Package 'objectSignals'

May 9, 2026

License GPL (>= 2)

Maintainer Michael Lawrence <michafla@gene.com>

Title Observer Pattern for S4

Author Michael Lawrence, Tengfei Yin

Description A mutable Signal object can report changes to its state, clients could register functions so that they are called whenever the signal is emitted. The signal could be emitted, disconnected, blocked, unblocked, and buffered.

Depends R (>= 2.12)

Imports methods

Version 0.10.3

Collate Signal-class.R utils.R

NeedsCompilation no

Repository CRAN

Date/Publication 2022-04-05 06:00:02 UTC

Contents

`declareSignal` *Declaring a signal field*

Description

Declares a signal field that is lazily populated when the field is first accessed. This avoids the need for the constructor/initializer to explicitly create the signal.

Usage

```
declareSignal(expr)
```

Arguments

expr	The expression that names the signal and specifies its signature. See the example.

Value

A list of field definitions, suitable for passing to setRefClass.

Author(s)

Michael Lawrence

Examples

```
setRefClass("Dataset", fields = c(elements = "list",
declareSignal(elementsChanged(which))))
```

---

fieldWithPrototype *Fields with prototypes*

Description

A convenience for declaring a default value for a field, in the vein of prototype for S4 classes, except the default value is quoted and evaluated lazily.

Usage

```
fieldWithPrototype(name, class, value)
```

Arguments

name	The name of the field
class	The class of the field
value	Default value that when evaluated initializes the field

Value

A list suitable for use with setRefClass

Author(s)

Michael lawrence

Examples

```
Brush.gen <- setRefClass("Brush",
fields = fieldWithPrototype("color", "character", "red"))
brush <- Brush.gen$new()
brush$color
brush$color <- "blue"
brush$color
```

`Signal-class` *Signal objects*

Description

Creates a `Signal` object, with which a mutable object can report changes to its state. Interested clients register a function that is called whenever the signal is emitted. This allows those clients to respond to changes in the object state.

Details

A `Signal` object is usually created by a constructor and stored as a field in a reference class object. Clients then access the signal either directly or through an accessor.

The `Signal` reference class has the following methods:

connect(FUN, namedArgs = FALSE, ...) Connect `FUN` as a handler for the signal. A unique identifier is returned, which can be used to later disconnect the handler. Handler invocation follows these rules:

- `namedArgs=TRUE` arguments are named in call to handler. Otherwise, they are unnamed and matching is by position.
- If a handler is missing a signal argument, the argument is dropped when calling the handler.
- A handler may have arguments not in the signal signature.
- Arguments in `...` are appended to the handler call.

disconnect(id) Disconnects the handler registered with the identifier `id`.

emit(<signal signature>) Emits the signal, calling all of its handlers with the passed arguments. The signature depends on how the signal was constructed. All signal args must be passed to `emit`, unless they have a default.

block() Blocks emission of the signal. All emission requests are ignored.

unblock() Unblock the signal.

buffer() Buffer emissions, waiting to pass them to the handlers until `flush` is called.

flush() Flush the emission buffer, calling every handler on every buffered emission.

accumulator(value) If called with no arguments, get the function, if any, used to combine events in the buffer into a single event. Otherwise, `value` replaces the current function. The accumulator function should take one or two arguments. If it takes one argument, it is invoked upon a flush and is passed the list of events in the buffer. An event is simply a list containing the arguments passed to `emit`. If the accumulator function takes two arguments, the function is invoked upon every emission, when buffering is active and there is one event in the buffer. The first argument is the currently buffered event and the second is the new event that the function should merge into the first. The returned event then replaces the event in the buffer.

Constructor

- `Signal(...)` Create an instance of the reference class `Signal`

  `...` Arguments that express the signature of the signal

Accessors

- `length(x)`: The number of listeners in signal x.
- `listeners(object)`: A list of listeners in signal x.

Author(s)

Michael Lawrence, Tengfei Yin

Examples

```
Signal(x, y)
signal <- Signal(x, y, z = NA)
signal$connect(function(n, x, option = "none") message("x:", x),
               namedArgs = TRUE)
signal$connect(function(z, ...) message("z:", z, " x:", list(...)$x),
               namedArgs = TRUE)
signal$emit(0, 1)
##'
signal$connect(function(x, y, option = "none")
               message("y:", y, " op:", option), TRUE)
signal$connect(function(x, y, option = "none")
               message("op:", option), option = "test")
signal$connect(function(x, y, option = "none")
               message("op:", option), FALSE, "test")
id <- signal$connect(function(x, y, option = "none")
                     message("op:", option), TRUE, "test")
##'
signal$emit(0, 1)
##'
signal$disconnect(id)
signal$emit(0, 2)
##'
signal <- Signal(x)
signal$connect(function(i) print(i))
##'
signal$block()
```

```
signal$emit(0)
signal$unblock()
signal$emit(0)
##'
signal$buffer()
signal$emit(0); signal$emit(1); signal$emit(3)
signal$flush()
##'
signal$accumulator(function(prev, cur) {
  prev$x <- c(prev$x, cur$x)
  prev
})
signal$buffer()
signal$emit(0); signal$emit(1); signal$emit(3)
signal$flush()
## accessors
length(signal)
listeners(signal)
```

Index