Package 'regulog'

July 16, 2026

Title Tamper-Evident Audit Logging for Regulated Environments

Version 0.2.1

Description Provides tamper-evident, hash-chained audit logging for analytical applications. Every log entry is linked via an 'SHA-256' hash to its predecessor, making insertions, deletions, and modifications detectable. Covers user attribution, timestamp integrity, mandatory reason capture, chain verification, structured export, and 'shiny' session instrumentation. For more details see <https://reprostats.org/regulog/>. Suitable for any context where accountability and traceability matter: regulated environments (21 CFR Part 11, EU Annex 11), internal tooling, data pipelines, and multi-user 'shiny' applications. Ships with optional qualification scripts (IQ, OQ, PQ) for use in validated computerised systems.

License MIT + file LICENSE

Encoding UTF-8

Language en-GB

RoxygenNote 7.3.3

Imports digest, jsonlite, utils

Suggests covr, dplyr, ggplot2, haven, knitr, readr, rmarkdown, shiny, testthat (>= 3.0.0), tidyr, withr

Config/testthat/edition 3

Config/Needs/website dplyr, ggplot2, tidyr

VignetteBuilder knitr

URL https://reprostats.org/regulog/, https://github.com/repro-stats/regulog

BugReports https://github.com/repro-stats/regulog/issues

NeedsCompilation no

Author Ndoh Penn [aut, cre] (ORCID: <https://orcid.org/0009-0003-9054-465X>)

Maintainer Ndoh Penn <ndohpenn9@gmail.com>

Repository CRAN

Date/Publication 2026-07-16 13:00:02 UTC

Contents

`regulog-package` *regulog: Tamper-Evident Audit Logging for R*

Description

Every analytical action taken in a consequential R environment should be documented — who did it, what they did, when, and why. In practice, almost none of it is.

`regulog` fills that gap. It records every action, change, note, and decision into a tamper-evident, hash-chained audit trail stored as newline-delimited JSON. Every entry is attributed to a named user, time-stamped in UTC, and linked to the previous entry via SHA-256 — so any modification after the fact, however subtle, is detectable by `verify_log()`.

The design is intentionally general. `regulog` works equally well in regulated pharmaceutical environments (21 CFR Part 11, EU Annex 11), internal data pipelines, multi-user Shiny applications, and any other context where accountability and traceability matter. The IQ/OQ/PQ qualification scripts are available for validated computerised systems but are not a prerequisite for general use.

Workflow

Step 1 — Initialise the session

```
log <- regulog_init(
  app     = "primary-analysis",
  version = "1.0.0",
  user    = "jsmith",
  path    = "logs/trial001_audit.rlog"
)
```

Step 2 — Log actions, changes, notes, and decisions

```
log_action(log, "data_read", "adsl.sas7bdat",
           "Reading ADSL for primary efficacy analysis")

log_change(log, object = "param_alpha", field = "value",
           before = "0.05", after = "0.025",
           reason = "Updated per protocol amendment 2 (2026-05-01)")

log_note(log, "Outlier in subject 042 at Week 16 retained per SAP
              section 8.3 after discussion with medical monitor")
```

Step 3 — Log data reads, explicitly or scoped to a block

```
# Single read
adsl <- rl_read(log, haven::read_sas, "data/adsl.sas7bdat")

# Scoped block — read() calls inside resolve to `log` automatically
with_log(log, {
  adae <- read(haven::read_sas, "data/adae.sas7bdat")
  adlb <- read(haven::read_sas, "data/adlb.sas7bdat")
})
```

Step 4 — Apply an electronic signature

```
log_signature(log,
  "I certify that this analysis is accurate and complete, conducted
   in accordance with SAP version 2.0 dated 2026-05-01"
)
```

Step 5 — Verify, query, and export

```
# Verify tamper integrity
verify_log(log)

# Query entries
filter_log(log, type = "SIGNATURE")
filter_log(log, action = "data_read", from = "2026-06-01")

# Export
export_audit_trail(log, format = "csv", signed = TRUE,
                   path = "outputs/audit_trail_TRIAL001.csv")
```

