

# Package ‘txtq’

October 15, 2019

**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** 0.2.0

**License** MIT + file LICENSE

**URL** <https://github.com/wlandau/txtq>

**BugReports** <https://github.com/wlandau/txtq/issues>

**Imports** base64url, filelock, R6

**Suggests** parallel, purrr, testthat (>= 2.1.0)

**Encoding** UTF-8

**Language** en-US

**RoxygenNote** 6.1.1

**NeedsCompilation** no

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

**Maintainer** William Michael Landau <[will.landau@gmail.com](mailto:will.landau@gmail.com)>

**Repository** CRAN

**Date/Publication** 2019-10-15 06:00:02 UTC

**R topics documented:**

txtq-package . . . . .	2
R6_txtq . . . . .	2
txtq . . . . .	3

<b>Index</b>	<b>5</b>
--------------	----------

---

txtq-package	<i>The txtq package is a small message queue for R.</i>
--------------	---

---

**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	<i>R6 class for txtq objects</i>
---------	----------------------------------

---

**Description**

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

**Usage**

```
R6_txtq
```

**Format**

An object of class R6ClassGenerator of length 24.

**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)
```

### Arguments

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

### 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

\*Topic **datasets**

R6\_txtq, [2](#)

R6\_txtq, [2](#)

txtq, [3](#)

txtq(), [2](#)

txtq-package, [2](#)