

Package: CanardAbsurd (via r-universe)

September 20, 2026

Title Checkpointed R Workflows over 'DuckDB' and 'Quack'

Version 0.0.1

Description Runs durable R tasks using 'DuckDB' for state and the 'Quack' extension for remote Structured Query Language (SQL) access. Provides database hosting, leased task claims, checkpoint replay, retries, durable sleeps, and a pull-based R worker. External side effects require application-level idempotency.

License GPL (>= 2)

URL <https://rgenomicsetl.github.io/CanardAbsurd/>,
<https://github.com/RGenomicsETL/CanardAbsurd>

BugReports <https://github.com/RGenomicsETL/CanardAbsurd/issues>

Encoding UTF-8

Depends R (>= 4.3.0)

Imports bit64, DBI, duckdb (>= 1.5.5), S7

Suggests callr, processx, tinytest, roxygen2, s7contract (>= 0.2.1),
withr, knitr, rmarkdown, parallelly, qs2, sakura

VignetteBuilder knitr

Config/Needs/website pkgdown, litedown

Config/Needs/graphics rsvg

Additional_repositories <https://rgenomicsetl.r-universe.dev>,
<https://sounkou-bioinfo.r-universe.dev>

Roxygen list(markdown = TRUE)

SystemRequirements The Quack extension matching the installed DuckDB runtime, for remote clients and database hosting, and the DuckDB httpfs extension for remote clients; the optional Canard coordinator extension for native lease maintenance

Config/roxygen2/version 8.0.0

Config/pak/sysreqs xz-utils

Repository <https://rgenomicsetl.r-universe.dev>
Date/Publication 2026-09-20 15:34:45 UTC
RemoteUrl <https://github.com/RGenomicsETL/CanardAbsurd>
RemoteRef HEAD
RemoteSha 1894ac0871743fba2b37cb4e118d43f5f73ff6b9

Contents

ca_cancel	2
ca_claim	3
ca_close	4
ca_complete	4
ca_conditions	5
ca_connect	6
ca_coordinator_start	6
ca_coordinator_status	7
ca_coordinator_stop	8
ca_fail	8
ca_heartbeat	9
ca_inspect	9
ca_open	10
ca_process	10
ca_result	12
ca_run	12
ca_runtime	13
ca_serve	13
ca_sleep	14
ca_spawn	15
ca_step	16
ca_tasks	16
ca_values	17
ca_work	18
Index	20

ca_cancel	<i>Cancel a nonterminal task</i>
-----------	----------------------------------

Description

Cancel a nonterminal task

Usage

ca_cancel(db, id)

Arguments

db	A handle returned by ca_open() , ca_serve() , or ca_connect() .
id	Stable task ID (1 to 256 characters), or NULL to generate a UUID.

Value

TRUE if cancelled, otherwise FALSE. Cancellation fences subsequent writes but cannot interrupt an external side effect already in progress.

ca_claim	<i>Claim an eligible task</i>
----------	-------------------------------

Description

Takes the next eligible task in queue under a lease. Expired leases are reclaimable until the failure budget runs out. Claims use optimistic DuckDB transactions, so callers outside [ca_work\(\)](#) handle canard_retryable themselves; see [ca_conditions\(\)](#). No transaction is held while a handler runs.

Usage

```
ca_claim(
  db,
  queue = "default",
  worker = paste0("R-", Sys.getpid()),
  lease_seconds = 30,
  task_names = NULL,
  reap_limit = 64L
)
```

Arguments

db	A handle returned by ca_open() , ca_serve() , or ca_connect() .
queue	Queue name.
worker	Worker label used for inspection; lease tokens establish ownership.
lease_seconds	Positive lease duration in seconds. Long-running handlers must explicitly heart-beat before this interval expires.
task_names	Optional character vector of handler names this worker supports.
reap_limit	Maximum exhausted, expired leases to mark failed before each claim. Batching bounds contention with other workers; it does not limit recovery of leases that still have failure budget.

