

R documentation

of all in 'pkg/REventLoop/man'

June 2, 2003

R topics documented:

addIdle	1
eventLoop	3
REventLoop	4
runEventLoop	5
Sys.msleep	7

Index	8
--------------	----------

addIdle	<i>Register background or timed actions</i>
---------	---

Description

These provide an event-loop-independent manner to specify actions that should happen in the background when there are no events to process, or at a particular time in the future. The action can be given as a function, expression, call or any language object which can be evaluated. The mechanism evaluates the action at the appropriate time and can reschedule it if the result of the action is the logical value `TRUE`. This is a simple way to run actions indefinitely or to have them terminate when a given condition arises.

Usage

```
addIdle(f, data = NULL)
addTimer(interval, f, data = NULL)
```

Arguments

<code>f</code>	the action to be invoked. This can be a function, a call or an expression. If it is a function, one can use the <code>data</code> argument to arrange to have a value passed to the function to parameterize its actions.
<code>data</code>	an arbitrary S value that is passed to the action if it is a function. This can be used to parameterize a function so that it can be used for different actions that behave differently depending on this <code>data</code> argument. The value is passed as the first and only argument to the action function.

interval number of milli-seconds to wait before the action is invoked. While every effort to execute the action at that time is made, it is not possible to guarantee it will happen at exactly that time; just no sooner. Other conditions on the machine (e.g. load) may influence when the action is actually triggered.

Details

This uses the C-level API for the abstract event loop to register the S action with that event processor.

Value

An identifier for the registered action. This can be used to remove/unregister that action. (Will be implemented in the future!)

Author(s)

Duncan Temple Lang <duncan@research.bell-labs.com>

References

<http://www.omegahat.org/REventLoop>

See Also

[runEventLoop](#) [eventLoop](#)

Examples

```
# Print "Still here" every 5 seconds!
addTimer(5000, quote({cat("Still here\n"); TRUE}))

# Same thing but using a general function rather than
# an expression.
timerMessage <- function(msg) {
  cat(msg, "\n")
  TRUE
}
addTimer(5000, timerMessage, "Still here")

# Keep a count of the number of times
# action was called after each 0.8 seconds
# Uses a global variable: BAD!
assign("ctr", 0, env = globalenv())
addTimer(800, function() {
  myCtr <<- myCtr + 1
  cat("myCtr", myCtr, "\n")
  TRUE # run again
})

# Same thing with a closure so no global variables
# GOOD
counter <- function(ctr = 0) {
  function() {
```

```

        ctr <- ctr + 1
        cat("Counter", ctr, "\n")
        TRUE # run again
    }
}
addTimer(800, counter)

# Suppose nextPage() shows the next frame or display
# in an animation of plots. This walks through those
# pages, waiting 1 second between pages.
addTimer(1000, nextPage)

# In the background, download a collection of URLs
# This is probably too long an action to be suitable for
# an idle action.

idleDownload <- function(urls, dir = "/tmp") {

    # need to use a try() to catch any errors to correctly move past a
    # problematic URL.
    status <- download.file(urls[1], destfile = paste(dir, basename(urls),
        sep=.Platform$file.sep))

    if(status) {
        # Remove the recently downloaded url.
        urls <- urls[-1]
        cat("Fetched", urls, "\n")
    }

    return(length(urls))
}

addIdle(idleDownload(c("http://www.omegahat.org/REventLoop/index.html",
    "http://cran.r-project.org/index.html")))

```

eventLoop

Register an (internal) Event Loop implementation

Description

This sets an internal C variable that holds the default event loop that can be run using [runEventLoop](#). This doesn't actually run the event loop; it just sets which one will be used when it is needed. The registered event loop should be a C `R_EventLoop` data structure with methods for all of the different facilities that provides.

Usage

```
eventLoop(sym)
```

Arguments

`sym` the name or address of a C symbol identifying the event loop data structure that contains the different methods

Value

An object of class `REventLoop` representing the previously registered event loop.

Author(s)

Duncan Temple Lang <duncan@research.bell-labs.com>

References

<http://www.omegahat.org/REventLoop>

See Also

[runEventLoop](#)

Examples

```
eventLoop("R_TclEventLoop")

old <- eventLoop("R_TclEventLoop")
# run some Tk code
quitEventLoop()
eventLoop(old)
```

`REventLoop`

Create Event Loop Object

Description

This is a generic function that is used to create an S object that represents an R event loop. Such objects are of class `REventLoop` and have just a name and a reference to a C-level data structure that provides the methods that implement the particular event loop.

