# Package 'shinylogs'

April 18, 2022

**Title** Record Everything that Happens in a 'Shiny' Application

**Version** 0.2.1

**Description** Track and record the use of applications and the user's interactions with 'Shiny' inputs. Allows to trace the inputs with which the user interacts, the outputs generated, as well as the errors displayed in the interface.

**License** GPL-3

**Encoding** UTF-8

**RoxygenNote** 7.1.2

**URL** https://github.com/dreamRs/shinylogs

**BugReports** https://github.com/dreamRs/shinylogs/issues

**Imports** htmltools, shiny (>= 1.1.0), jsonlite, data.table, bit64, nanotime, digest, anytime

**Suggests** testthat, knitr, rmarkdown, covr, DBI, RSQLite, googledrive

**VignetteBuilder** knitr

**NeedsCompilation** no

**Author** Fanny Meyer [aut], Victor Perrier [aut, cre], Silex Technologies [fnd] (https://www.silex-ip.com)

**Maintainer** Victor Perrier <victor.perrier@dreamrs.fr>

**Repository** CRAN

**Date/Publication** 2022-04-18 16:20:02 UTC

## R topics documented:

---

`read_json_logs` *Read a directory containing JSON logs*

---

## Description

Read a directory containing JSON logs

## Usage

```
read_json_logs(path)
```

## Arguments

| | |
|---|---|
| `path` | Path of the directory containing JSON files or a vector of path to JSON files. |

## Value

a `list` of `data.table`

## Examples

```
# Read all JSON in a directory
path_directory <- system.file("extdata/json", package = "shinylogs")
logs <- read_json_logs(path = path_directory)

# Read a single file
single_file <- dir(
  path = system.file("extdata/json", package = "shinylogs"),
  full.names = TRUE
)[1]
logs <- read_json_logs(path = single_file)
```

## read_rds_logs *Read a directory containing RDS logs*

### Description

Read a directory containing RDS logs

### Usage

```
read_rds_logs(path)
```

### Arguments

| | |
|---|---|
| path | Path of the directory containing RDS files or a vector of path to RDS files. |

### Value

a list of data.table

### Examples

```
## Not run:
# Read all RDS in a directory
logs <- read_rds_logs(path = "path/to/directory")

# Read a single file
logs <- read_rds_logs(path = "path/to/log.rds")

## End(Not run)
```

## store_custom *Use custom function to save logs*

### Description

Store logs tracked where you want by providing a custom function to write them in your prefered location.

### Usage

```
store_custom(FUN, ...)
```

### Arguments

| | |
|---|---|
| FUN | A function that take at least one argument logs, that will correspond to logs recorded as a list. |
| ... | Extra parameters that will be passed to FUN. |

## Value

A list that can be used in `track_usage()`.

## Examples

```
library(shiny)
library(shinylogs)

# Classic Iris clustering with Shiny
ui <- fluidPage(

  headerPanel("Iris k-means clustering"),

  sidebarLayout(
    sidebarPanel(
      selectInput(
        inputId = "xcol",
        label = "X Variable",
        choices = names(iris)
      ),
      selectInput(
        inputId = "ycol",
        label = "Y Variable",
        choices = names(iris),
        selected = names(iris)[[2]]
      ),
      numericInput(
        inputId = "clusters",
        label = "Cluster count",
        value = 3,
        min = 1,
        max = 9
      )
    ),
    mainPanel(
      plotOutput("plot1")
    )
  )
)

server <- function(input, output, session) {

  # Just take a look at what is generated
  track_usage(
    storage_mode = store_custom(FUN = function(logs) {
      str(logs, max.level = 3)
      invisible()
    })
  )

  # classic server logic
```

```
  selectedData <- reactive({
    iris[, c(input$xcol, input$ycol)]
  })

  clusters <- reactive({
    kmeans(selectedData(), input$clusters)
  })

  output$plot1 <- renderPlot({
    palette(c("#E41A1C", "#377EB8", "#4DAF4A", "#984EA3",
              "#FF7F00", "#FFFF33", "#A65628", "#F781BF", "#999999"))

    par(mar = c(5.1, 4.1, 0, 1))
    plot(selectedData(),
         col = clusters()$cluster,
         pch = 20, cex = 3)
    points(clusters()$centers, pch = 4, cex = 4, lwd = 4)
  })

}

if (interactive())
  shinyApp(ui, server)
```

---

`store_googledrive` *Use Google Drive as storage mode*

---

## Description

All logs will be written in the same file.