Value

An S7 CanardTask, or NULL when no eligible task exists. Properties id, name, input, and attempt describe the claim; db, token, and lease_seconds are used by task operations. Treat claims as process-local.

ca_close	<i>Close a client or server</i>
----------	---------------------------------

Description

Stops a server handle's Quack endpoint, stops a loaded native coordinator, and closes the DuckDB connection. Every step is attempted even if an earlier one fails, and the failures are raised together as `canard_close_error`.

Usage

```
ca_close(db)
```

Arguments

db	A handle returned by <code>ca_open()</code> , <code>ca_serve()</code> , or <code>ca_connect()</code> .
----	--

Details

After a statement on the connection has failed, DuckDB R holds the database, and its file lock, until R garbage-collects that statement; only then can another process open a file-backed database.

Value

NULL, invisibly. Closing an already closed handle is harmless.

ca_complete	<i>Finish a claimed task</i>
-------------	------------------------------

Description

Finish a claimed task

Usage

```
ca_complete(task, result = NULL)
```

Arguments

task	A claim returned by <code>ca_claim()</code> or passed to an R handler.
result	An R value supported by <code>ca_values</code> .

Value

NULL, invisibly. Stale claims raise `canard_lease_lost`.

ca_conditions

Workflow conditions

Description

CanardAbsurd signals R conditions with machine-readable subclasses and fields. Errors inherit from canard_error, error and condition.

Details

- canard_input_error: invalid arguments, rejected before any database effect. input_class names the S7 constructor and parent holds the validation failure. Its subclass canard_value_error reports an unsupported value; see [ca_values](#).
- canard_storage_error: a failed SQL operation, carrying operation, attempts, elapsed seconds and the DBI condition in parent. Its subclasses are canard_conflict (a known write conflict, whose message_based flag is TRUE when the compatibility adapter below classified it), canard_lease_lost (the claim is gone; adds id and attempt), canard_restore_error (a stored value would not decode as its recorded R type; adds id), canard_failure_recording_error (a handler failed and recording it failed too, kept in parent and persistence_error), and canard_protocol_error.
- canard_spawn_conflict: the id already holds a different submission.
- canard_result_error: the task is unknown or unfinished; carries id and state. See [ca_result\(\)](#).
- canard_replay_error: a step name repeated within an attempt, or its kind changed; carries id and name.
- canard_process_error: a command from [ca_process\(\)](#) failed or timed out; carries id, status (NA on timeout) and the directory holding logs.
- canard_close_error: [ca_close\(\)](#) released everything it owns, but some step failed; errors lists them and parent is the first.
- canard_schema_error carries the unsupported version, canard_extension_error the load failure in parent, and canard_coordinator_error and canard_connection_error report a coordinator called on the wrong or closed handle.

Two conditions are notifications rather than errors. canard_suspended unwinds to [ca_run\(\)](#) once [ca_sleep\(\)](#) has saved a suspension, so an ordinary error handler does not swallow it. canard_retryable announces a known write conflict before it is raised: a withCallingHandlers() handler may wait and invoke the canard_retry restart to repeat that one SQL statement, never an R callback. [ca_work\(\)](#) does this by default and signals canard_conflict_retry (conflict, attempts, delay) before each retry; conflict_retries = 0 leaves the policy to you. Handlers that return without restarting, and errors they raise, propagate to the caller.

Unless given on_result, [ca_work\(\)](#) warns about a handler failure with canard_task_failed, keeping the condition in parent and the attempt record in outcome.

Released DuckDB R and Quack builds discard the structured type of some errors (<https://github.com/duckdb/duckdb-r/issues/2711>, <https://github.com/duckdb/duckdb-quack/pull/212>).

Until the installed versions carry both fixes, write conflicts are recognized from exact tested message text, which depends on upstream wording rather than a stable protocol. Structured non-transaction errors take precedence over text, and unrecognized messages never become retryable. See vignette("durability").

ca_connect	<i>Connect to a workflow server</i>
------------	-------------------------------------

Description

Connects to a `ca_serve()` endpoint. The client holds only an in-memory DuckDB connection, never a second writable handle to the server’s file. Quack clients also load DuckDB’s `httpfs` extension; install both explicitly, because connections never download extensions. With `extension = NULL` both come from DuckDB’s shared home; a path loads that build plus any `httpfs.duckdb_extension` beside it into session-private storage.

