Package: shinyreprex (via r-universe)

October 28, 2026

Type Package

Title Reproducible Code for 'Shiny' Objects

Version 0.4.0

Description Provides functionality to extract reactive expressions from a 'shiny' application and convert them into stand-alone R scripts. This enables users to reproduce tables and visualisations outside the interactive UI, facilitating integration into static reports or automated workflows without requiring access to the original application source code.

License MIT + file LICENSE

URL https://github.com/AscentSoftware/shinyreprex, https://ascentsoftware.github.io/shinyreprex/

BugReports https://github.com/AscentSoftware/shinyreprex/issues

Depends R (>= 4.3.0)

Imports S7, cli, constructive, purrr, renv, rlang, styler

Suggests dplyr, knitr, rmarkdown, shiny, spelling, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/Needs/website rmarkdown, ascentsoftware/acuitydown

Config/roxygen2/version 8.1.1

Config/testthat/edition 3

Encoding UTF-8

Language en-GB

Collate 'Repro_S7.R' 'S7_utils.R' 'repro_call_chunk.R' 'call_chunk_function.R' 'call_chunk_generic.R' 'call_chunk_if.R' 'call_chunk_null.R' 'call_chunk_reactive.R' 'call_chunk_reactval.R' 'call_chunk_shiny.R' 'call_chunk_subset.R' 'call_chunk_switch.R' 'call_chunk_utils.R' 'repro_chunk.R' 'chunk_call.R' 'chunk_generic.R' 'chunk_reactive.R' 'package.R' 'reactive_expression.R' 'register_reactives.R'

'reprex_header.R' 'reprex_lockfile.R' 'reprex_reactive.R'
'walk_packages.R'

NeedsCompilation no

Author Ashley Baldry [aut, cre]

Maintainer Ashley Baldry <ashley.baldry@acuityanalytics.com>

Repository https://cran.r-universe.dev

Date/Publication 2026-10-28 21:45:46 UTC

RemoteUrl https://github.com/cran/shinyreprex

RemoteRef HEAD

RemoteSha 7d252ed4ccb1751803ef2304fc334673cb6bfc4a

Contents

register_reactives 2
reprex_header 3
reprex_lockfile 5
reprex_packages 6
reprex_reactive 7

Index **10**

`register_reactives` *Register Reactives for Reproduction*

Description

Record reactives against the current Shiny session, so that `reprex_packages()` and `reprex_lockfile()` can be called with no arguments and still cover the whole application.

This avoids having to return reactives out of every module purely so a single top-level call can see them. Each module registers what it owns, and the session holds the collection.

Namespacing:

Registrations are namespaced by the calling module, so two modules may register reactives of the same name without collision, and re-registering the same name in the same module replaces the previous entry rather than adding a duplicate.

When to Register:

Registering does not evaluate the reactive. Packages are resolved by reading the expression held in the reactive, so it may be registered while still gated behind `shiny::req()` or inputs that have yet to be set. Registering at module setup is therefore both safe and preferred, as it does not depend on the user having visited the output first.

Session Scope:

The collection lives on the session, so it is discarded when the session ends and is never shared between concurrent users of the same application.

Usage

```
register_reactives(..., session = shiny::getDefaultReactiveDomain())
```

Arguments

`...`	One or more `shiny::reactive` objects to register. Names are taken from the arguments, so `register_reactives(summary_tbl)` registers under `"summary_tbl"`. Supply names explicitly to override.
`session`	The Shiny session to register against. Defaults to the current reactive domain, which inside a module is that module's session.

Value

The registered reactives, invisibly, as a named list.

See Also

`reprex_packages()` and `reprex_lockfile()`, which read the registered collection when called with no reactives.

Examples

```
library(shiny)

summaryServer <- function(id) {
  moduleServer(id, function(input, output, session) {
    summary_tbl <- reactive(purrr::keep(iris, is.numeric))

    register_reactives(summary_tbl)

    summary_tbl
  })
}

# Elsewhere in the application, with no reactives threaded through:
# reprex_lockfile()
```

`reprex_header` *Create an* `ir` *Header to Reproduce Reactives*

Description

Capture the exact package versions, R version and a dependency snapshot date needed to reproduce one or more `shiny::reactive` objects, as a single ir header. Prepending the header to the script returned by `reprex_reactive()` lets the recipient run `ir run script.R` with no separate lockfile or project to share.

