Package ‘shinytest’

June 18, 2020

Title Test Shiny Apps

Version 1.4.0

Description For automated testing of Shiny applications, using a headless browser, driven through 'WebDriver'.

License MIT + file LICENSE

LazyData true

URL `https://github.com/rstudio/shinytest`

BugReports `https://github.com/rstudio/shinytest/issues`

RoxygenNote 7.1.0

Imports assertthat, digest, crayon, debugme, parsedate, pingr, callr (>= 2.0.3), R6, rematch, httr, shiny (>= 1.3.2), testthat (>= 1.0.0), utils, webdriver (>= 1.0.5), htmlwidgets, jsonlite, withr, httpuv, rstudioapi (>= 0.8.0.9002)

Suggests rmarkdown, flexdashboard

Encoding UTF-8

SystemRequirements PhantomJS (http://phantomjs.org/)

NeedsCompilation no

Author Winston Chang [aut, cre],
Gábor Csárdi [aut],
RStudio [cph, fnd],
Mango Solutions [cph, ccp]

Maintainer Winston Chang `<winston@rstudio.com>`

Repository CRAN

Date/Publication 2020-06-18 18:00:07 UTC

R topics documented:

`dependenciesInstalled` *Checks all dependencies are installed*

Description

Checks that all the required system dependencies are installed properly, returns. If dependencies are missing, consider running installDependencies.

Usage

```
dependenciesInstalled()
```

Value

`TRUE` when all dependencies are fulfilled; otherwise, `FALSE`.

See Also

`installDependencies` to install missing dependencies.

`diffviewer_widget` *Creat an htmlwidget that shows differences between files or directories*

Description

This function can be used for viewing differences between current test results and the expected results

Usage

```
diffviewer_widget(old, new, width = NULL, height = NULL, pattern = NULL)
```

Arguments

`old, new`	Names of the old and new directories to compare. Alternatively, they can be a character vectors of specific files to compare.
`width`	Width of the htmlwidget.
`height`	Height of the htmlwidget
`pattern`	A filter to apply to the old and new directories.

`expectUpdate` `testthat` *expectation for a Shiny update*

Description

`testthat` expectation for a Shiny update

Usage

```
expectUpdate(
  app,
  output,
  ...,
  timeout = 3000,
  iotype = c("auto", "input", "output")
)
```

Arguments

`app`	A `ShinyDriver` object.
`output`	Character vector, the name(s) of the output widgets that are required to update for the test to succeed.
`...`	Named arguments specifying updates for Shiny input widgets.
`timeout`	Timeout for the update to happen, in milliseconds.
`iotype`	Type of the widget(s) to change. These are normally input widgets.

Examples

```
## Not run:
## https://github.com/rstudio/shiny-examples/tree/master/050-kmeans-example
app <- ShinyDriver$new("050-kmeans-example")
expectUpdate(app, xcol = "Sepal.Width", output = "plot1")
expectUpdate(app, ycol = "Petal.Width", output = "plot1")
expectUpdate(app, clusters = 4, output = "plot1")

## End(Not run)
```

`expect_pass` *Expectation: shinytest object passed snapshot tests*

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).

Examples

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

## End(Not run)
```

`installDependencies` *Installs missing dependencies*

Description

Installs all the required system depencies to record and run tests. This will install a headless web browser, PhantomJS.

Usage

```
installDependencies()
```

See Also

`dependenciesInstalled` to check if dependencies are missing. For more information about where PhantomJS will be installed, see `install_phantomjs`.

Examples

```
## Not run:

if (!dependenciesInstalled() &&
    identical(menu(c("Yes", "No"), "Install missing dependencies?"), 1L)) {
  installDependencies()
}


## 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

```
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
     |   `- mytest.R
     |- testthat.R
     `- testthat
         `- test-script.R
```

This allows for tests using the **shinytest** package as well as other testing tools, such as the `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.

