R
    |    `-- shinytest
```
This allows for tests using the **shinytest** package as well as other testing tools, such as the \texttt{shiny::testServer()} function, which can be used for testing module and server logic, and for unit tests of functions in an R/ subdirectory.

In **shinytest** 1.4.0 and above, it defaults to creating the new directory structure.

---

### osName

Get the name of the OS

**Description**

Returns the name of the current OS. This can be useful for the suffix when running \texttt{testApp()}.

**Usage**

\begin{verbatim}
osName()
\end{verbatim}

---

### recordTest

Launch test event recorder for a Shiny app

**Description**

Launch test event recorder for a Shiny app

**Usage**

\begin{verbatim}
recordTest(  
  app = ".",
  save_dir = NULL,
  load_mode = FALSE,
  seed = NULL,
  loadTimeout = 10000,
  debug = "shiny_console",
  shinyOptions = list()
)
\end{verbatim}
Arguments

- **app**
  A `ShinyDriver()` object, or path to a Shiny application.

- **save_dir**
  A directory to save stuff.

- **load_mode**
  A boolean that determines whether or not the resulting test script should be appropriate for load testing.

- **seed**
  A random seed to set before running the app. This seed will also be used in the test script.

- **loadTimeout**
  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.

- **debug**
  start the underlying `ShinyDriver()` in debug mode and print those debug logs to the R console once recording is finished. The default, 'shiny_console', captures and prints R console output from the recorded R shiny process. Any value that the debug argument in `ShinyDriver()` accepts may be used (e.g., 'none' may be used to completely suppress the driver logs).

- **shinyOptions**
  A list of options to pass to `runApp()`. If a value is provided, it will be saved in the test script.

---

**ShinyDriver**

*Remote control a Shiny app running in a headless browser*

---

**Description**

This class starts a Shiny app in a new R session, along with a phantom.js 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**

- **Public methods:**

  - `ShinyDriver$new()`
  - `ShinyDriver$finalize()`
  - `ShinyDriver$stop()`
  - `ShinyDriver$getValue()`
  - `ShinyDriver$setValue()`
  - `ShinyDriver$click()`
  - `ShinyDriver$getAllValues()`
  - `ShinyDriver$sendKeys()`
  - `ShinyDriver$setWindowSize()`
  - `ShinyDriver$getWindowSize()`
  - `ShinyDriver$getDebugLog()`
  - `ShinyDriver$enableDebugLogMessages()`
  - `ShinyDriver$logEvent()`
• ShinyDriver$getEventLog()
• ShinyDriver$getURL()
• ShinyDriver$getTitle()
• ShinyDriver$getSource()
• ShinyDriver$goBack()
• ShinyDriver$refresh()
• ShinyDriver$takeScreenshot()
• ShinyDriver$findElement()
• ShinyDriver$findElements()
• ShinyDriver$waitFor()
• ShinyDriver$waitForShiny()
• ShinyDriver$waitForValue()
• ShinyDriver$listWidgets()
• ShinyDriver$checkUniqueWidgetNames()
• ShinyDriver$executeScript()
• ShinyDriver$executeScriptAsync()
• ShinyDriver$findWidget()
• ShinyDriver$expectUpdate()
• ShinyDriver$setInputs()
• ShinyDriver$uploadFile()
• ShinyDriver$snapshotInit()
• ShinyDriver$snapshot()
• ShinyDriver$snapshotCompare()
• ShinyDriver$snapshotDownload()
• ShinyDriver$getAppDir()
• ShinyDriver$getAppFilename()
• ShinyDriver$getTestsDir()
• ShinyDriver$getRelativePathToApp()
• ShinyDriver$getSnapshotDir()
• ShinyDriver$isRmd()
• ShinyDriver$clone()

**Method** `new()`:

**Usage:**

```r
ShinyDriver$new(
  path = ".", 
  loadTimeout = NULL, 
  checkNames = TRUE, 
  debug = c("none", "all", shinytest::ShinyDriver$debugLogTypes), 
  phantomTimeout = 5000, 
  seed = NULL, 
  cleanLogs = TRUE, 
  shinyOptions = list(), 
  renderArgs = NULL, 
)```

---

**Usage:**

```r
ShinyDriver$new(
  path = ".", 
  loadTimeout = NULL, 
  checkNames = TRUE, 
  debug = c("none", "all", shinytest::ShinyDriver$debugLogTypes), 
  phantomTimeout = 5000, 
  seed = NULL, 
  cleanLogs = TRUE, 
  shinyOptions = list(), 
  renderArgs = NULL, 
)```

---

**Usage:**