Snapshot Scope:
As with reprex_lockfile(), the package versions pinned are read from the currently loaded library (.libPaths()), so they match the running Shiny session that produced the reactives.

Usage

```
reprex_header(
  ...,
  packages = NULL,
  exclude = NULL,
  r_version = TRUE,
  exclude_newer = Sys.Date(),
  isolated = TRUE,
  session = shiny::getDefaultReactiveDomain()
)
```

Arguments

...	One or more shiny::reactive objects to reproduce. If none are supplied, every reactive registered by register_reactives() is used, so a modular application can offer a whole-app header without threading reactives up to the top level.
packages	Character vector of package names to pin. If NULL (the default), every package detected across ... is used. Supplying this lets the user narrow the set, or add a package the detector could not find (for example one attached only for an operator or method).
exclude	Character vector of package names to omit from the header.
r_version	Whether to pin the R version the script was generated under. TRUE (the default) pins the running getRversion() exactly; FALSE omits the r-version field; a string is used as-is, for example ">= 4.3" to require a minimum rather than an exact version.
exclude_newer	A Date, or a string coercible to one, naming the Posit Package Manager snapshot ir resolves dependencies against. Defaults to today, so dependencies are fixed as of the day the header is created.
isolated	Whether ir installs packages into an isolated, per-script library rather than reusing packages already available. Defaults to TRUE.
session	The Shiny session to read registered reactives from. Only used when ... is empty and packages is NULL.

Value

A single string holding the ir header, including a trailing newline, ready to be pasted above the output of reprex_reactive().

See Also

reprex_packages() to inspect the detected set without building a header, reprex_lockfile() for a whole-project renv lockfile, and register_reactives() to record reactives from within each module.

Examples

```
library(shiny)

numeric_iris <- reactive(purrr::keep(iris, is.numeric))

# Outside a running application, isolate supplies the reactive context
isolate(cat(reprex_header(numeric_iris, exclude_newer = "2025-05-15")))
```

`reprex_lockfile` *Create a Lockfile to Reproduce Reactives*

Description

Capture the exact package versions, sources and R version needed to reproduce one or more `shiny::reactive` objects, as an `renv` lockfile. Restoring the lockfile with `renv::restore()` recreates the environment the reactives were generated in, including the recursive dependency tree.

A single lockfile covers every reactive passed, so an application can offer one download that reproduces all of its outputs.

Snapshot Scope:

The snapshot is taken from the currently loaded library (`.libPaths()`), so the versions recorded are exactly those the running Shiny session used to produce the reactives. `renv` resolves and records the full recursive dependency tree of `packages`, so only the top-level set needs to be supplied.

Isolation:

The snapshot runs against a throwaway temporary project, so it never writes `renv` infrastructure into, or otherwise modifies, the application's own directory.

Output Path:

A relative `lockfile` is resolved against the working directory, not against the temporary project used for the snapshot. The default therefore writes `renv.lock` into whichever directory the application is running from. Pass an absolute path to control where it lands; inside a `shiny::downloadHandler()` that is the `file` argument.

Usage

```
reprex_lockfile(
  ...,
  packages = NULL,
  lockfile = "renv.lock",
  exclude = NULL,
  session = shiny::getDefaultReactiveDomain()
)
```

Arguments

`...`	One or more `shiny::reactive` objects to reproduce. If none are supplied, every reactive registered by `register_reactives()` is used, so a modular application can offer a whole-app lockfile without threading reactives up to the top level.
`packages`	Character vector of package names to snapshot. If `NULL` (the default), every package detected across `...` is used. Supplying this lets the user narrow the set, or add a package the detector could not find (for example one attached only for an operator or method).
`lockfile`	Path to write the lockfile to. Defaults to `"renv.lock"`.
`exclude`	Character vector of package names to omit from the snapshot.
`session`	The Shiny session to read registered reactives from. Only used when `...` is empty and `packages` is `NULL`.

Value

The absolute path the lockfile was written to, invisibly.

See Also

`reprex_packages()` to inspect the detected set without writing a lockfile, and `register_reactives()` to record reactives from within each module.

