Package 'shinyDTC'

April 17, 2025

Type Package

Title Simple Dynamic Timer Control

Version 0.1.0

Description

A dynamic timer control (DTC) is a 'shiny' widget that enables time-based processes in applications. It allows users to execute these processes manually in individual steps or at customizable speeds. The timer can be paused, resumed, or restarted. This control is particularly well-suited for simulations, animations, countdowns, or interactive visualizations.

URL https://github.com/sigbertklinke/shinyDTC

License GPL-3

Encoding UTF-8

RoxygenNote 7.3.2

Imports rstudioapi, shiny, shinyjs

Suggests knitr, rmarkdown, shinydashboard

VignetteBuilder knitr

NeedsCompilation no

Author Sigbert Klinke [aut, cre],
Kleio Chrysopoulou Tseva [ctb]

Maintainer Sigbert Klinke <sigbert@hu-berlin.de>

Repository CRAN

Date/Publication 2025-04-17 08:00:02 UTC

Contents

---

openApp *openApp*

Description

Opens an example Shiny app into the RStudio editor.

Usage

```
openApp(dir = "mini", ...)
```

Arguments

dir	character: name of the app
...	further parameter given to rstudioapi::navigateToFile()

Value

invisibly the result of navigateToFile

Examples

```
# Example requires the use of a `shiny` app (See Vignette)
# Runs only in RStudio and in an interactive session
if (interactive()) {
  # Runs the minimal example app file in the editor
  openApp("mini")
}

## Not run:
# If the directory doesn't match, an error will appear
openApp("nonexistent")

## End(Not run)
```

---

resetTimer *resetTimer*

Description

Simulates a button press for Reset.

Usage

```
resetTimer(inputId)
```

Arguments

inputId	character: input slot

Value

nothing

Examples

```
# Example requires the use of a `shiny` app (See Vignette)
if (interactive()) {
  library(shiny)
  library(shinyDTC)
  library(shinyjs)

  ui <- fluidPage(
    useShinyjs(),
    timerUI("timer1"),
    actionButton("reset", "Trigger Reset")
  )

  server <- function(input, output, session) {
    timerServer("timer1")

    observeEvent(input$reset, {
      resetTimer("timer1")
    })
  }

  shinyApp(ui, server)
}

## Not run:
# `resetTimer` must be used inside a `shiny` server function
# and requires `shinyjs` + a corresponding timer UI

## End(Not run)
```

---

runAppx *runAppx*

Description

Runs an example Shiny app via `shiny::runExample()`.

Usage

```
runAppx(example = "mini", x = NULL, ...)
```

Arguments

example	character: name of the example to run, or NA to list the available examples
x	data frame: data to pass to the app
...	further parameter given to shiny::runExample()

Details

Note that for running a Shiny app more libraries might be necessary.

Value

invisibly the (modified) data set x

Examples

```
# Example requires the use of a `shiny` app (See Vignette)
# List all available examples
runAppx(NA)

## Not run:
# Run the "mini" example app
runAppx("mini")

# Run with a dataset passed to the app
check <- data.frame(a = 1:9)
runAppx("mini", x = check)

# Can also pass arguments to shiny::runExample
runAppx("mini", display.mode = "showcase")

## End(Not run)
```

---

stepTimer *stepTimer*

Description

Simulates a button press on Step.

Usage

```
stepTimer(inputId)
```

Arguments

inputId	character: input slot

Value

nothing

Examples

```
# Example requires the use of a `shiny` app (See Vignette)
if (interactive()) {
  library(shiny)
  library(shinyDTC)
  library(shinyjs)

  ui <- fluidPage(
    useShinyjs(),
    timerUI("ttimer"),
    actionButton("step", "Trigger Step (Externally)")
  )

  server <- function(input, output, session) {
    timerServer("ttimer")

    observeEvent(input$step, {
      stepTimer("ttimer")
    })
  }

  shinyApp(ui, server)
}

## Not run:
# stepTimer("ttimer") should be used inside a `shiny` server context

## End(Not run)
```

`timer` *timer*

Description

A reactive function.

Usage

```
timer(inputId, input, session, ...)
```

Arguments

`inputId`	character: input slot for a timer
`input`	an inpur object
`session`	a session object
`...`	further parameters for `shiny::reactive()`

Value

a reactive function which returns a counter

Examples

```
if (interactive()) vignette("shinyDTC")
```

`timerInput` *timerInput*

Description

Constructs a timer widget with a slider and two buttons.

Usage

```
timerInput(inputId, step1 = list(), reset = list(), ...)
```

Arguments

`inputId`	character: input slot
`step1`	list: list of parameters for the left (step) button
`reset`	list: list of parameters for the right (reset) button
`...`	parameters to the slider

Value

a widget to timer

Examples

```
if (interactive()) vignette("shinyDTC")
```

updateTimerInput *updateTimerInput*

Description

updateTimerInput

Usage

```
updateTimerInput(
  session = getDefaultReactiveDomain(),
  inputId,
  step1 = list(),
  reset = list(),
  ...
)
```

Arguments

session	session object
inputId	id of the input object
step1	modified values for Step button, see shiny::updateActionButton()
reset	modified values for Reset button, see shiny::updateActionButton()
...	modified values for Speed slider, see shiny::updateSliderInput()

Value

nothing

Examples

```
# Example requires the use of a `shiny` app (See Vignette)
# Updates the timer UI in a `shiny` app
if (interactive()) {
  library(shiny)
  library(shinyDTC)

  ui <- fluidPage(
    timerUI("timer1"),
    actionButton("update", "Update Timer UI")
  )

  server <- function(input, output, session) {
    timerServer("timer1")

    observeEvent(input$update, {
      updateTimerInput(
```

```
      session, "timer1",
      step1 = list(label = "Advance"),
      reset = list(label = "Clear"),
      min = 0, max = 100, value = 10
    )
  })
}

shinyApp(ui, server)
}
```

Index