Key functions

Function	Purpose
regulog_init()	Initialise an audit logging session
log_action()	Log a discrete action (approval, export, run, etc.)
log_change()	Log a before/after field change

log_note()	Log a free-text annotation or analytical decision
log_signature()	Apply an electronic signature
rl_read()	Explicit, logged read of any data source
with_log()	Scoped logging: read() calls inside the block log automatically
verify_log()	Verify the SHA-256 hash chain integrity
filter_log()	Query log entries by type, user, action, or date
export_audit_trail()	Export to CSV or JSON, with optional signing
regulog_shiny_init()	Initialise inside a Shiny server function
regulog_observer()	Auto-log Shiny reactive input events

The hash chain

Every entry stores the SHA-256 hash of all prior entries:

```
h_0 = SHA256("GENESIS" | app | version | timestamp)
h_n = SHA256(entry_id | timestamp | app | version | user | type |
             <payload fields> | h_{n-1})
```

Altering any field in any entry — including the timestamp or reason — breaks the chain from that entry forward. verify_log() recomputes every hash and reports the first broken link. This works offline, from the raw .rlog file, without an active R session.

Entry types

Type	Created by	Purpose
ACTION	log_action()	Discrete events: reads, runs, approvals
CHANGE	log_change()	Before/after field modifications
NOTE	log_note()	Free-text decisions and annotations
SIGNATURE	log_signature()	Named, dated, meaningful sign-off

Use in regulated environments

For regulated pharmaceutical and clinical contexts, regulog addresses the following requirements. IQ/OQ/PQ qualification scripts are available to generate a validation dossier for your specific environment.

Regulation	Clause	Coverage
21 CFR Part 11	§11.10(e)	Hash-chained, time-stamped, user-attributed entries
21 CFR Part 11	§11.10(b)	export_audit_trail() — CSV and JSON
21 CFR Part 11	§11.10(c)	Append-only .rlog format
21 CFR Part 11	§11.100	log_signature() — named signer identity

21 CFR Part 11	§11.200	Signature components: identity, timestamp, meaning
EU Annex 11	Clause 9	Date, time, user, and action on every entry
EU Annex 11	Clause 11	`verify_log()` — periodic integrity verification

```
source(system.file("validation/IQ_regulog.R", package = "regulog"))
source(system.file("validation/OQ_regulog.R", package = "regulog"))
source(system.file("validation/PQ_regulog.R", package = "regulog"))
```

Author(s)

Maintainer: Ndoh Penn <ndohpenn9@gmail.com> (ORCID)

See Also

Useful links:

- https://reprostats.org/regulog/
- https://github.com/repro-stats/regulog
- Report bugs at https://github.com/repro-stats/regulog/issues

---

as.data.frame.regulog *Convert a regulog object to a data frame*

Description

Coerces the entries list of a `regulog` object into a flat `data.frame`, one row per entry (genesis record excluded). Columns match those produced by `export_audit_trail()` with `format = "csv"`.

Usage

```
## S3 method for class 'regulog'
as.data.frame(x, ...)
```

Arguments

`x`	A `regulog` object.
`...`	Unused; for S3 compatibility.

Details

Called implicitly by `filter_log()` and useful for direct inspection.

Value

A `data.frame` with columns `entry_id`, `timestamp`, `app`, `app_version`, `user`, `type`, `action`, `object`, `field`, `before`, `after`, `reason`, `text`, `meaning`, `entry_hash`, `prev_hash`.

Examples

```
log <- regulog_init(app = "analysis", version = "1.0", user = "jsmith")
log_action(log,
  action = "run",
  object = "primary.R",
  reason = "Primary model fitted"
)
log_note(log, "Outlier in subject 042 retained per SAP")

as.data.frame(log)
```

---

export_audit_trail *Export the audit trail*

Description

Serialises log entries to CSV or JSON, with optional date filtering. Use `signed = TRUE` to run chain verification and stamp the integrity result into the export — useful for handoffs, audits, or archival.

Usage

```
export_audit_trail(
  log,
  format = c("csv", "json"),
  from = NULL,
  to = NULL,
  path = NULL,
  signed = FALSE,
  include_genesis = FALSE
)
```

Arguments

`log`	A `regulog` object or a path to a `.rlog` file.
`format`	Character. `"csv"` or `"json"`.
`from`	Character or `NULL`. Include entries on or after this date (ISO-8601, e.g. `"2026-01-01"`).
`to`	Character or `NULL`. Include entries on or before this date.
`path`	Character or `NULL`. Output file path. If `NULL`, returns the data without writing to disk.
`signed`	Logical. If `TRUE`, verify the chain and include `chain_intact` and `verified_at` fields in the export.
`include_genesis`	Logical. Include the genesis record. Default `FALSE`.