Examples

```
library(shiny)

numeric_iris <- reactive(purrr::keep(iris, is.numeric))

isolate(reprex_lockfile(numeric_iris, lockfile = tempfile(fileext = ".lock")))
```

`reprex_packages` *Packages Required to Reproduce Reactives*

Description

Extract the set of non-base packages needed to reproduce one or more `shiny::reactive` objects, with duplicates removed across every reactive passed.

Use this when building a custom UI around `reprex_lockfile()`: it gives you the list of detected packages to present to the user for selection.

Packages are found by reading the expression held in each reactive rather than by generating its script, so every branch of an `if` or `switch` contributes. The result may therefore be a superset of the `library()` calls `reprex_reactive()` emits for the branch actually taken, on the basis that a lockfile is safer holding a package that is not needed than missing one.

Usage

```
reprex_packages(..., session = shiny::getDefaultReactiveDomain())
```

Arguments

`...`	One or more `shiny::reactive` objects to inspect. If none are supplied, every reactive registered against `session` by `register_reactives()` is used.
`session`	The Shiny session to read registered reactives from. Only used when `...` is empty. Defaults to the current reactive domain.

Value

A character vector of unique package names. Base packages are excluded, as they require no installation.

See Also

`reprex_lockfile()` to turn this set into an `renv` lockfile, and `register_reactives()` to record reactives from within each module.

Examples

```
library(shiny)

numeric_iris <- reactive(purrr::keep(iris, is.numeric))
styled_code <- reactive(styler::style_text("1 + 1"))

# Outside a running application, isolate supplies the reactive context
isolate(reprex_packages(numeric_iris, styled_code))
```

`reprex_reactive` *Reproduce Code*

Description

Construct the code within a given `shiny::reactive` object to be able to re-create the output outside of a Shiny session.

Usage

```
reprex_reactive(x, header = FALSE, ...)
```

Arguments

x	shiny::reactive object to make reproducible
header	Whether to prepend a reprex_header() pinning the packages, R version and a dependency snapshot date x needs, so the returned script is reproducible as a single file. Defaults to FALSE.
...	Passed on to reprex_header() when header is TRUE, for example exclude_newer or isolated. Ignored, with a warning, otherwise.

Value

A character string, that when printed (using base::cat), displays the script that reproduces the contents of x, optionally preceded by the ir header from reprex_header().

See Also

reprex_header() to build the header on its own, for example to cover several reactives with one header.

Examples

```
library(shiny)

ui <- fluidPage(
  h1("Reproducible Code Example"),
  inputPanel(
    sliderInput(
      "min_width",
      "Minimum Petal Width",
      min(iris$Petal.Width),
      max(iris$Petal.Width),
      min(iris$Petal.Width),
      step = 0.1
    ),
    selectInput(
      "summary_fn",
      "Summary Function",
      c("Mean" = "mean", "Median" = "median", "SD" = "sd"),
      selected = "mean"
    )
  ),
  fluidRow(
    column(
      width = 5,
      h2("Table"),
      tableOutput("table")
    ),
    column(
      width = 7,
      h2("Code"),
      verbatimTextOutput("code")
    )
```

```
  )
)

server <- function(input, output, session) {
  iris_filt <- reactive({
    iris[with(iris, Petal.Width > input$min_width), ]
  })

  summary_tbl <- reactive({
    aggregate(
      Sepal.Width ~ Species,
      data = iris_filt(),
      FUN = get(input$summary_fn)
    )
  })

  output$table <- renderTable(summary_tbl())
  output$code <- renderText(reprex_reactive(summary_tbl))

  # A single file, reproducible with `ir run script.R`
  output$code_with_header <- renderText(
    reprex_reactive(summary_tbl, header = TRUE, exclude_newer = "2025-05-15")
  )
}

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

Index

register_reactives, 2
register_reactives(), *4*, *6, 7*
renv::restore(), *5*
reprex_header, 3
reprex_header(), *8*
reprex_lockfile, 5
reprex_lockfile(), *2–4*, *6, 7*
reprex_packages, 6
reprex_packages(), *2–4*, *6*
reprex_reactive, 7
reprex_reactive(), *3, 4*, *6*

shiny::downloadHandler(), *5*
shiny::req(), *2*