Usage

```
REventLoop(name, address)
```

Arguments

name	the user-level name to use for the event loop. This does not have to be in any way related to the name of the C variable used to define the event loop. It is just a name that can be used to remind the user of the type of the event loop. If address is not specified, this name is used as the name of the C symbol.
address	the address of the C symbol that implements the event loop functionality. This can be provided in several forms: the name of the C variable which is resolved using <code>getNativeSymbolInfo</code> ; an object of class <code>NativeSymbolInfo</code> previously resolved using <code>getNativeSymbolInfo</code> ; or an object of class <code>NativeSymbol</code> .

Value

An object of class `REventLoop`. This has two slots:

<code>name</code>	A user-understandable name for the event loop
<code>address</code>	the internal memory address of the C variable that implements the event loop

Author(s)

Duncan Temple Lang <duncan@research.bell-labs.com>

References

<http://www.omegahat.org/REventLoop>

See Also

[eventLoop](#) [runEventLoop](#)

Examples

```
eventLoop(REventLoop("R_TclEventLoop"))
REventLoop("R_GtkEventLoop")
REventLoop("GTK", getNativeSymbolInfo("R_GtkEventLoop", PACKAGE="REventLoop"))
```

<code>runEventLoop</code>	<i>Run/Terminate external event loop</i>
---------------------------	--

Description

`runEventLoop` runs the current event loop data structure registered with the `REventLoop` package, handling all the events it knows. When its termination or quit method is invoked, control is returned to the original event loop. This allows for nested event loops.

The event loop can be run in either of two ways: i) as an interactive replacement for the standard R event loop, providing a prompt for user input; and ii) as a non-interactive event loop that ensures that GUIs, etc. remain active while not allowing user-input from the console. The latter approach is used for implementing dialogs, or simply when there is a GUI from which we get user input and no console input is desired.

`quitEventLoop` terminates or exits the currently active event loop, returning control to the previous or outer event loop.

Usage

```
runEventLoop(repl = TRUE, loop = NULL, copy = TRUE)
quitEventLoop()
```

Arguments

- repl** a logical value indicating to run the REPL version of the event loop or simply the basic event-processing loop. The REPL handles emitting prompts and uses readline.
- loop** an optional object of class `REventLoop`. If this is specified and identifies a C symbol (in the address slot), this address is used as the event loop to run, rather than the current one on the top of the event loop stack.
- copy** a logical value indicating whether to copy the currently registered event sources and handlers from the standard R event loop. This is typically done just once when one has started an X11 graphics device before running an event loop from this package. If this is `TRUE` under these circumstances, the X11 event handler for the X11 device windows are registered with the new event loop and will still be responsive. Otherwise, no events will be processed for them under the new event loop. Similarly, if you start an event loop from this package and subsequently open an X11 device, it will not receive any events and hence won't resize or redraw.
- In the future, we will integrate this general event loop mechanism into the R source code and make the different devices use it when registering event handlers, processing locator requests, etc.

Details

See the `Docs/` directory of the `REventLoop` package.

Value

Empty list (`list()`).

Author(s)

Duncan Temple Lang <duncan@research.bell-labs.com>

References

<http://www.omegahat.org/REventLoop>

See Also

[quitEventLoop](#)

Examples

```
runEventLoop()
library(RGtk)
source(system.file("examples", "testgtk.S", package = "RGtk"))
```

Sys.msleep*Blocking sleep with active event handlers*

Description

This allows one to temporarily pause before the evaluation of the next R expression for a "fixed" period of time. This version of sleep differs from `Sys.sleep` in two ways:

- a it has a finer resolution for the time interval, using milliseconds rather than simply seconds.
- b while "sleeping", events are still processed. This means that windows, idle tasks, etc. are still responsive.

Usage

```
Sys.msleep(time)
```

Arguments

`time` the number of milliseconds to sleep

Details

This uses the general event loop mechanism. It starts by registering a timer callback. The callback is setup to terminate the active event loop. Then we immediately start a secondary instance of the event loop. This process all the events while waiting for the timer callback to terminate it. When this happens, the event loop terminates and returns control to the caller and hence `Sys.msleep` returns.

Value

TRUE

Author(s)

Duncan Temple Lang <duncan@research.bell-labs.com>

References

See Also

[runEventLoop](#)

Examples

```
runEventLoop()
if(require(RGtk)) {
  source(system.file("examples", "testgtk.S", package = "RGtk"))
  h$progressBar()
}
Sys.msleep(1000)
```

Index

*Topic **interface**

- addIdle, [1](#)
- eventLoop, [3](#)
- REventLoop, [4](#)
- runEventLoop, [5](#)
- Sys.msleep, [7](#)

addIdle, [1](#)

addTimer (*addIdle*), [1](#)

eventLoop, [2](#), [3](#), [5](#)

getNativeSymbolInfo, [4](#)

quitEventLoop, [6](#)

quitEventLoop (*runEventLoop*), [5](#)

REventLoop, [4](#)

runEventLoop, [2-4](#), [5](#), [5](#), [7](#)

Sys.msleep, [7](#)

Sys.sleep, [7](#)