Usage

```
ca_connect(uri = "quack:127.0.0.1:9494", token, extension = NULL)
```

Arguments

uri	Quack endpoint, including a port when not using the default.
token	Shared authentication token. Quack currently requires at least four characters. A holder can run any SQL, including file access, with the server process’s privileges; give it only to trusted workers and services.
extension	Optional path to a compatible signed Quack extension. <code>NULL</code> loads an explicitly preinstalled quack extension.

Value

An `S7 CanardConnection` handle.

ca_coordinator_start	<i>Start native task maintenance</i>
----------------------	--------------------------------------

Description

Loads the optional `canard_coordinator` extension in the database-owner process and starts it marking abandoned tasks failed across every queue, on its own DuckDB connection and native thread. Workers keep doing the same for the queues they poll; this covers queues nobody polls.

Usage

```
ca_coordinator_start(
    db,
    extension = NULL,
    poll_milliseconds = 1000,
    reap_limit = 64
)
```

Arguments

db	A local handle returned by ca_open() or ca_serve() .
extension	Optional path to a compatible coordinator extension. NULL loads an explicitly preinstalled canard_coordinator extension.
poll_milliseconds	Delay between maintenance polls, in milliseconds.
reap_limit	Maximum expired tasks failed by one poll.

Details

It can start once per database instance. Call [ca_coordinator_stop\(\)](#) or [ca_close\(\)](#) to join the thread and release the connection: the extension holds that connection from the moment it loads, so closing the DBI connection directly leaves the database open, locked and maintained until the process exits.

No signed build is distributed, so a local build loads only into a database opened with `allow_unsigned_extensions = TRUE`; see [ca_open\(\)](#).

Value

TRUE if this call started the coordinator and FALSE if it was already running.

ca_coordinator_status *Inspect native task maintenance*

Description

Inspect native task maintenance

Usage

```
ca_coordinator_status(db)
```

Arguments

db	A local handle returned by ca_open() or ca_serve() .
----	--

Value

A one-row data frame containing lifecycle state, configuration, successful poll and reaped-task counts, and the latest polling error.

ca_coordinator_stop	<i>Stop native task maintenance</i>
---------------------	-------------------------------------

Description

Interrupts active coordinator SQL, joins the native thread, and closes the coordinator’s dedicated DuckDB connection. A stopped C API v1 coordinator cannot restart on the same database handle.

Usage

ca_coordinator_stop(db)

Arguments

db A local handle returned by [ca_open\(\)](#) or [ca_serve\(\)](#).

Value

TRUE if this call closed the coordinator connection and FALSE if it was already closed or was never loaded.

ca_fail	<i>Record a task failure</i>
---------	------------------------------

Description

Record a task failure

Usage

ca_fail(task, message, delay_seconds = 0)

Arguments

task A claim returned by [ca_claim\(\)](#) or passed to an R handler.
message Error text, stored without truncation.
delay_seconds Nonnegative delay before another claim is eligible.

Value

NULL, invisibly. The task becomes ready or terminal according to its failure budget. Stale claims raise canard_lease_lost.

ca_heartbeat	<i>Extend a live task lease</i>
--------------	---------------------------------

Description

Extend a live task lease

Usage

```
ca_heartbeat(task, seconds = task@lease_seconds)
```

Arguments

task	A claim returned by ca_claim() or passed to an R handler.
seconds	Positive duration from the database clock. An existing longer lease is preserved. An expired lease cannot be revived.

Value

NULL, invisibly. Stale claims raise `canard_lease_lost`.

ca_inspect	<i>Inspect a task</i>
------------	-----------------------

Description

Reads one task's metadata, then its stored values in a second query, so work saved in between appears on the next call. Stored values are immutable under the package protocol; editing payloads directly breaks that. Use [ca_tasks\(\)](#) to poll many tasks and [ca_result\(\)](#) for a result on its own.