## Usage

```
store_googledrive(path)
```

## Arguments

| | |
|---|---|
| `path` | Path to folder on Drive where to send logs. |

## Value

A list that can be used in `track_usage()`.

## Note

See the gargle package to manage authentication, and especially this vignette from gargle package to manage the process.

## Examples

```
## Not run:
# In your global, manage Google Drive access
drive_auth(path = "/path/to/your/service-account-token.json")
# see https://gargle.r-lib.org/articles/articles/managing-tokens-securely.html
# to manage your token securely

# Then in server, use:
track_usage(storage_mode = store_googledrive(path = "my-logs/"))

# you may have to share my-logs/ folder with your service account


## End(Not run)
```

---

`store_json` *Use JSON files as storage mode*

---

## Description

One JSON will be written for each session of the application.

## Usage

```
store_json(path)
```

## Arguments

| | |
|---|---|
| `path` | Path where to write JSON files. |

## Value

A list that can be used in `track_usage()`.

## Examples

```
library(shiny)
library(shinylogs)

# temp directory for writing logs
tmp <- tempdir()

# when app stop,
# navigate to the directory containing logs
onStop(function() {
  browseURL(url = tmp)
})

# Classic Iris clustering with Shiny
```

```
ui <- fluidPage(

  headerPanel("Iris k-means clustering"),

  sidebarLayout(
    sidebarPanel(
      selectInput(
        inputId = "xcol",
        label = "X Variable",
        choices = names(iris)
      ),
      selectInput(
        inputId = "ycol",
        label = "Y Variable",
        choices = names(iris),
        selected = names(iris)[[2]]
      ),
      numericInput(
        inputId = "clusters",
        label = "Cluster count",
        value = 3,
        min = 1,
        max = 9
      )
    ),
    mainPanel(
      plotOutput("plot1")
    )
  )
)

server <- function(input, output, session) {

  # Store JSON with logs in the temp dir
  track_usage(
    storage_mode = store_json(path = tmp)
  )

  # classic server logic

  selectedData <- reactive({
    iris[, c(input$xcol, input$ycol)]
  })

  clusters <- reactive({
    kmeans(selectedData(), input$clusters)
  })

  output$plot1 <- renderPlot({
    palette(c("#E41A1C", "#377EB8", "#4DAF4A", "#984EA3",
              "#FF7F00", "#FFFF33", "#A65628", "#F781BF", "#999999"))

    par(mar = c(5.1, 4.1, 0, 1))
```

```
    plot(selectedData(),
         col = clusters()$cluster,
         pch = 20, cex = 3)
    points(clusters()$centers, pch = 4, cex = 4, lwd = 4)
  })

}

if (interactive())
  shinyApp(ui, server)
```

---

`store_null` *No storage on disk*

---

## Description

Doesn't write anything, special inputs created by `track_usage()` are available in server and optionally logs are printed in console.

## Usage

```
store_null(console = TRUE)
```

## Arguments

| | |
|---|---|
| `console` | Print logs in R console. |

## Value

A list that can be used in `track_usage()`.

