Package: cs9 (via r-universe)

July 1, 2026

Title A Framework for Real-Time Analysis and Disease Surveillance

Version 26.5.13

Author Richard Aubrey White [aut, cre] (<https://orcid.org/0000-0002-6747-1726>), Core Surveillance [cph]

Maintainer Richard Aubrey White <hello@rwhite.no>

Description A comprehensive framework for building real-time disease surveillance systems. Provides R6-based infrastructure for database-driven surveillance tasks, with support for parallel processing, data validation, and automated pipeline execution. Designed for epidemiological analysis and public health monitoring applications.

URL https://niphr.github.io/cs9/, https://github.com/niphr/cs9

BugReports https://github.com/niphr/cs9/issues

Depends R (>= 4.1.0)

Imports callr, csdb, cstime (>= 2023.5.3), data.table, dplyr, fs, glue, later, lubridate, magrittr, plnr (>= 2021.5.4), processx, progress, progressr, R6, rstudioapi, stats, stringr

Suggests testthat, knitr, usethis, rmarkdown

License MIT + file LICENSE

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

VignetteBuilder knitr

Encoding UTF-8

Config/Needs/website niphr/cstemplate

Config/pak/sysreqs cmake git make libgit2-dev libicu-dev libuv1-dev unixodbc-dev libssl-dev libx11-dev

Repository https://niphr.r-universe.dev

Date/Publication 2026-07-01 09:02:08 UTC

RemoteUrl https://github.com/niphr/cs9

RemoteRef HEAD

RemoteSha 25d11d919bcbde4edf02e7180ad44f86b6aa1121

Contents

`check_environment_setup` *Check Environment Setup*

Description

Diagnostic function to check CS9 environment configuration. This function validates required environment variables and database connectivity, providing detailed feedback for troubleshooting configuration issues.

Usage

```
check_environment_setup(verbose = TRUE, use_startup_message = FALSE)
```

Arguments

`verbose`	Logical. If TRUE (default), prints detailed diagnostic output. If FALSE, runs validation silently and only returns result object.
`use_startup_message`	Logical. If TRUE, uses packageStartupMessage() for output (suppressible). If FALSE (default), uses cat() for console output.

Details

CS9 requires specific environment variables for database connectivity and configuration. This function checks for:

- Required environment variables (CS9_DBCONFIG_ACCESS, CS9_DBCONFIG_DRIVER, etc.)
- Database configuration availability
- Database table initialization status

When CS9 is installed from CRAN without database configuration, the package loads with limited functionality. Use this function to diagnose what needs to be configured for full database-driven surveillance functionality.

Value

A list containing environment setup status and recommendations:

- status: "ok", "warning", or "error"
- issues: Character vector of identified problems
- recommendations: Character vector of suggested fixes

See Also

The installation vignette: `vignette("installation", package = "cs9")`

Examples

```
# Check environment setup with verbose output
check_environment_setup()

# Check silently and examine results
result <- check_environment_setup(verbose = FALSE)
if(result$status != "ok") {
  cat("Issues found:", result$issues, "\n")
  cat("Recommendations:", result$recommendations, "\n")
}
```

`config` *Flags/values to be used in the 'dashboards' scene*

Description

Flags/values to be used in the 'dashboards' scene

Usage

```
config
```

Format

An object of class `environment` of length 7.

`create_folder_if_doesnt_exist` *Create Folder If It Doesn't Exist*

Description

Creates a directory and all necessary parent directories if they don't already exist.

Usage

```
create_folder_if_doesnt_exist(path)
```

Arguments

`path`	Character string specifying the directory path to create

Value

Character string containing the created directory path

Examples

```
## Not run:
# Create a new directory
create_folder_if_doesnt_exist("/tmp/my_analysis/results")

## End(Not run)
```

`create_latest_folder` *Create Latest Folder*

Description

Copies results from a dated folder to a "latest" folder, providing easy access to the most recent analysis results.

Usage

```
create_latest_folder(results_folder_name, date)
```

Arguments

results_folder_name	Character string specifying the name of the results folder (subdirectory under "output")
date	Character string specifying the date of extraction (used to identify the source folder)

Details

This function copies all contents from output/results_folder_name/date to output/results_folder_name/latest, overwriting existing files.

Value

No return value. This function is called for its side effect of copying files from the dated folder to the latest folder.

Examples

```
## Not run:
# Copy today's results to latest folder
create_latest_folder("covid_reports", "2024-01-15")

## End(Not run)
```

---

get_config_data_hash_for_each_plan

Get Configuration Data Hash for Each Plan

Description

Retrieves data hash configuration entries from the database table for tracking data changes across surveillance task plans and elements.

Usage

```
get_config_data_hash_for_each_plan(
  task = NULL,
  index_plan = NULL,
  element_tag = NULL
)
```

Arguments

task	Character string specifying the task name to filter by. If NULL, returns data for all tasks.
index_plan	Integer specifying the plan index to filter by. If NULL, returns data for all plans.
element_tag	Character string specifying the element tag to filter by. If NULL, returns data for all elements.