Usage

```
ca_inspect(db, id)
```

Arguments

db	A handle returned by ca_open() , ca_serve() , or ca_connect() .
id	Stable task ID (1 to 256 characters), or NULL to generate a UUID.

Value

A named list containing task state, counters, timestamps, R input and result, and checkpoint records; NULL for an unknown ID. Step records contain kind and value; sleep records contain kind.

ca_open	<i>Open a local workflow database</i>
---------	---------------------------------------

Description

Opens a workflow database in this process, installing schema version 1 or checking the existing one. The handle owns its DBI connection; close it with `ca_close()`. Extensions are never downloaded, and the DuckDB home is session-private. A file-backed database has one owning process: others reach it through a Quack server instead of opening the file.

Usage

```
ca_open(path = ":memory:", allow_unsigned_extensions = FALSE)
```

Arguments

path	DuckDB database path, or ":memory:".
allow_unsigned_extensions	Development-only opt-in for loading an unsigned build of the native coordinator. It lets any SQL on this database load arbitrary native code into the owning process. On a served database, that includes every Quack token holder. Leave it FALSE in deployments.

Value

An S7 CanardConnection handle.

Examples

```
db <- ca_open()
id <- ca_spawn(db, "sum", list(x = 2, y = 3))
ca_work(db, list(sum = function(input, ctx) input$x + input$y), max_tasks = 1)
ca_inspect(db, id)$result
ca_close(db)
```

ca_process	<i>Supervise an external command within a task lease</i>
------------	--

Description

Runs a command in a child process while the worker keeps the task lease alive, so work longer than one lease is not reclaimed by another worker. Use it for external tools, or for blocking R computation run through Rscript: a timer inside the blocked process cannot renew anything.

Usage

```
ca_process(
  task,
  command,
  args = character(),
  directory,
  timeout = Inf,
  heartbeat_seconds = task@lease_seconds/3
)
```

Arguments

task	A claim returned by ca_claim() or passed to an R handler.
command	Executable to run.
args	Character vector of arguments.
directory	Directory that holds this step's attempt directories. Use one directory per task and step.
timeout	Positive seconds before the command is killed, or Inf.
heartbeat_seconds	Positive renewal interval, shorter than the task's lease_seconds.

Details

The lease is renewed before the command starts, so nothing begins on a lost claim, and then every heartbeat_seconds. The command runs in attempt-*<n>* under directory and writes stdout.log and stderr.log there, so relative output paths stay inside that attempt and no later attempt overwrites an earlier one's files. An existing attempt directory is an error. Publish results by returning paths from a [ca_step\(\)](#) callback: only the current lease holder can save that checkpoint.

Lease loss, including from [ca_cancel\(\)](#), a timeout, or an error or interrupt in the supervising code kills the command's process tree. Lease loss raises canard_lease_lost; a nonzero exit or timeout raises canard_process_error, which a handler records as an ordinary failure.

Overlap remains possible: a paused or partitioned worker can outlive its lease before the next heartbeat. Work submitted to Slurm or a similar backend needs that backend's own job identity, since supervising the submitting command does not supervise the job.

Value

The attempt directory path, after the command exits with status 0.

ca_result	<i>Retrieve a completed task's result</i>
-----------	---

Description

Reads only the result value, not the input or checkpoints.

Usage

```
ca_result(db, id)
```

Arguments

db	A handle returned by ca_open() , ca_serve() , or ca_connect() .
id	Stable task ID (1 to 256 characters), or NULL to generate a UUID.

Value

The stored R result. A task that is unknown or not completed raises `canard_result_error` with its id and state (NA when unknown).

ca_run	<i>Execute one claimed workflow attempt</i>
--------	---

Description

Calls `handler(task@input, task)` and records its result, or records a handler error as a task failure. Interrupts, lease loss and storage errors propagate instead, and do not use the failure budget. If recording a handler failure also fails, `canard_failure_recording_error` keeps both conditions; see [ca_conditions\(\)](#).