```r
ShinyDriver$new(
  path = ".", 
  loadTimeout = NULL, 
  checkNames = TRUE, 
  debug = c("none", "all", shinytest::ShinyDriver$debugLogTypes), 
  phantomTimeout = 5000, 
  seed = NULL, 
  cleanLogs = TRUE, 
  shinyOptions = list(), 
  renderArgs = NULL, 
)```

---

**Usage:**

```r
ShinyDriver$new(
  path = ".", 
  loadTimeout = NULL, 
  checkNames = TRUE, 
  debug = c("none", "all", shinytest::ShinyDriver$debugLogTypes), 
  phantomTimeout = 5000, 
  seed = NULL, 
  cleanLogs = TRUE, 
  shinyOptions = list(), 
  renderArgs = NULL, 
)```
options = list()
)

Arguments:
path Path to a directory containing a Shiny app, i.e. a single app.R file or a server.R-ui.R pair.
loadTimeout How long to wait for the app to load, in ms. This includes the time to start R. Defaults to 5s when running locally and 10s when running on CI.
checkNames Check if widget names are unique?
depbug Start the app in debugging mode? In debugging mode debug messages are printed to the console.
phantomTimeout How long to wait when connecting to phantomJS process, in ms
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.
cleanLogs Whether to remove the stdout and stderr logs when the Shiny process object is garbage collected.
shinyOptions A list of options to pass to shiny::runApp()
renderArgs Passed to rmarkdown::run() for interactive .Rmds.
options A list of base::options() to set in the driver’s child process.

Method finalize(): Stop app and clean up logs.
Usage:
ShinyDriver$finalize()

Method stop(): Stop the app, the terminate external R process that runs the app and the phantomjs instance.
Usage:
ShinyDriver$stop()

Method getValue(): Finds a widget and queries its value. See the getValue() method of Widget for more details.
Usage:
ShinyDriver$getValue(name, iotype = c("auto", "input", "output"))

Arguments:
name Name of a shiny widget.
iotype Type of the Shiny widget. Usually shinytest finds the widgets by their name, so this is only needed if you use the same name for an input and output widget.

Method setValue(): Finds a widget and sets its value. It’s a shortcut for findElement() plus setValue(); see the Widget documentation for more details.
Usage:
ShinyDriver$setValue(name, value, iotype = c("auto", "input", "output"))

Arguments:
name Name of a shiny widget.
value New value.
iotype  Type of the Shiny widget. Usually shinytest finds the widgets by their name, so this is only needed if you use the same name for an input and output widget.

*Method* click(): Find a widget and click it. It's a shortcut for findElement() plus click(); see the Widget documentation for more details.

*Usage:*
ShinyDriver$click(name, iotype = c("auto", "input", "output"))

*Arguments:*
- name  Name of a shiny widget.
- iotype  Type of the Shiny widget. Usually shinytest finds the widgets by their name, so this is only needed if you use the same name for an input and output widget.

*Method* getAllValues(): Returns a named list of all inputs, outputs, and export values.

*Usage:*
ShinyDriver$getAllValues(input = TRUE, output = TRUE, export = TRUE)

*Arguments:*
- input, output, export  Either TRUE to return all input/output/exported values, or a character vector of specific controls.

*Method* sendKeys(): Sends the specified keys to specific HTML element. Shortcut for findWidget() plus sendKeys().

*Usage:*
ShinyDriver$sendKeys(name, keys)

*Arguments:*
- name  Name of a shiny widget.
- keys  Keys to send to the widget or the app. See webdriver::key for how to specific special keys.

*Returns:  Self, invisibly.*

*Method* setWindowSize(): Sets size of the browser window.

*Usage:*
ShinyDriver$setWindowSize(width, height)

*Arguments:*
- width, height  Height and width of browser, in pixels.

*Returns:  Self, invisibly.*

*Method* getWindowSize(): Get current size of the browser window, as list of integer scalars named width and height.

*Usage:*
ShinyDriver$getWindowSize()

*Method* getDebugLog(): Query one or more of the debug logs.

*Usage:*

ShinyDriver$getDebugLog(type = c("all", ShinyDriver$debugLogTypes))

Arguments:
type Log type: "all", "shiny_console", "browser", or "shinytest".

Method enableDebugLogMessages(): Enable/disable debugging messages
Usage:
ShinyDriver$enableDebugLogMessages(enable = TRUE)
Arguments:
enable New value.

Method logEvent(): Add event to log.
Usage:
ShinyDriver$logEvent(event, ...)
Arguments:
event Event name
... Addition data to store for event

Method getEventLog(): Retrieve event log.
Usage:
ShinyDriver$getEventLog()

Method getUrl(): Get current url
Usage:
ShinyDriver$getUrl()

Method getTitle(): Get page title
Usage:
ShinyDriver$getTitle()

Method getSource(): Get complete source of current page.
Usage:
ShinyDriver$getSource()

Method goBack(): Return to previous page
Usage:
ShinyDriver$goBack()
Returns: Self, invisibly.

Method refresh(): Refresh the browser
Usage:
ShinyDriver$refresh()
Returns: Self, invisibly.
**Method** `takeScreenshot()`: Takes a screenshot of the current page and writes it to a PNG file or shows it on the current graphics device.

*Usage:*