Details

CSV column layout:

Column	Description
`entry_id`	Monotone sequence number
`timestamp`	ISO-8601 UTC
`app`	Application name
`app_version`	Application version
`user`	Acting user identity
`type`	`ACTION`, `CHANGE`, `NOTE`, or `SIGNATURE`
`action`	Action label (`ACTION` entries)
`object`	Target of the action or change
`field`	Field name (`CHANGE` entries)
`before`	Prior value (`CHANGE` entries)
`after`	New value (`CHANGE` and `SIGNATURE` entries)
`reason`	Justification (`ACTION`, `CHANGE`, `NOTE` entries)
`text`	Free-text annotation (`NOTE` entries)
`meaning`	Signature meaning (`SIGNATURE` entries)
`entry_hash`	SHA-256 of this entry
`prev_hash`	SHA-256 of prior entry
`chain_intact`	`TRUE`/`FALSE` (signed exports only)
`verified_at`	ISO-8601 UTC of export (signed exports only)

Value

A data frame (CSV) or list (JSON), invisibly.

Examples

```
log <- regulog_init(app = "my-app", user = "jsmith")
log_action(log,
  action = "approved",
  object = "model_v3",
  reason = "Metrics passed threshold"
)
df <- export_audit_trail(log, format = "csv")


export_audit_trail(log,
  format = "csv",
  from   = "2026-01-01",
  signed = TRUE,
  path   = tempfile(fileext = ".csv")
)
```

`filter_log` *Filter audit log entries*

Description

Extracts a subset of entries from a `regulog` object or a `.rlog` file as a plain `data.frame`. All filter arguments are optional — omitting all returns every entry.

Usage

```
filter_log(
  log,
  type = NULL,
  user = NULL,
  action = NULL,
  from = NULL,
  to = NULL
)
```

Arguments

`log`	A `regulog` object **or** a path to a `.rlog` file.
`type`	Character vector of entry types to keep: `"ACTION"`, `"CHANGE"`, `"NOTE"`, `"SIGNATURE"`. `NULL` returns all types.
`user`	Character vector of user identifiers to keep. `NULL` returns all users.
`action`	Character vector of action values to keep (e.g. `"approved"`, `"data_read"`). `NULL` returns all actions.
`from`	Start of the time window. ISO 8601 string (`"2026-06-01"`) or `Date`. `NULL` applies no lower bound.
`to`	End of the time window. Same format as `from`. Inclusive. `NULL` applies no upper bound.

Value

A `data.frame` of matching entries, sorted by `entry_id`. Returns a zero-row data frame when nothing matches.

See Also

`as.data.frame.regulog()`, `export_audit_trail()`, `verify_log()`

Examples

```
log <- regulog_init(app = "analysis", version = "1.0", user = "jsmith")
log_action(log,
  action = "run",
  object = "primary.R",
  reason = "Primary model fitted"
)
log_note(log, "Outlier in subject 042 retained per SAP")
log_action(log,
  action = "export",
  object = "results.csv",
  reason = "Sent to sponsor"
)
log_signature(log, "Analysis complete and accurate per SAP v2")

# All entries as a data frame
filter_log(log)

# Only signatures
filter_log(log, type = "SIGNATURE")

# Actions and notes by a specific user
filter_log(log, type = c("ACTION", "NOTE"), user = "jsmith")

# Entries within a date range
filter_log(log, from = "2026-06-01", to = "2026-12-31")

# Works directly on a .rlog file — no live session needed

tmp <- tempfile(fileext = ".rlog")
log2 <- regulog_init(app = "analysis", version = "1.0", user = "jsmith",
  path = tmp)
log_action(log2,
  action = "run",
  object = "primary.R",
  reason = "Primary model fitted"
)
filter_log(tmp, type = "ACTION")
```

---

log_action *Log a discrete action in the audit trail*

Description

Records a user action (approval, rejection, sign-off, deployment, export, etc.) as a tamper-evident, hash-chained entry in the audit log.

Usage

```
log_action(log, action, object, reason, user = log$user)
```

Arguments

log	A regulog object from regulog_init().
action	Character. What happened (e.g. "approved", "deployed", "rejected", "exported").
object	Character. What it happened to (filename, model ID, record ID, pipeline step, etc.).
reason	Character. **Mandatory.** Why it happened. No default.
user	Character. Override the session user for this entry. Defaults to the user set at regulog_init().

