Package 'undomanager'

September 12, 2026

Title Manage the History of Any Object with Undo/Redo Operations

Version 0.1.0

Description Track the history of any R object and move through it with undo and redo operations. Anything can be stored, from a single number to a data frame or an entire application state. A manager can restrict its history to specific classes and cap how many items it keeps, and it can be reactive to integrate with 'Shiny'.

URL https://github.com/daattali/undomanager

BugReports https://github.com/daattali/undomanager/issues

Depends R (>= 4.0.0)

Imports checkmate, R6

License MIT + file LICENSE

Encoding UTF-8

Suggests testthat (>= 3.0.0), shiny

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Dean Attali [aut, cre] (ORCID: <https://orcid.org/0000-0002-5645-3493>)

Maintainer Dean Attali <daattali@gmail.com>

Repository CRAN

Date/Publication 2026-09-12 08:50:02 UTC

Contents

all.equal.UndoManager *Compare two UndoManager objects*

Description

Two managers are equal when they hold the same value, the same undo and redo history, and the same `type` restriction. Internal bookkeeping, such as the counter used to trigger shiny reactivity, is ignored, so two managers that reached the same state by different routes compare as equal.

Usage

```
## S3 method for class 'UndoManager'
all.equal(target, current, ...)
```

Arguments

`target`, `current`	The two `UndoManager` objects to compare.
`...`	Passed on to `all.equal()`.

Details

Note that `identical()` cannot be used to compare managers: R6 objects are environments, so `identical()` always reports two separate managers as different regardless of their contents.

Value

`TRUE` if the two managers are equal, otherwise a character vector describing the differences.

UndoManager *Undo/Redo manager*

Description

With the undo manager, you can manage the history of an object by using undo and redo operations.

Managers are usually created with `undomanager()` rather than by calling `UndoManager$new()` directly. The class itself is exported for anyone who wants to work with it directly, such as to subclass it.

Active bindings

`value` Get the value

`is_empty` Whether the manager holds no value at all

`can_undo` Whether there are any undo operations available

`can_redo` Whether there are any redo operations available

`undo_size` Get the number of undo operations

`redo_size` Get the number of redo operations

Methods

Public methods:

- UndoManager$new()
- UndoManager$reactive()
- UndoManager$print()
- UndoManager$undo()
- UndoManager$redo()
- UndoManager$do()
- UndoManager$clear()

`UndoManager$new()`: Create a new undo manager. The manager starts out empty; the first call to `do()` gives it a value. Most users should call `undomanager()` instead, which is equivalent.

Usage:

```
UndoManager$new(type = NULL, allow_null = FALSE, max_size = Inf)
```

Arguments:

`type` The permitted classes of the items (`NULL` to allow any object). An item is accepted when any of these classes appears among the classes R would dispatch on, so `"numeric"` also accepts integers and numeric matrices.

`allow_null` Whether `NULL` values are allowed. Only used when `type` is given; an untyped manager accepts any object, including `NULL`.

`max_size` The maximum number of items to keep. Once the history grows past this, the oldest items are dropped. Use `Inf` for no limit.

Returns: A new `UndoManager` object.

Examples:

```
UndoManager$new()
UndoManager$new(type = "numeric")
UndoManager$new(type = "numeric", allow_null = TRUE)
UndoManager$new(max_size = 10)
```

`UndoManager$reactive()`: Get a shiny reactive for this manager, so that it can be used inside a reactive context. The returned reactive invalidates whenever the value changes, and calling it returns the manager itself.

Usage:

```
UndoManager$reactive()
```

Returns: A shiny reactive expression that returns the manager.

Examples:

```
if (requireNamespace("shiny", quietly = TRUE)) {
  nums <- undomanager()$do(5)
  rx <- nums$reactive()
  shiny::isolate(rx()$value)
}
```

`UndoManager$print()`: Print the manager: the classes it accepts, how many undo and redo operations are available, the current item, and the items in the undo and redo history.

Usage:

```
UndoManager$print(...)
```

Arguments:

`...` Not used.

Returns: The manager, invisibly.

Examples:

```
undomanager()$do(5)$do(7)$do(10)$undo()$print()
```

`UndoManager$undo()`**:** Move back in the history, making the previous item current. Undoing when there is nothing left to undo does nothing.

Usage:

```
UndoManager$undo(n = 1)
```

Arguments:

`n` The number of undo operations to perform. If `n` is larger than the number of available undo operations, all of them are performed. Use `Inf` to undo the entire history.

Returns: The manager, invisibly.

Examples:

```
nums <- undomanager()$do(5)$do(7)$do(10)
nums$undo()$value
nums$undo(Inf)$value
```

`UndoManager$redo()`**:** Move forward in the history, reversing an undo. Any redo history is discarded as soon as a new item is added with `do()`. Redoing when there is nothing left to redo does nothing.