```r
ShinyDriver$takeScreenshot(file = NULL, id = NULL, parent = FALSE)
```

*Arguments:*

- **file** File name to save the screenshot to. If NULL, then it will be shown on the R graphics device.
- **id** If not-NULL, will take a screenshot of element with this id.
- **parent** If TRUE, will take screenshot of parent of id; this is useful if you also want to capture the label attached to a Shiny control.

*Returns:* Self, invisibly.

**Method** `findElement()`: Find an HTML element on the page, using a CSS selector, XPath expression, or link text (for `<a>` tags). If multiple elements are matched, only the first is returned.

*Usage:*

```r
ShinyDriver$findElement(
  css = NULL,
  linkText = NULL,
  partialLinkText = NULL,
  xpath = NULL
)
```

*Arguments:*

- **css** CSS selector to find an HTML element.
- **linkText** Find `<a>` HTML elements based on exact `innerText`
- **partialLinkText** Find `<a>` HTML elements based on partial `innerText`
- **xpath** Find HTML elements using XPath expressions.

*Returns:* A `webdriver::Element`.

**Method** `findElements()`: Find all elements matching CSS selection, xpath, or link text.

*Usage:*

```r
ShinyDriver$findElements(
  css = NULL,
  linkText = NULL,
  partialLinkText = NULL,
  xpath = NULL
)
```

*Arguments:*

- **css** CSS selector to find an HTML element.
- **linkText** Find `<a>` HTML elements based on exact `innerText`
- **partialLinkText** Find `<a>` HTML elements based on partial `innerText`
- **xpath** Find HTML elements using XPath expressions.

*Returns:* A list of `webdriver::Elements`. 
Method waitFor(): Waits until a JavaScript expression evaluates to true or the timeout is exceeded.

Usage:
ShinyDriver$waitFor(expr, checkInterval = 100, timeout = 3000)

Arguments:
expr  A string containing JavaScript code. Will wait until the condition returns true.
checkInterval  How often to check for the condition, in ms.
timeout  Amount of time to wait before giving up (milliseconds).

Returns: TRUE if expression evaluates to true without error, before timeout. Otherwise returns NA.

Method waitForShiny(): Waits until Shiny is not busy, i.e. the reactive graph has finished updating. This is useful, for example, if you’ve resized the window with setWindowSize() and want to make sure all plot redrawing is complete before take a screenshot.

Usage:
ShinyDriver$waitForShiny()

Returns: TRUE if done before before timeout; NA otherwise.

Method waitForValue(): Waits until the input or output with name name is not one of ignored values, or the timeout is reached. This function can be useful in helping determine if an application has initialized or finished processing a complex reactive situation.

Usage:
ShinyDriver$waitForValue(
  name,  
  ignore = list(NULL, ""),
  iotype = c("input", "output", "export"),
  timeout = 10000,
  checkInterval = 400
)

Arguments:
name  Name of a shiny widget.
ignore  List of possible values to ignore when checking for updates.
iotype  Type of the Shiny widget. Usually shinytest finds the widgets by their name, so this is only needed if you use the same name for an input and output widget.
timeout  Amount of time to wait before giving up (milliseconds).
checkInterval  How often to check for the condition, in ms.

Method listWidgets(): Lists the names of all input and output widgets

Usage:
ShinyDriver$listWidgets()

Returns: A list of two character vectors, named input and output.

Method checkUniqueWidgetNames(): Check if Shiny widget names are unique.
Usage:
ShinyDriver$checkUniqueWidgetNames()

Method executeScript(): Execute JS code
Usage:
ShinyDriver$executeScript(script, ...)
Arguments:
script JS to execute.
... Additional arguments to script.
Returns: Self, invisibly.

Method executeScriptAsync(): Execute JS code asynchronously.
Usage:
ShinyDriver$executeScriptAsync(script, ...)
Arguments:
script JS to execute.
... Additional arguments to script.
Returns: Self, invisibly.

Method findWidget(): Finds the a Shiny input or output control.
Usage:
ShinyDriver$findWidget(name, iotype = c("auto", "input", "output"))
Arguments:
name Name of a shiny widget.
iotype Type of the Shiny widget. Usually shinytest finds the widgets by their name, so this is