Usage

```
ca_run(task, handler, failure_delay = 0)
```

Arguments

task	A claim returned by ca_claim() or passed to an R handler.
handler	Function taking (input, ctx).
failure_delay	Nonnegative seconds before a failed task can be claimed again. This is separate from caller-managed SQL conflict retries.

Value

A list, invisibly, with id, claim attempt, attempt status ("completed", "suspended", or "failed"), persisted task state, stored R result, and original handler error. A failed attempt can leave the task ready for retry or terminally failed. Error text is stored without truncation; the R condition remains available in this outcome.

ca_runtime	<i>Inspect the executing database runtime</i>
------------	---

Description

Inspect the executing database runtime

Usage

```
ca_runtime(db)
```

Arguments

db A handle returned by `ca_open()`, `ca_serve()`, or `ca_connect()`.

Value

A one-row data frame with server-side DuckDB, loaded Quack, and schema versions. The Quack version is missing for local databases without Quack.

ca_serve	<i>Serve a workflow database over Quack</i>
----------	---

Description

The calling process owns the database and must stay alive. Quack answers SQL on its own server threads; handlers run in clients. Put non-local endpoints behind authenticated, TLS-terminating ingress. With `extension = NULL` the preinstalled quack extension is loaded from DuckDB's shared home; a path loads that build into session-private storage.

Usage

```
ca_serve(
  path,
  uri = "quack:127.0.0.1:9494",
  token,
  extension = NULL,
  allow_unsigned_extensions = FALSE
)
```

Arguments

path	DuckDB database path, or ":memory:".
uri	Quack endpoint, including a port when not using the default.
token	Shared authentication token. Quack currently requires at least four characters. A holder can run any SQL, including file access, with the server process's privileges; give it only to trusted workers and services.
extension	Optional path to a compatible signed Quack extension. NULL loads an explicitly preinstalled quack extension.
allow_unsigned_extensions	Development-only opt-in for loading an unsigned build of the native coordinator. It lets any SQL on this database load arbitrary native code into the owning process. On a served database, that includes every Quack token holder. Leave it FALSE in deployments.

Value

A local S7 CanardConnection handle owning the server.

ca_sleep	<i>Suspend a workflow until a database-clock deadline</i>
----------	---

Description

Saves a sleep checkpoint and releases the claim in one statement. Code after the sleep does not run in this attempt: after the deadline a worker reruns the handler from the top, replaying completed steps, and the sleep returns at once. Suspension travels as canard_suspended, a control-flow condition, so an ordinary error handler does not swallow it.

Usage

```
ca_sleep(task, name, seconds)
```

Arguments

task	A claim returned by <code>ca_claim()</code> or passed to an R handler.
name	Stable checkpoint name.
seconds	Nonnegative sleep duration in seconds.

Value

NULL, invisibly, on replay. The first call signals canard_suspended after persisting the suspension.

ca_spawn

*Submit a durable task***Description**

Reusing an ID with an identical submission is idempotent, even after the task has completed, so a lost response can be resubmitted safely. A different submission under that ID raises `canard_spawn_conflict`. Equality covers the native value and its R type descriptor.

Usage

```
ca_spawn(
  db,
  name,
  input = NULL,
  queue = "default",
  id = NULL,
  priority = 0L,
  max_failures = 3L
)
```

Arguments

<code>db</code>	A handle returned by <code>ca_open()</code> , <code>ca_serve()</code> , or <code>ca_connect()</code> .
<code>name</code>	Registered handler name.
<code>input</code>	An R value supported by <code>ca_values</code> . No package-imposed byte limit applies to input, results, checkpoints, or error text.
<code>queue</code>	Queue name.
<code>id</code>	Stable task ID (1 to 256 characters), or NULL to generate a UUID.
<code>priority</code>	Integer priority; larger values are claimed first.
<code>max_failures</code>	Number of failures, including expired leases, before a task becomes terminal. Durable sleeps do not consume this budget. Schema version 1 accepts values from 1 through 1,000,000.