`recordTest` *Launch test event recorder for a Shiny app*

Description

Launch test event recorder for a Shiny app

Usage

```
recordTest(
  app = ".",
  save_dir = NULL,
  load_mode = FALSE,
  seed = NULL,
  loadTimeout = 10000,
  debug = "shiny_console",
  shinyOptions = list()
)
```

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` *Class to manage a shiny app and a phantom.js headless browser*

Description

Class to manage a shiny app and a phantom.js headless browser

Usage

```
app <- ShinyDriver$new(path = ".", loadTimeout = 5000,
              checkNames = TRUE, debug = c("none", "all",
              ShinyDriver$debugLogTypes), phantomTimeout = 5000,
              seed = NULL, cleanLogs = TRUE, shinyOptions = list()))
app$stop()
app$getDebugLog(type = c("all", ShinyDriver$debugLogTypes))

app$getValue(name, iotype = c("auto", "input", "output"))
app$setValue(name, value, iotype = c("auto", "input", "output"))
app$sendKeys(name = NULL, keys)

app$getWindowSize()
app$setWindowSize(width, height)

app$getUrl()
app$goBack()
app$refresh()
app$getTitle()
app$getSource()
app$takeScreenshot(file = NULL)

app$findElement(css = NULL, linkText = NULL,
     partialLinkText = NULL, xpath = NULL)

app$findElements(css = NULL, linkText = NULL,
     partialLinkText = NULL, xpath = NULL)

app$waitFor(expr, checkInterval = 100, timeout = 3000)

app$waitForValue(name, ignore = list(NULL, ""), iotype = "input", timeout = 10000, checkInterval = 400)
```

```
app$listWidgets()

app$checkUniqueWidgetNames()

app$findWidget(name, iotype = c("auto", "input", "output"))

app$expectUpdate(output, ..., timeout = 3000,
    iotype = c("auto", "input", "output"))
```

Arguments

app A `ShinyDriver` instance.

path Path to a directory containing a Shiny app, i.e. a single `app.R` file or a `server.R` and `ui.R` pair.

loadTimeout How long to wait for the app to load, in ms. This includes the time to start R.

phantomTimeout How long to wait when connecting to phantomJS process, in ms.

checkNames Whether to check if widget names are unique in the app.

debug Whether to start the app in debugging mode. In debugging mode debug messages are printed to the console.

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 `runApp()`.

name Name of a shiny widget. For `$sendKeys` it can be `NULL`, in which case the keys are sent to the active HTML element.

iotype Type of the Shiny widget. Usually `shinytest` finds the widgets by their name, so this need not be specified, but Shiny allows input and output widgets with identical names.

keys Keys to send to the widget or the app. See the `sendKeys` method of the `webdriver` package.

width Scalar integer, the desired width of the browser window.

height Scalar integer, the desired height of the browser window.

file File name to save the screenshot to. If `NULL`, then it will be shown on the R graphics device.

css CSS selector to find an HTML element.

linkText Find <a> HTML elements based on their `innerText`.

partialLinkText Find <a> HTML elements based on their `innerText`. It uses partial matching.

xpath Find HTML elements using XPath expressions.

expr A string scalar containing JavaScript code that evaluates to the condition to wait for.

checkInterval How often to check for the condition, in milliseconds.

ignore List of possible values that are to not be considered valid. `app$waitForValue` will continue to poll until it finds a value not contained in `ignore`.

timeout Timeout for the condition, in milliseconds.

output Character vector, the name(s) of the Shiny output widgets that should be updated.

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.

... For `expectUpdate` these can be named arguments. The argument names correspond to Shiny input widgets: each input widget will be set to the specified value.

Details

`ShinyDriver$new()` function creates a `ShinyDriver` object. It starts the Shiny app in a new R session, and it also starts a `phantomjs` headless browser that connects to the app. It waits until the app is ready to use. It waits at most `loadTimeout` milliseconds, and if the app is not ready, then it throws an error. You can increase `loadTimeout` for slow loading apps. Currently it supports apps that are defined in a single `app.R` file, or in a `server.R` and `ui.R` pair.

`app$stop()` stops the app, i.e. the external R process that runs the app, and also the phantomjs instance.

