Package 'txtq'

May 8, 2026

Title A Small Message Queue for Parallel Processes

Description This queue is a data structure that lets
parallel processes send and receive messages,
and it can help coordinate the work
of complicated parallel tasks.
Processes can push new messages to the queue,
pop old messages, and obtain a
log of all the messages ever pushed. File locking
preserves the integrity of the data even when
multiple processes access the queue simultaneously.

Version 0.2.4

License MIT + file LICENSE

URL https://github.com/wlandau/txtq

BugReports https://github.com/wlandau/txtq/issues

Imports base64url, filelock (>= 1.0.2), R6

Suggests parallel, purrr, testthat (>= 2.1.0)

Encoding UTF-8

Language en-US

RoxygenNote 7.1.1

NeedsCompilation no

Author William Michael Landau [aut, cre] (ORCID:
<https://orcid.org/0000-0003-1878-3253>),
Ian E. Fellows [ctb],
Eli Lilly and Company [cph]

Maintainer William Michael Landau <will.landau@gmail.com>

Repository CRAN

Date/Publication 2021-03-27 19:00:02 UTC

Contents

`txtq-package` *txtq-package*

Description

The txtq package is a small message queue for R.

Author(s)

William Michael Landau <will.landau@gmail.com>

References

https://github.com/wlandau/txtq

Examples

```
# See ?txtq for examples.
```

`R6_txtq` *R6 class for* `txtq` *objects*

Description

See the `txtq()` function for full documentation and usage.

Methods

Public methods:

- `R6_txtq$new()`
- `R6_txtq$path()`
- `R6_txtq$count()`
- `R6_txtq$total()`
- `R6_txtq$empty()`
- `R6_txtq$log()`
- `R6_txtq$list()`
- `R6_txtq$pop()`

- R6_txtq$push()
- R6_txtq$reset()
- R6_txtq$clean()
- R6_txtq$destroy()
- R6_txtq$validate()
- R6_txtq$import()
- R6_txtq$clone()

Method `new()`**:** Initialize a txtq.

Usage:

```
R6_txtq$new(path, use_lock_file = TRUE)
```

Arguments:

`path` Character string giving the file path of the queue. The `txtq()` function creates a folder at this path to store the messages.

`use_lock_file` Logical, whether to use a lock file for blocking operations. Should only be `FALSE` in specialized use cases with no parallel computing (for example, when a `txtq` is used as a database and accessed by only one process.)

Method `path()`**:** Get the txtq path.

Usage:

```
R6_txtq$path()
```

Method `count()`**:** Get the number of messages in the queue.

Usage:

```
R6_txtq$count()
```

Method `total()`**:** Get the number of messages in the database.

Usage:

```
R6_txtq$total()
```

Method `empty()`**:** Detect whether the txtq is empty.

Usage:

```
R6_txtq$empty()
```

Method `log()`**:** List the whole database.

Usage:

```
R6_txtq$log()
```

Method `list()`**:** List messages.

Usage:

```
R6_txtq$list(n = -1)
```

Arguments:

`n` Number of messages.

Method `pop()`**:** Pop messages.

Usage:

```
R6_txtq$pop(n = 1)
```

Arguments:

`n` Number of messages.

Method `push()`**:** Push messages.

Usage:

```
R6_txtq$push(title, message)
```

Arguments:

`title` Titles of the messages.

`message` Contents of the messages.

Method `reset()`**:** Reset the txtq.

Usage:

```
R6_txtq$reset()
```

Method `clean()`**:** Clean the txtq.

Usage:

```
R6_txtq$clean()
```

Method `destroy()`**:** Destroy the txtq.

Usage:

```
R6_txtq$destroy()
```

Method `validate()`**:** Validate the txtq.

Usage:

```
R6_txtq$validate()
```

Method `import()`**:** Import another txtq.

Usage:

```
R6_txtq$import(queue)
```

Arguments:

`queue` External txtq to import.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
R6_txtq$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

txtq

txtq *Create a message queue.*

Description

See the README at https://github.com/wlandau/txtq and the examples in this help file for instructions.

Usage

```
txtq(path, use_lock_file = TRUE)
```

Arguments

path	Character string giving the file path of the queue. The txtq() function creates a folder at this path to store the messages.
use_lock_file	Logical, whether to use a lock file for blocking operations. Should only be FALSE in specialized use cases with no parallel computing (for example, when a txtq is used as a database and accessed by only one process.)

NFS

As an interprocess communication tool, txtq relies on the filelock package to prevent race conditions. Unfortunately, filelock cannot prevent race conditions on network file systems (NFS), which means neither can txtq. In other words, on certain common kinds of clusters, txtq cannot reliably manage interprocess communication for processes on different computers. However, it can still serve as a low-tech replacement for a simple non-threadsafe database.

Examples

```
path <- tempfile() # Define a path to your queue.
q <- txtq(path) # Create a new queue or recover an existing one.
q$validate() # Check if the queue is corrupted.
list.files(q$path()) # The queue lives in this folder.
q$list() # You have not pushed any messages yet.
# Let's say two parallel processes (A and B) are sharing this queue.
# Process A sends Process B some messages.
# You can only send character vectors.
q$push(title = "Hello", message = "process B.")
q$push(
  title = c("Calculate", "Calculate"),
  message = c("sqrt(4)", "sqrt(16)")
)
q$push(title = "Send back", message = "the sum.")
# See your queued messages.
# The `time` is a formatted character string from `Sys.time()`
# indicating when the message was pushed.
q$list()
q$count() # Number of messages in the queue.
```

```
q$total() # Number of messages that were ever queued.
q$empty()
# Now, let's assume process B comes online. It can consume
# some messages, locking the queue so process A does not
# mess up the data.
q$pop(2) # Return and remove the first messages that were added.
# With those messages popped, we are farther along in the queue.
q$list()
q$count() # Number of messages in the queue.
q$list(1) # You can specify the number of messages to list.
# But you still have a log of all the messages that were ever pushed.
q$log()
q$total() # Number of messages that were ever queued.
# q$pop() with no arguments just pops one message.
# Call pop(-1) to pop all the messages at once.
q$pop()
# There are more instructions.
q$pop()
# Let's say Process B follows the instructions and sends
# the results back to Process A.
q$push(title = "Results", message = as.character(sqrt(4) + sqrt(16)))
# Process A now has access to the results.
q$pop()
# Clean out the popped messages
# so the database file does not grow too large.
q$push(title = "not", message = "popped")
q$count()
q$total()
q$list()
q$log()
q$clean()
q$count()
q$total()
q$list()
q$log()
# Optionally remove all messages from the queue.
q$reset()
q$count()
q$total()
q$list()
q$log()
# Destroy the queue's files altogether.
q$destroy()
# This whole time, the queue was locked when either Process A
# or Process B accessed it. That way, the data stays correct
# no matter who is accessing/modifying the queue and when.
#
# You can import a `txtq` into another `txtq`.
# The unpopped messages are grouped together
# and sorted by timestamp.
# Same goes for the popped messages.
q_from <- txtq(tempfile())
q_to <- txtq(tempfile())
```

```
q_from$push(title = "from", message = "popped")
q_from$push(title = "from", message = "unpopped")
q_to$push(title = "to", message = "popped")
q_to$push(title = "to", message = "unpopped")
q_from$pop()
q_to$pop()
q_to$import(q_from)
q_to$list()
q_to$log()
```

Index