Details

A constraint failure is resolved by looking the existing task up rather than writing again. If no task is found, the original storage error stands; if the lookup itself fails, its condition carries that error as `submission_error`.

Value

The task ID, a character scalar.

ca_step	<i>Execute or replay a named step</i>
---------	---------------------------------------

Description

Runs `fn` and saves its result under `name`, or returns the saved value when this task already has one. The callback runs outside any transaction, and its result is saved only once it returns, so external effects must be idempotent: lease tokens fence database writes, not other services. Give external requests stable business keys.

Usage

```
ca_step(task, name, fn)
```

Arguments

task	A claim returned by <code>ca_claim()</code> or passed to an R handler.
name	Stable checkpoint name.
fn	Zero-argument function returning a value supported by <code>ca_values</code> .

Details

Step names must be unique within an attempt and stable across deployments, so index repeated steps in a loop. Keep saved results readable by handlers that may resume existing tasks.

Value

The stored R value, both on initial execution and replay. Timestamps and durations have DuckDB microsecond precision. A cached NULL is distinct from an absent checkpoint.

ca_tasks	<i>List task metadata</i>
----------	---------------------------

Description

Reads state, counters, ownership, and timestamps for many tasks in one query, without inputs, results, or checkpoint values. Use it to poll a jobs view; use `ca_result()` or `ca_inspect()` for one task's values.

Usage

```
ca_tasks(db, queue = NULL, state = NULL, id = NULL, limit = 100L)
```

Arguments

db	A handle returned by <code>ca_open()</code> , <code>ca_serve()</code> , or <code>ca_connect()</code> .
queue	Optional queue name.
state	Optional character vector of task states: "ready", "running", "completed", "failed", or "cancelled".
id	Optional character vector of task IDs.
limit	Maximum number of rows, most recently updated first.

Value

A data frame with one row per task: id, queue, name, state, priority, attempt, failures, max_failures, available_at, worker, lease_until, error, created_at, and updated_at.

ca_values	<i>Native workflow values</i>
-----------	-------------------------------

Description

Inputs, results and checkpoint values are stored as DuckDB VARIANT values, queryable with field access, `variant_extract()` and casts. An R type descriptor stored beside each value records what SQL cannot: NULL versus a typed NA, vector versus list, data frame versus named list, and names, factor levels, time zones, row names and `I()` wrappers.

Details

Supported values are NULL; logical, integer, double, character and raw vectors; Date, POSIXct, difftime, factors, ordered factors and `bit64::integer64`; and lists and ordinary data frames composed of those. They map as:

- length-one logical, integer, double and character values to the matching DuckDB scalar type, and longer vectors to typed lists;
- raw vectors to BLOB, Date to DATE, POSIXct to TIMESTAMPTZ, difftime to INTERVAL, factors to VARCHAR, and `integer64` to BIGINT;
- named lists and data frames to objects, unnamed lists to arrays.

Timestamps and durations are rounded to DuckDB microsecond precision and must be finite or missing; dates must be whole finite days or missing. Double infinities and NaN survive, and factor levels must not be missing. Names of list elements and data frame columns become STRUCT fields, so they must be nonempty and unique ignoring ASCII case: A and a cannot share a container. Atomic vector names are unrestricted.

Anything else, including custom classes, matrices, environments and language objects, needs explicit conversion. The package never serializes objects for you, nor downloads extensions to convert them: store your own bytes as a raw vector and decode them in the handler, deserializing only trusted payloads. See `vignette("serialized-values")`. The mappings follow the declared-type approach of **Rducks**, which is not a dependency.

Values are read back through descriptor-typed projections rather than the driver's generic VARIANT decoder; `vignette("native-values")` explains why and lists the driver defects involved. Files need DuckDB storage compatibility v1.5.0. There is no package byte limit, though engine memory, disk and transport still apply.