## Examples

```
library(shiny)
library(shinylogs)

ui <- fluidPage(
  tags$h2("Record inputs change"),
  fluidRow(
    column(
      width = 3,
      selectInput(
        inputId = "select",
        label = "Select input",
        choices = month.name
      ),
      numericInput(
        inputId = "numeric",
        label = "Numerci input",
        value = 4,
```

```
          min = 0, max = 20
        ),
        checkboxGroupInput(
          inputId = "checkboxGroup",
          label = "Checkbox group input",
          choices = LETTERS[1:5]
        ),
        sliderInput(
          inputId = "slider",
          label = "Slider input",
          min = 0, max = 100, value = 50
        )
      ),
      column(
        width = 9,
        tags$b("Last input:"),
        verbatimTextOutput(outputId = "last_input"),
        tags$b("All inputs:"),
        verbatimTextOutput(outputId = "all_inputs")
      )
    )
  )

  server <- function(input, output, session) {

    track_usage(
      storage_mode = store_null() # dont store on disk
    )

    output$last_input <- renderPrint({
      input$.shinylogs_lastInput # last input triggered
    })

    output$all_inputs <- renderPrint({
      input$.shinylogs_input # all inputs that have changed
    })

  }


  if (interactive())
    shinyApp(ui, server)
```

---

`store_rds` *Use RDS files as storage mode*

---

## Description

One RDS will be written for each session of the application.

## Usage

```
store_rds(path)
```

## Arguments

| | |
|---|---|
| path | Path where to write RDS files. |

## Value

A list that can be used in track_usage().

## Examples

```
library(shiny)
library(shinylogs)

# temp directory for writing logs
tmp <- tempdir()

# when app stop,
# navigate to the directory containing logs
onStop(function() {
  browseURL(url = tmp)
})

# Classir Iris clustering with Shiny
ui <- fluidPage(

  headerPanel("Iris k-means clustering"),

  sidebarLayout(
    sidebarPanel(
      selectInput(
        inputId = "xcol",
        label = "X Variable",
        choices = names(iris)
      ),
      selectInput(
        inputId = "ycol",
        label = "Y Variable",
        choices = names(iris),
        selected = names(iris)[[2]]
      ),
      numericInput(
        inputId = "clusters",
        label = "Cluster count",
        value = 3,
        min = 1,
        max = 9
      )
    ),
    mainPanel(
```

```
      plotOutput("plot1")
    )
  )
)

server <- function(input, output, session) {

  # Store RDS with logs in the temp dir
  track_usage(
    storage_mode = store_rds(path = tmp)
  )

  # classic server logic

  selectedData <- reactive({
    iris[, c(input$xcol, input$ycol)]
  })

  clusters <- reactive({
    kmeans(selectedData(), input$clusters)
  })

  output$plot1 <- renderPlot({
    palette(c("#E41A1C", "#377EB8", "#4DAF4A", "#984EA3",
              "#FF7F00", "#FFFF33", "#A65628", "#F781BF", "#999999"))

    par(mar = c(5.1, 4.1, 0, 1))
    plot(selectedData(),
         col = clusters()$cluster,
         pch = 20, cex = 3)
    points(clusters()$centers, pch = 4, cex = 4, lwd = 4)
  })

}


if (interactive())
  shinyApp(ui, server)
```

---

`store_sqlite` *Use SQLite database as storage mode*

---

## Description

All logs will be written in the same file.

## Usage

```
store_sqlite(path)
```

## Arguments

| | |
|---|---|
| path | Path to the SQLite file or a directory where to create one. |

## Value

A list that can be used in `track_usage()`.