`app$getDebugLog()` queries one or more of the debug logs: `shiny_console`, `browser` or `shinytest`.

`app$getValue()` finds a widget and queries its value. See the `getValue` method of the `Widget` class.

`app$setInputs()` sets the value of inputs. The arguments must all be named; an input with each name will be assigned the given value.

`app$uploadFile()` uploads a file to a file input. The argument must be named and the value must be the path to a local file; that file will be uploaded to a file input with that name.

`app$getAllValues()` returns a named list of all inputs, outputs, and export values.

`app$setValue()` finds a widget and sets its value. See the `setValue` method of the `Widget` class.

`app$sendKeys` sends the specified keys to the HTML element of the widget.

`app$getWindowSize()` returns the current size of the browser window, in a list of two integer scalars named 'width' and 'height'.

`app$setWindowSize()` sets the size of the browser window to the specified width and height.

`app$getUrl()` returns the current URL.

`app$goBack()` "presses" the browser's 'back' button.

`app$refresh()` "presses" the browser's 'refresh' button.

`app$getTitle()` returns the title of the page. (More precisely the document title.)

`app$getSource()` returns the complete HTML source of the current page, in a character scalar.

`app$takeScreenshot()` takes a screenshot of the current page and writes it to a file, or (if `file` is `NULL`) shows it on the R graphics device. The output file has PNG format.

`app$findElement()` find an HTML element on the page, using a CSS selector or an XPath expression. The return value is an `Element` object from the `webdriver` package.

`app$findElements()` finds potentially multiple HTML elements, and returns them in a list of `Element` objects from the `webdriver` package.

`app$waitFor()` waits until a JavaScript expression evaluates to `true`, or a timeout happens. It returns `TRUE` is the expression evaluated to `true`, possible after some waiting.

app$waitForValue() waits until the current application's input (or output) value is not one of the supplied invalid values. The function returns the value found if the time limit has not been reached (default is 10 seconds). This function can be useful in helping determine if an application has initialized or finished processing a complex reactive situation.

app$listWidgets() lists the names of all input and output widgets. It returns a list of two character vectors, named input and output.

app$checkUniqueWidgetNames() checks if Shiny widget names are unique.

app$findWidget() finds the corresponding HTML element of a Shiny widget. It returns a Widget object.

expectUpdate() is one of the main functions to test Shiny apps. It performs one or more update operations via the browser, and then waits for the specified output widgets to update. The test succeeds if all specified output widgets are updated before the timeout. For updates that involve a lot of computation, you increase the timeout.

Examples

```
## Not run:
## https://github.com/rstudio/shiny-examples/tree/master/050-kmeans-example
app <- ShinyDriver$new("050-kmeans-example")
expectUpdate(app, xcol = "Sepal.Width", output = "plot1")
expectUpdate(app, ycol = "Petal.Width", output = "plot1")
expectUpdate(app, clusters = 4, output = "plot1")

## End(Not run)
```

shinytest *Test Shiny Apps*

Description

Uses a headless browser, driven through 'WebDriver'. See ShinyDriver to get started.

snapshotCompare *Compare current and expected snapshots*

Description

This compares current and expected snapshots for a test set, and prints any differences to the console.

Usage

```
snapshotCompare(
  appDir,
  testnames = NULL,
  autoremove = TRUE,
  images = TRUE,
  quiet = FALSE,
  interactive = base::interactive(),
  suffix = NULL
)

snapshotUpdate(appDir = ".", testnames = NULL, quiet = FALSE, suffix = NULL)
```

Arguments

`appDir`	Directory that holds the tests for an application. This is the parent directory for the expected and current snapshot directories.
`testnames`	Name or names of a test. If NULL, compare all test results.
`autoremove`	If the current results match the expected results, should the current results be removed automatically? Defaults to TRUE.
`images`	Should screenshots and PNG images be compared? It can be useful to set this to `FALSE` when the expected results were taken on a different platform from the current results.
`quiet`	Should output be suppressed? This is useful for automated testing.
`interactive`	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

`testApp`

`testApp` *Run tests for a Shiny application*

Description

Run tests for a Shiny application

Usage

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

Arguments

`appDir`	Path to the Shiny application to be tested.
`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.
`interactive`	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.