only needed if you use the same name for an input and output widget.
Returns: A Widget.

Method expectUpdate(): It performs one or more update operations via the browser, thens
waits for the specified output(s) to update. The test succeeds if all specified output widgets are
updated before the timeout. For updates that involve a lot of computation, increase the timeout.
Usage:
ShinyDriver$expectUpdate(
  output,
  ..., timeout = 3000,
  iotype = c("auto", "input", "output")
)
Arguments:
output Name of output control to check.
... Name-value pairs used to update inputs.
timeout Amount of time to wait before giving up (milliseconds).
ioType  Type of the Shiny widget. Usually shinytest finds the widgets by their name, so this is only needed if you use the same name for an input and output widget.

**Method** setInputs()  Sets input values.

**Usage:**

ShinyDriver$setInputs(
  ...,  
  wait_ = TRUE,  
  values_ = TRUE,  
  timeout_ = 3000,  
  allowInputNoBinding_ = FALSE,  
  priority_ = c("input", "event")
)

**Arguments:**

... Name-value pairs, name1 = value1, name2 = value2 etc. Enput with name name1 will be assigned value value1.

wait_  Wait until all reactive updates have completed?

values_  If TRUE, will return final updated values of inputs.

timeout_  Amount of time to wait before giving up (milliseconds).

allowInputNoBinding_  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.

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

**Returns:**  Returns updated values, invisibly.

**Method** uploadFile()  Uploads a file to a file input.

**Usage:**

ShinyDriver$uploadFile(..., wait_ = TRUE, values_ = TRUE, timeout_ = 3000)

**Arguments:**

... Name-path pairs, e.g. name1 = path1. The file located at path1 will be uploaded to file input with name name1.

wait_  Wait until all reactive updates have completed?

values_  If TRUE, will return final updated values of download control.

timeout_  Amount of time to wait before giving up (milliseconds).

**Method** snapshotInit()  Download a snapshot. Generally, you should not call this function yourself; it will be generated by recordTest() as needed.

**Usage:**

ShinyDriver$snapshotInit(path, screenshot = TRUE)

**Arguments:**

path  Directory to save snapshots.

screenshot  Take screenshots for each snapshot?

**Method** snapshot()  Take a snapshot. Generally, you should not call this function yourself; it will be generated by recordTest() as needed.
Usage:
ShinyDriver$snapshot(items = NULL, filename = NULL, screenshot = NULL)

Arguments:
items  Elements to include in snapshot
filename  Filename to use. It is recommended to use a .json file extension.
screenshot  Take a screenshot? Overrides value set by $snapshotInit()

Method snapshotCompare(): Deprecated
Usage:
ShinyDriver$snapshotCompare(...)
Arguments:
...  Ignored

Method snapshotDownload(): Snapshot a file download action. Generally, you should not call this function yourself; it will be generated by recordTest() as needed.

Usage:
ShinyDriver$snapshotDownload(id, filename = NULL)
Arguments:
id  Output id of shiny::downloadButton()/shiny::downloadLink()
filename  File name to save file to. The default, NULL, generates an ascending sequence of names: 001.download, 002.download, etc.

Method getAppDir(): Directory where app is located
Usage:
ShinyDriver$getAppDir()

Method getAppFilename(): App file name, i.e. app.R or server.R. NULL for Rmds.
Usage:
ShinyDriver$getAppFilename()

Method getTestsDir(): Directory where tests are located
Usage:
ShinyDriver$getTestsDir()

Method getRelativePathToApp(): Relative path to app from current directory.
Usage:
ShinyDriver$getRelativePathToApp()

Method getSnapshotDir(): Directory where snapshots are located.
Usage:
ShinyDriver$getSnapshotDir()

Method isRmd(): Is this app an Shiny Rmd document?
Usage:
ShinyDriver$isRmd()

Method clone(): The objects of this class are cloneable with this method.

Usage:
ShinyDriver$clone(deep = FALSE)

Arguments:
deep Whether to make a deep clone.

Description
Run tests for a Shiny application

Usage

testApp(
  appDir = ".",
  testnames = NULL,
  quiet = FALSE,
  compareImages = TRUE,
  interactive = is_interactive(),