## Examples

```
if (interactive()) {

  library(shiny)
  library(shinylogs)

  # temp directory for writing logs
  tmp <- tempdir()

  # when app stop,
  # navigate to the directory containing logs
  onStop(function() {
    browseURL(url = tmp)
  })

  # Classir Iris clustering with Shiny
  ui <- fluidPage(

    headerPanel("Iris k-means clustering"),

    sidebarLayout(
      sidebarPanel(
        selectInput(
          inputId = "xcol",
          label = "X Variable",
          choices = names(iris)
        ),
        selectInput(
          inputId = "ycol",
          label = "Y Variable",
          choices = names(iris),
          selected = names(iris)[[2]]
        ),
        numericInput(
          inputId = "clusters",
          label = "Cluster count",
          value = 3,
          min = 1,
          max = 9
        )
      ),
      mainPanel(
        plotOutput("plot1")
      )
```

```
  )
)

server <- function(input, output, session) {

  # Store RDS with logs in the temp dir
  track_usage(
    storage_mode = store_sqlite(path = tmp)
  )

  # classic server logic

  selectedData <- reactive({
    iris[, c(input$xcol, input$ycol)]
  })

  clusters <- reactive({
    kmeans(selectedData(), input$clusters)
  })

  output$plot1 <- renderPlot({
    palette(c("#E41A1C", "#377EB8", "#4DAF4A", "#984EA3",
              "#FF7F00", "#FFFF33", "#A65628", "#F781BF", "#999999"))

    par(mar = c(5.1, 4.1, 0, 1))
    plot(selectedData(),
         col = clusters()$cluster,
         pch = 20, cex = 3)
    points(clusters()$centers, pch = 4, cex = 4, lwd = 4)
  })

}

shinyApp(ui, server)

}
```

---

`track_usage` *Track usage of a Shiny app*

---

## Description

Used in Shiny `server` it will record all inputs and output changes and errors that occurs through an output.

## Usage

```
track_usage(
  storage_mode,
  what = c("session", "input", "output", "error"),
```

```
  exclude_input_regex = NULL,
  exclude_input_id = NULL,
  on_unload = FALSE,
  app_name = NULL,
  exclude_users = NULL,
  get_user = NULL,
  dependencies = TRUE,
  session = getDefaultReactiveDomain()
)
```

## Arguments

| | |
|---|---|
| storage_mode | Storage mode to use : store_json(), store_rds(), store_sqlite() or store_null(). |
| what | Elements to record between "session", "input", "output" and "error". |
| exclude_input_regex | Regular expression to exclude inputs from tracking. |
| exclude_input_id | Vector of inputId to exclude from tracking. |
| on_unload | Logical, save log when user close the browser window or tab, if TRUE it prevent to create shinylogs input during normal use of the application, there will be created only on close, downside is that a popup will appear asking to close the page. |
| app_name | Name of the app as a character string. If NULL, basename(getwd()) (name of the folder where application is located) is used. |
| exclude_users | Character vectors of user for whom it is not necessary to save the log. |
| get_user | A function to get user name, it should return a character and take one argument: the Shiny session. |
| dependencies | Load dependencies in client, can be set to FALSE if use_tracking() has been called in UI. |
| session | The shiny session. |

## Note

The following inputs will be accessible in the server:

- **.shinylogs_lastInput** : last input used by the user
- **.shinylogs_input** : all inputs send from the browser to the server
- **.shinylogs_error** : all errors generated by outputs elements
- **.shinylogs_output** : all outputs generated from the server
- **.shinylogs_browserData** : information about the browser where application is displayed.