Usage:

```
UndoManager$redo(n = 1)
```

Arguments:

`n` The number of redo operations to perform. If `n` is larger than the number of available redo operations, all of them are performed. Use `Inf` to redo the entire history.

Returns: The manager, invisibly.

Examples:

```
nums <- undomanager()$do(5)$do(7)$do(10)$undo(2)
nums$redo()$value
nums$redo(Inf)$value
```

`UndoManager$do()`**:** Add an item and make it the current value. The item that was current becomes the most recent undo, and any redo history is discarded.

Usage:

```
UndoManager$do(item)
```

Arguments:

`item` The item to add. It must satisfy the manager's `type`, if one was given.

Returns: The manager, invisibly.

Examples:

```
nums <- undomanager()
nums$do(5)
nums$do(7)$value
nums$undo()$do(99)$redo_size
```

`UndoManager$clear()`**:** Forget the undo and redo history, keeping the current value.

Usage:

```
UndoManager$clear(clear_value = FALSE)
```

Arguments:

`clear_value` Whether to also discard the current value, leaving the manager empty.

Returns: The manager, invisibly.

Examples:

```
nums <- undomanager()$do(5)$do(7)
nums$clear()$undo_size
nums$value
nums$clear(clear_value = TRUE)$is_empty
```

See Also

`undomanager()`, the recommended way to create a manager.

Examples

```
nums <- undomanager()
nums$do(5)
nums$do(7)
nums$do(10)
nums$undo()$value
nums$redo()$value

# operations return the manager, so they can be chained
undomanager()$do(1)$do(2)$do(3)$undo(2)$value

## ------------------------------------------------
## Method `UndoManager$new()`
## ------------------------------------------------

UndoManager$new()
UndoManager$new(type = "numeric")
UndoManager$new(type = "numeric", allow_null = TRUE)
UndoManager$new(max_size = 10)

## ------------------------------------------------
## Method `UndoManager$reactive()`
## ------------------------------------------------

if (requireNamespace("shiny", quietly = TRUE)) {
  nums <- undomanager()$do(5)
```

```
  rx <- nums$reactive()
  shiny::isolate(rx()$value)
}

## ------------------------------------------------
## Method `UndoManager$print()`
## ------------------------------------------------

undomanager()$do(5)$do(7)$do(10)$undo()$print()

## ------------------------------------------------
## Method `UndoManager$undo()`
## ------------------------------------------------

nums <- undomanager()$do(5)$do(7)$do(10)
nums$undo()$value
nums$undo(Inf)$value

## ------------------------------------------------
## Method `UndoManager$redo()`
## ------------------------------------------------

nums <- undomanager()$do(5)$do(7)$do(10)$undo(2)
nums$redo()$value
nums$redo(Inf)$value

## ------------------------------------------------
## Method `UndoManager$do()`
## ------------------------------------------------

nums <- undomanager()
nums$do(5)
nums$do(7)$value
nums$undo()$do(99)$redo_size

## ------------------------------------------------
## Method `UndoManager$clear()`
## ------------------------------------------------

nums <- undomanager()$do(5)$do(7)
nums$clear()$undo_size
nums$value
nums$clear(clear_value = TRUE)$is_empty
```

`undomanager` *Create an undo manager*

Description

Create a new UndoManager to track the history of an object and move through it with undo and redo operations. The manager starts out empty; the first call to `$do()` gives it a value.

Usage

```
undomanager(type = NULL, allow_null = FALSE, max_size = Inf)
```

Arguments

type	The permitted classes of the items (NULL to allow any object). An item is accepted when any of these classes appears among the classes R would dispatch on, so "numeric" also accepts integers and numeric matrices.
allow_null	Whether NULL values are allowed. Only used when type is given; an untyped manager accepts any object, including NULL.
max_size	The maximum number of items to keep. Once the history grows past this, the oldest items are dropped. Use Inf for no limit.

Details

This is the recommended way to create a manager. It is equivalent to `UndoManager$new()`, which is also available for anyone who wants to work with the R6 class directly, such as to subclass it.

A manager is an R6 object, so it has reference semantics: its methods modify the manager in place instead of returning a modified copy, and assigning a manager to a second variable does not copy it.

Value

A new UndoManager object.

See Also

UndoManager for the full list of methods and active bindings.

Examples

```
nums <- undomanager()
nums$do(5)
nums$do(7)
nums$do(10)
nums$undo()$value
nums$redo()$value

# operations return the manager, so they can be chained
undomanager()$do(1)$do(2)$do(3)$undo(2)$value

undomanager(type = "numeric")
undomanager(type = "numeric", allow_null = TRUE)
undomanager(max_size = 10)
```

Index