`textTestDiff` *Get textual diff of test results*

Description

Get textual diff of test results

Usage

```
textTestDiff(appDir = ".", testnames = NULL, images = TRUE, suffix = NULL)
```

Arguments

`appDir`	Directory of the Shiny application that was tested.
`testnames`	A character vector of names of tests to compare. If NULL, compare all test results for which there are differences.
`images`	Compare screenshot images.
`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

`viewTestDiff` for interactive diff viewer.

`viewTestDiff` *View differences in test results*

Description

View differences in test results

Usage

```
viewTestDiff(
  appDir = ".",
  testnames = NULL,
  interactive = base::interactive(),
  images = TRUE,
  suffix = NULL
)
```

Arguments

`appDir`	Directory of the Shiny application that was tested.
`testnames`	A character vector of names of tests to compare. If NULL, compare all test results for which there are differences.
`interactive`	If TRUE, use the interactive diff viewer, which runs in a Shiny app. If FALSE, print a textual diff, generated by `textTestDiff`.
`images`	Compare screenshot images (only used when `interactive` is FALSE).
`suffix`	An optional suffix for the expected results directory. For example, if the suffix is `"mac"`, the expected directory would be `mytest-expected-mac`.

Value

A character vector the same length as `testnames`, with `"accept"` or `"reject"` for each test.

See Also

`textTestDiff` to get a text diff as a string.

`viewTestDiffWidget` *Interactive viewer widget for changes in test results*

Description

Interactive viewer widget for changes in test results

Usage

```
viewTestDiffWidget(appDir = ".", testname = NULL, suffix = NULL)
```

Arguments

`appDir`	Directory of the Shiny application that was tested.
`testname`	Name of test to compare.
`suffix`	An optional suffix for the expected results directory. For example, if the suffix is `"mac"`, the expected directory would be `mytest-expected-mac`.

`Widget` *Class for a Shiny widget*

Description

Class for a Shiny widget

Usage

```
w <- app$findWidget(name,
    iotype = c("auto", "input", "output"))

w$getName()
w$getElement()
w$getType()
w$getIoType()
w$isInput()
w$isOutput()

w$getValue()
w$setValue(value)

w$sendKeys(keys)

w$listTabs()
```

Arguments

app A `ShinyDriver` object.

w A `Widget` object.

name Name of a Shiny widget.

iotype Character scalar, whether the widget is 'input' or 'output'. The default 'auto' value works well, provided that widgets have unique names. (Shiny allows an input and an output widget with the same name.)

value Value to set for the widget. Its interpretation depends on the type of the widget, see details below.

keys Keys to send to the widget. See the `sendKeys` method of the `Element` class in the `webdriver` package.

Details

A `Widget` object represents a Shiny input or output widget. `app$findWidget` creates a widget object from a `ShinyDriver` object.

`w$getName()` returns the name of the widget.

`w$getElement()` returns an HTML element. This is an `Element` object from the `webdriver` package.

`w$getType()` returns the type of the widget, possible values are `textInput`, `selectInput`, etc.

`w$getIoType()` returns 'input' or 'output', whether the widget is an input or output widget.

`w$isInput()` returns `TRUE` for input widgets, `FALSE` otherwise.

`w$isOutput()` returns `TRUE` for output widgets, `FALSE` otherwise.

`w$getValue()` returns the value of the widget. The exact type returned depends on the type of the widget. TODO: list widgets and their return types.

`w$setValue()` sets the value of the widget, through the web browser. Different widget types expect different different `value` arguments. TODO: list widgets and types.

`w$sendKeys` sends the specified keys to the HTML element of the widget.

`w$listTabs` lists the tab names of a `tabsetPanel` widget. It fails for other types of widgets.

Examples

```
{

}
```

Index