## Examples

```
# Save logs on disk ---------------------------------

if (interactive()) {
```

```
# temporary directory for writing logs
tmp <- tempdir()

# when app stop,
# navigate to the directory containing logs
onStop(function() {
  browseURL(url = tmp)
})

# Classic Iris clustering with Shiny
ui <- fluidPage(

  headerPanel("Iris k-means clustering"),

  sidebarLayout(
    sidebarPanel(
      selectInput(
        inputId = "xcol",
        label = "X Variable",
        choices = names(iris)
      ),
      selectInput(
        inputId = "ycol",
        label = "Y Variable",
        choices = names(iris),
        selected = names(iris)[[2]]
      ),
      numericInput(
        inputId = "clusters",
        label = "Cluster count",
        value = 3,
        min = 1,
        max = 9
      )
    ),
    mainPanel(
      plotOutput("plot1")
    )
  )
)

server <- function(input, output, session) {

  # Store JSON with logs in the temp dir
  track_usage(
    storage_mode = store_json(path = tmp)
  )

  # classic server logic

  selectedData <- reactive({
    iris[, c(input$xcol, input$ycol)]
```

```
    })

    clusters <- reactive({
      kmeans(selectedData(), input$clusters)
    })

    output$plot1 <- renderPlot({
      palette(c("#E41A1C", "#377EB8", "#4DAF4A", "#984EA3",
                "#FF7F00", "#FFFF33", "#A65628", "#F781BF", "#999999"))

      par(mar = c(5.1, 4.1, 0, 1))
      plot(selectedData(),
           col = clusters()$cluster,
           pch = 20, cex = 3)
      points(clusters()$centers, pch = 4, cex = 4, lwd = 4)
    })

  }

  shinyApp(ui, server)

}

# Logs in console & special inputs ------------------------

if (interactive()) {
  library(shiny)
  library(shinylogs)

  ui <- fluidPage(
    tags$h2("Record inputs change"),
    fluidRow(
      column(
        width = 3,
        selectInput(
          inputId = "select",
          label = "Select input",
          choices = month.name
        ),
        numericInput(
          inputId = "numeric",
          label = "Numerci input",
          value = 4,
          min = 0, max = 20
        ),
        checkboxGroupInput(
          inputId = "checkboxGroup",
          label = "Checkbox group input",
          choices = LETTERS[1:5]
        ),
        sliderInput(
          inputId = "slider",
          label = "Slider input",
```

```
            min = 0, max = 100, value = 50
          )
        ),
        column(
          width = 9,
          tags$b("Last input triggered:"),
          verbatimTextOutput(outputId = "last_input"),
          tags$b("All inputs:"),
          verbatimTextOutput(outputId = "all_inputs")
        )
      )
    )

    server <- function(input, output, session) {

      # dont store on disk, just show in R console
      track_usage(
        storage_mode = store_null()
      )

      # last input triggered
      output$last_input <- renderPrint({
        input$.shinylogs_lastInput
      })

      # all inputs that have changed
      output$all_inputs <- renderPrint({
        input$.shinylogs_input
      })

    }

    shinyApp(ui, server)
  }
```

---

`use_tracking` *Insert dependencies to track usage of a Shiny app*

---

## Description

If used in UI of an application, this will create new `inputs` available in the server. Set `dependencies = FALSE` in `track_usage()` server-side to load dependencies only once.

## Usage

```
use_tracking(
  what = c("session", "input", "output", "error"),
  exclude_input_regex = NULL,
  exclude_input_id = NULL,
  on_unload = FALSE,
```

```
  app_name = NULL
)
```

## Arguments

| | |
|---|---|
| what | Elements to record between "session", "input", "output" and "error". |
| exclude_input_regex | Regular expression to exclude inputs from tracking. |
| exclude_input_id | Vector of inputId to exclude from tracking. |
| on_unload | Logical, save log when user close the browser window or tab, if TRUE it prevent to create shinylogs input during normal use of the application, there will be created only on close, downside is that a popup will appear asking to close the page. |
| app_name | Name of the app as a character string. If NULL, basename(getwd()) (name of the folder where application is located) is used. |

## Note

The following inputs will be accessible in the server (according to what is used in record argument):

- **.shinylogs_lastInput** : last input used by the user
- **.shinylogs_input** : all inputs send from the browser to the server
- **.shinylogs_error** : all errors generated by outputs elements
- **.shinylogs_output** : all outputs generated from the server
- **.shinylogs_browserData** : information about the browser where application is displayed.

## Examples

```
if (interactive()) {

  library(shiny)
  library(shinylogs)

  ui <- fluidPage(

    # Load tracking dependencies
    use_tracking(),

    splitLayout(
      cellArgs = list(style = "height: 250px"),
      radioButtons("radio", "Radio:", names(iris)),
      checkboxGroupInput("checkbox", "Checkbox:", names(iris)),
      selectInput("select", "Select:", names(iris))
    ),

    tags$p("Last input used, the 'name' slot correspond to inputId:"),
    verbatimTextOutput("last")
```

```
  )

  server <- function(input, output, session) {

    output$last <- renderPrint({
      input$.shinylogs_lastInput
    })

  }

  shinyApp(ui, server)

}
```

# Index