Value

The regulog object, invisibly (pipe-friendly).

Examples

```
log <- regulog_init(app = "my-app", user = "jsmith")
log_action(log,
  action = "approved",
  object = "model_v3",
  reason = "Validation metrics passed agreed threshold"
)
```

---

log_change *Log a before/after field change in the audit trail*

Description

Records a data modification with both prior and new values. Use this whenever a specific field on a record is changed and you need a full history of what it was, what it became, and why.

Usage

```
log_change(log, object, field, before, after, reason, user = log$user)
```

Arguments

log	A regulog object from regulog_init().
object	Character. The record being modified (e.g. "user_42", "config.yaml", "experiment_7").
field	Character. The field that changed (e.g. "status", "threshold").
before	The value before the change (coerced to character).
after	The value after the change (coerced to character).
reason	Character. **Mandatory.** Why the change was made. No default.
user	Character. Override the session user. Defaults to session user.

Value

The regulog object, invisibly.

Examples

```
log <- regulog_init(app = "my-app", user = "jsmith")
log_change(log,
  object = "experiment_7",
  field  = "learning_rate",
  before = "0.01",
  after  = "0.001",
  reason = "Loss diverging at 0.01 — reduced per tuning protocol"
)
```

---

log_note *Log a free-text note in the audit trail*

Description

Records a NOTE entry — a free-text annotation that adds context, intent, or an observation without requiring a discrete action verb or a before/after value. Use it to document analytical decisions, assumptions, or rationale that do not fit log_action() or log_change().

Usage

```
log_note(log, text)
```

Arguments

log	A regulog object returned by regulog_init() or regulog_shiny_init().
text	The note text. Cannot be blank or whitespace-only.

Details

Like all regulog entries, text is mandatory with no default and is included in the hash chain, making it tamper-evident.

Value

The regulog object, invisibly (pipe-friendly).

See Also

log_action(), log_change(), log_signature()

Examples

```
log <- regulog_init(app = "analysis", version = "1.0", user = "jsmith")

log_note(log, "Baseline window defined as Day -1 to Day 1 per protocol v3 §5.2")
log_note(log, "Outlier in subject 042 discussed with medical monitor — retained per SAP")
```

`log_signature` *Apply an electronic signature to the audit trail*

Description

Records a SIGNATURE entry capturing the signer identity (from the session user), UTC timestamp, number of prior entries covered, and the stated meaning of the signature. Addresses 21 CFR Part 11 §11.100 / §11.200 requirements:

Usage

```
log_signature(log, meaning)
```

Arguments

`log`	A `regulog` object returned by `regulog_init()` or `regulog_shiny_init()`.
`meaning`	The meaning of the signature — what you are certifying. Cannot be blank. Example: `"I certify that this analysis is accurate and complete per SAP version 2.0"`.

Details

- **Identity** — resolved from the session user set in `regulog_init()`
- **Date and time** — UTC timestamp generated automatically at call time
- **Meaning** — the `meaning` argument; mandatory, cannot be blank
- **Coverage** — number of prior entries in the session, recorded automatically

The SIGNATURE entry is part of the hash chain: any tampering with entries preceding the signature, or with the signature entry itself, is detectable by `verify_log()`.

Value

The `regulog` object, invisibly (pipe-friendly).

See Also

`log_action()`, `log_note()`, `verify_log()`

Examples

```
log <- regulog_init(app = "analysis", version = "1.0", user = "jsmith")
log_action(
  log, "run", "primary_analysis.R",
  "Primary ANCOVA model executed per SAP section 6.1"
)

log_signature(
  log,
  "I certify that this analysis is accurate and complete per SAP version 2.0"
)
```

`regulog_init` *Initialise a regulog audit log session*

Description

Creates a new audit log session object. Subsequent calls to `log_action()`, `log_change()`, `log_note()`, and `log_signature()` append hash-chained entries. If `path` is supplied, entries are written to a newline-delimited JSON file (`.rlog`).

Usage

```
regulog_init(
  app,
  version = "unknown",
  user = Sys.info()[["user"]],
  path = NULL,
  hash_algo = "sha256"
)
```

Arguments

`app`	Character. Application or system name (e.g. `"data-pipeline"`, `"review-tool"`, `"ml-trainer"`).
`version`	Character. Application version string.
`user`	Character. Identity of the acting user. Defaults to `Sys.info()[["user"]]`. In Shiny, pass `session$user`.
`path`	Character or `NULL`. Path for persistent storage. If `NULL`, the log is in-memory only (suitable for development / testing).
`hash_algo`	Character. Algorithm passed to `digest::digest()`. Defaults to `"sha256"`. Do not change once a log file is in use.