  suffix = NULL
)

Arguments

appDir Path to directory containing a Shiny app (e.g. app.R) or single interactive .Rmd.
testnames Test script(s) to run. The .R extension of the filename is optional. For example, "mytest" or c("mytest", "mytest2.R"). If NULL (the default), all scripts in the tests/ directory will be run.
quiet Should output be suppressed? This is useful for automated testing.
compareImages Should screenshots be compared? It can be useful to set this to FALSE when the expected results were taken on a different platform from the one currently being used to test the application.
interative If there are any differences between current results and expected results, provide an interactive graphical viewer that shows the changes and allows the user to accept or reject the changes.
suffix An optional suffix for the expected results directory. For example, if the suffix is "mac", the expected directory would be mytest-expected-mac.

See Also

snapshotCompare() and snapshotUpdate() if you want to compare or update snapshots after testing. In most cases, the user is prompted to do these tasks interactively, but there are also times where it is useful to call these functions from the console.
Widget

A Shiny Widget

Description

A Widget object represents a Shiny input or output control, and provides methods for finer grained interaction.

Methods

Public methods:

• Widget$new()
• Widget$getName()
• Widget$getElement()
• Widget$getHtml()
• Widget$getType()
• Widget$getIoType()
• Widget$isInput()
• Widget$isOutput()
• Widget$getValue()
• Widget$setValue()
• Widget$click()
• Widget$sendKeys()
• Widget$listTabs()
• Widget$uploadFile()
• Widget$clone()

Method new(): Create new Widget

Usage:
Widget$new(name, element, type, iotype = c("input", "output"))

Arguments:
name Name of a Shiny widget.
element webdriver::Element
type Widget type
iotype Input/output type.

Method getName(): Control id (i.e. inputId or outputId that control was created with).

Usage:
Widget$getName()

Method getElement(): Underlying webdriver::Element() object.

Usage:
Method `getElement()`

Method `getHtml()`: retrieve the underlying HTML for a widget

Usage:
`Widget$getElement()`

`getHtml()`:
```
retrieve the underlying HTML for a widget
```

Usage:
`Widget$getHtml()`

Method `getType()`: Widget type, e.g. `textInput`, `selectInput`.

Usage:
`Widget$getType()`

Method `getIoType()`: Is this an input or output control?

Usage:
`Widget$getIoType()`

Method `isInput()`: Is this an input control?

Usage:
`Widget$isInput()`

Method `isOutput()`: Is this an output control?

Usage:
`Widget$isOutput()`

Method `getValue()`: Get current value of control.

Usage:
`Widget$getValue()`

Method `setValue()`: Set value of control.

Usage:
`Widget$setValue(value)`

Arguments:
- `value` Value to set for the widget.

Method `click()`: scrolls the element into view, then clicks the in-view centre point of it.

Usage:
`Widget$click()`

Returns: `self`, invisibly.

Method `sendKeys()`: Send specified key presses to control.

Usage:
`Widget$sendKeys(keys)`

Arguments:
- `keys` Keys to send to the widget or the app. See `webdriver::key` for how to specific special keys.

Method `listTabs()`: Lists the tab names of a `shiny::tabsetPanel()`. It fails for other types of widgets.
Usage:
Widget$listTabs()

Method uploadFile(): Upload a file to a `shiny::fileInput()`. It fails for other types of widgets.

Usage:
Widget$uploadFile(filename)

Arguments:
filename  Path to file to upload

Method clone(): The objects of this class are cloneable with this method.

Usage:
Widget$clone(deep = FALSE)

Arguments:
deep  Whether to make a deep clone.
Index

base::options(), 7

dependenciesInstalled, 2

expect_pass, 2

installDependencies
  (dependenciesInstalled), 2

migrateShinytestDir, 3

osName, 4

recordTest, 4
recordTest(), 13, 14

shiny::downloadButton(), 14
shiny::downloadLink(), 14
shiny::fileInput(), 18
shiny::runApp(), 7
shiny::tabsetPanel(), 17
ShinyDriver, 5
ShinyDriver(), 5
snapshotCompare(), 15
snapshotUpdate(), 15

testApp, 15
testApp(), 2, 4

webdriver::Element, 10, 16
webdriver::Element(), 16
webdriver::install_phantomjs(), 2
webdriver::key, 8, 17
Widget, 7, 8, 12, 16