An unsupported value raises `canard_value_error`, a `canard_input_error` carrying the conversion failure in parent. A step callback returning one has failed, and no checkpoint is saved for it.

ca_work

Pull and execute R tasks

Description

Claims and runs tasks whose names appear in handlers, one attempt at a time, until `max_tasks` or `idle_timeout` is reached. Scale out with more worker processes connected through Quack. Leases are renewed by `ca_step()`, `ca_sleep()` and `ca_heartbeat()` only; no background thread does it, so supervise long external commands with `ca_process()`. Losing a lease never ends the worker process.

Usage

```
ca_work(
  db,
  handlers,
  queue = "default",
  max_tasks = Inf,
  poll_seconds = 0.1,
  idle_timeout = Inf,
  lease_seconds = 30,
  worker = paste0("R-", Sys.getpid()),
  reap_limit = 64L,
  failure_delay = 0,
  conflict_retries = 8L,
  on_result = NULL
)
```

Arguments

db	A handle returned by <code>ca_open()</code> , <code>ca_serve()</code> , or <code>ca_connect()</code> .
handlers	Named list of functions taking (input, ctx).
queue	Queue name.
max_tasks	Maximum number of claimed attempts, including failures and suspensions. Inf runs without an attempt limit.
poll_seconds	Positive delay when no work is found.
idle_timeout	Nonnegative seconds without a claim before returning. Inf waits indefinitely. Zero drains currently eligible work.

lease_seconds	Positive lease duration in seconds. Long-running handlers must explicitly heart-beat before this interval expires.
worker	Worker label used for inspection; lease tokens establish ownership.
reap_limit	Maximum exhausted, expired leases to mark failed before each claim. Batching bounds contention with other workers; it does not limit recovery of leases that still have failure budget.
failure_delay	Nonnegative seconds before a failed task can be claimed again. This is separate from caller-managed SQL conflict retries.
conflict_retries	Nonnegative number of automatic retries for each statement aborted by a known write conflict. Zero installs no retry policy, leaving <code>canard_retryable</code> to the caller's own calling handler.
on_result	Optional function called with each <code>ca_run()</code> outcome after persistence. The default warns about handler failures with <code>canard_task_failed</code> . A supplied callback owns outcome reporting instead. Errors from the callback propagate to the caller.

Details

Competing workers pick the same eligible task, and DuckDB aborts the losing statement. The worker retries such a statement up to `conflict_retries` times, backing off from 10 ms to 0.5 s (about 1.6 seconds in total by default), repeating only the SQL and never an R callback, and signalling `canard_conflict_retry` beforehand; see `ca_conditions()`. Jitter comes from the clock and process ID, leaving R's random number stream untouched. An exhausted budget raises `canard_conflict`. Lease loss and transport failures are never retried.

Value

The number of executed attempts, invisibly. Use `on_result` to collect outcomes without accumulating an unbounded result list inside the worker.

Index

ca_cancel, 2
ca_cancel(), 11
ca_claim, 3
ca_claim(), 4, 8, 9, 11, 12, 14, 16
ca_close, 4
ca_close(), 5, 7, 10
ca_complete, 4
ca_conditions, 5
ca_conditions(), 3, 12, 19
ca_connect, 6
ca_connect(), 3, 4, 9, 12, 13, 15, 17, 18
ca_coordinator_start, 6
ca_coordinator_status, 7
ca_coordinator_stop, 8
ca_coordinator_stop(), 7
ca_fail, 8
ca_heartbeat, 9
ca_heartbeat(), 18
ca_inspect, 9
ca_inspect(), 16
ca_open, 10
ca_open(), 3, 4, 7–9, 12, 13, 15, 17, 18
ca_process, 10
ca_process(), 5, 18
ca_result, 12
ca_result(), 5, 9, 16
ca_run, 12
ca_run(), 5, 19
ca_runtime, 13
ca_serve, 13
ca_serve(), 3, 4, 6–9, 12, 13, 15, 17, 18
ca_sleep, 14
ca_sleep(), 5, 18
ca_spawn, 15
ca_step, 16
ca_step(), 11, 18
ca_tasks, 16
ca_tasks(), 9
ca_values, 4, 5, 15, 16, 17
ca_work, 18
ca_work(), 3, 5