Details

Entry structure:

Every entry written to disk is a JSON object on a single line:

```
{
  "entry_id":    1,
  "timestamp":   "2026-06-18T14:32:01.123456Z",
  "app":         "my-app",
  "app_version": "1.0.0",
  "user":        "jsmith",
  "type":        "ACTION",
  "action":      "approved",
  "object":      "model_v3",
  "reason":      "Validation metrics passed threshold",
  "prev_hash":   "e3b0c44298fc1c149afb...",
  "entry_hash":  "a87ff679a2f3e71d9181..."
}
```

The flat structure is intentional: the log should be inspectable with a text editor, without any specialist software.

Hash chain:

Each `entry_hash` is SHA-256 of a canonical string encoding all fields plus `prev_hash`. Altering any field — including the timestamp or reason — invalidates the hash and all subsequent chain links, detectable by `verify_log()`.

What the chain captures:

Property	Implementation
Who acted	`user` field on every entry
What happened	`action` + `object` fields
When	ISO-8601 UTC timestamp, microsecond resolution
Why	Mandatory `reason` — no default
What changed	`before`/`after` in `log_change()`
Tamper evidence	SHA-256 hash chain; verified by `verify_log()`
Portable export	`export_audit_trail()` to CSV or JSON

Value

An S3 object of class `"regulog"` (an environment).

Examples

```
log <- regulog_init(
  app     = "my-app",
  version = "1.0.0",
  user    = "jsmith"
)
log
```

`regulog_observer` *Create a logging observer for a reactive Shiny input*

Description

Wraps `shiny::observeEvent()` to log an action whenever `eventExpr` fires. Reduces boilerplate when many UI events need to be audited.

Usage

```
regulog_observer(log, session, eventExpr, action, object, reason, ...)
```

Arguments

`log`	A `regulog` object.
`session`	The Shiny session object.
`eventExpr`	Reactive expression to observe.
`action`	Character. Action label.
`object`	Character or reactive. The object acted upon.
`reason`	Character or reactive. Business justification.
`...`	Additional arguments passed to `log_action()`.

Value

A Shiny observer (invisibly).

Examples

```
if(interactive()){
regulog_observer(log, session,
  eventExpr = input$approve,
  action    = "approved",
  object    = reactive(input$selected_dataset),
  reason    = reactive(input$reason_text)
)
}
```

`regulog_shiny_init` *Initialise a regulog session inside a Shiny server*

Description

A thin wrapper around `regulog_init()` that resolves the authenticated user from `session$user` (set by Shiny Server Pro / Posit Connect) and automatically logs `session_start` and `session_end` events.

Usage

```
regulog_shiny_init(
  session,
  app,
  version = "unknown",
  path = NULL,
  hash_algo = "sha256"
)
```

Arguments

`session`	The Shiny `session` object.
`app`	Character. Application name.
`version`	Character. Application version.
`path`	Character or `NULL`. Persistent log file path. When `NULL`, a per-session temp file is created (suitable for development only; logs will be lost when the session ends).
`hash_algo`	Character. Hashing algorithm. Defaults to `"sha256"`.

Details

User resolution:

`session$user` is the authenticated identity set by Shiny Server Pro or Posit Connect. In open deployments where authentication is not configured, this will be `NULL` or `""`. `regulog_shiny_init()` falls back to `Sys.info()[["user"]]` in that case, with a warning.

Session instrumentation:

Two entries are added automatically:

- `session_start` when `regulog_shiny_init()` is called
- `session_end` via `shiny::onSessionEnded()`

These bracket all user-driven entries, giving regulators a complete picture of each session lifecycle.

Recommended pattern:

```
server <- function(input, output, session) {

  log <- regulog_shiny_init(
    session = session,
    app     = "my-app",
    version = "1.2.0",
    path    = "/logs/audit.rlog"
  )

  observeEvent(input$approve, {
    log_action(log,
      action = "approved",
      object = input$dataset,
      reason = input$reason
    )
  })
}
```

Value

A `regulog` object with the log tied to the authenticated session user.

Examples

```
if(interactive()){
library(shiny)
library(regulog)

server <- function(input, output, session) {
  log <- regulog_shiny_init(
    session = session,
    app     = "my-app",
    version = "1.0.0",
    path    = "logs/audit.rlog"
  )
  observeEvent(input$submit, {
    log_action(log,
      action = "submitted",
      object = input$form_id,
      reason = input$justification
    )
  })
}

shinyApp(ui = fluidPage(), server = server)
}
```

`rl_read` *Log a data read operation*

Description

Calls `reader` with `...`, then records the call as a `data_read` ACTION entry. Unlike namespace patching, `rl_read()` wraps the call explicitly — no package internals are modified, and behaviour is identical whether called from a single script or concurrently across multiple Shiny sessions.

Usage

```
rl_read(log, reader, ...)
```

Arguments

`log`	A `regulog` object from `regulog_init()` or `regulog_shiny_init()`.
`reader`	A function that reads data, e.g. `haven::read_sas`, `readr::read_csv`, `data.table::fread`, `utils::read.csv`.
`...`	Arguments passed to `reader`.

Details

The path/file recorded in the audit entry is resolved as follows:

1. A named argument in `...` called `file`, `path`, or `data_file`.
2. The first unnamed argument in `...`, if any.
3. `"unknown"`, if neither is found.

This avoids the failure mode of positional-only extraction, where a reordered named call (e.g. `read_csv(col_types = "ccd", file = "x.csv")`) would otherwise record the wrong value.

Value

The result of calling `reader(...)`.

See Also

`with_log()`

Examples

```
log <- regulog_init(app = "pipeline", version = "1.0", user = "jsmith")

## Not run:
adsl <- rl_read(log, haven::read_sas, "data/adsl.sas7bdat")
adae <- rl_read(log, readr::read_csv, file = "data/adae.csv")

## End(Not run)
```

verify_log *Verify the integrity of an audit log chain*

Description

Recomputes every entry hash and confirms each matches the stored value, and that each `prev_hash` matches its predecessor's `entry_hash`. Any discrepancy indicates tampering or corruption.

Usage

```
verify_log(log, verbose = TRUE)
```

Arguments

`log`	A `regulog` object **or** a character path to a `.rlog` file.
`verbose`	Logical. Print a summary. Defaults to `TRUE`.

Details

Verification algorithm:

For each entry *i* (excluding the genesis record):

1. Reconstruct `hash_input` from the stored fields in canonical order.
2. Recompute `digest(hash_input, algo = hash_algo)`.
3. Assert `computed == entry$entry_hash` (content integrity).
4. Assert `entry$prev_hash == entry[i-1]$entry_hash` (chain continuity).

Step 3 failure: the entry's content was modified after writing. Step 4 failure: entries were inserted, deleted, or reordered.

Value

A list (invisibly) with components:

`intact` Logical. `TRUE` if the chain is unbroken.

`n_entries` Integer. Number of data entries verified (genesis excluded).

`first_broken` Integer or `NA`. `entry_id` of the first invalid entry.

`errors` Character vector of error descriptions.

Examples

```
log <- regulog_init(app = "my-app", user = "jsmith")
log_action(log,
  action = "approved",
  object = "file.csv",
  reason = "Review complete"
)
verify_log(log)
#> regulog: Log intact: 1 entry, chain unbroken
```

`with_log` *Run an expression with automatic data read logging*

Description

Evaluates `expr` with a local `read()` binding tied to `log`, so calls inside the block don't need to repeat the `log` argument. Reads must use `read()` explicitly inside the block; calling a reader function directly (e.g. bare `haven::read_sas(...)`) is not logged. This keeps logging coverage unambiguous: every logged read is visible at the call site, and there are no implicit gaps.

Usage

```
with_log(log, expr)
```

Arguments

`log`	A `regulog` object.
`expr`	An expression, typically a `{}` block. Inside the block, `read(reader, ...)` is available and logs to `log` automatically.

Value

The value of `expr`, invisibly.

See Also

`rl_read()`

Examples

```
log <- regulog_init(app = "pipeline", version = "1.0", user = "jsmith")

## Not run:
with_log(log, {
  adsl <- read(haven::read_sas, "data/adsl.sas7bdat")
  adae <- read(haven::read_sas, "data/adae.sas7bdat")
})

filter_log(log, action = "data_read")

## End(Not run)
```

Index