Value

A data.table containing the filtered hash configuration entries with columns: task, index_plan, element_tag, date, datetime, element_hash, all_hash

Examples

```
## Not run:
# Get all hash data for a specific task
get_config_data_hash_for_each_plan(task = "covid_analysis")

# Get hash data for specific task and plan
get_config_data_hash_for_each_plan(task = "covid_analysis", index_plan = 1)

## End(Not run)
```

---

get_config_log *Get Configuration Log*

Description

Retrieves configuration log entries from the config_log table with optional filtering by surveillance system identifier, task name, and date range.

Usage

```
get_config_log(ss = NULL, task = NULL, start_date = NULL, end_date = NULL)
```

Arguments

ss	Character. Surveillance system identifier to filter by. Defaults to NULL (no filtering).
task	Character. Task name to filter by. Defaults to NULL (no filtering).
start_date	Character. Start date (YYYY-MM-DD) for filtering log entries. Defaults to NULL.
end_date	Character. End date (YYYY-MM-DD) for filtering log entries. Defaults to NULL.

Details

The function retrieves entries from the config_log table in the current configuration. If date filters are provided, they are applied to the timestamp field of the log entries.

Value

A data.table containing the filtered log entries.

Examples

```
## Not run:
# Get all log entries
get_config_log()

# Get logs for a specific surveillance system
get_config_log(ss = "weather")

# Get logs for a specific task and date range
get_config_log(task = "data_import", start_date = "2024-01-01", end_date = "2024-12-31")

## End(Not run)
```

---

get_config_tables_last_updated

Get Configuration Tables Last Updated

Description

Retrieves the last updated timestamps for database tables from the configuration tracking system.

Usage

```
get_config_tables_last_updated(table_name = NULL)
```

Arguments

table_name	Character string specifying the table name to filter by. If NULL, returns data for all tables.

Value

A data.table containing last updated information with columns: table_name, last_updated_datetime, and other tracking metadata

Examples

```
## Not run:
# Get last updated info for all tables
get_config_tables_last_updated()

# Get info for a specific table
get_config_tables_last_updated(table_name = "anon_covid_cases")
```

```
## End(Not run)
```

`get_config_tasks_stats` *Get Configuration Tasks Statistics*

Description

Retrieves runtime statistics and performance metrics for surveillance tasks from the configuration database.

Usage

```
get_config_tasks_stats(task = NULL, last_run = FALSE)
```

Arguments

`task`	Character string specifying the task name to filter by. If NULL, returns statistics for all tasks.
`last_run`	Logical value indicating whether to return only the most recent run statistics. If FALSE, returns all historical data.

Value

A data.table containing task statistics with columns including: task, datetime, runtime_seconds, memory_usage, status, and other metrics

Examples

```
## Not run:
# Get all task statistics
get_config_tasks_stats()

# Get statistics for a specific task
get_config_tasks_stats(task = "covid_analysis")

# Get only the last run for a task
get_config_tasks_stats(task = "covid_analysis", last_run = TRUE)

## End(Not run)
```

mandatory_db_filter *Filter surveillance data by standard epidemiological dimensions*

Description

Applies a set of standard include/exclude filters to a database table or data frame using the conventional surveillance field names. Each NULL argument is a no-op, so callers only need to supply the dimensions they actually want to restrict.

Usage

```
mandatory_db_filter(
  .data,
  granularity_time = NULL,
  granularity_time_not = NULL,
  granularity_geo = NULL,
  granularity_geo_not = NULL,
  country_iso3 = NULL,
  location_code = NULL,
  age = NULL,
  age_not = NULL,
  sex = NULL,
  sex_not = NULL
)
```

Arguments

.data	A data frame, data.table, or remote database table (e.g. the result of dplyr::tbl()) that contains the standard surveillance columns.
granularity_time	Character vector. Values of granularity_time to keep. NULL (default) applies no filter.
granularity_time_not	Character vector. Values of granularity_time to drop. NULL (default) applies no filter.
granularity_geo	Character vector. Values of granularity_geo to keep. NULL (default) applies no filter.
granularity_geo_not	Character vector. Values of granularity_geo to drop. NULL (default) applies no filter.
country_iso3	Character vector. Values of country_iso3 to keep. NULL (default) applies no filter.
location_code	Character vector. Values of location_code to keep. NULL (default) applies no filter.
age	Character vector. Values of age to keep. NULL (default) applies no filter.

age_not	Character vector. Values of age to drop. NULL (default) applies no filter.
sex	Character vector. Values of sex to keep. NULL (default) applies no filter.
sex_not	Character vector. Values of sex to drop. NULL (default) applies no filter.

Value

The filtered object in the same class as .data.

Examples

```
## Not run:
# Inside a data_selector function, filter a remote table before collecting:
d <- schema$anon_covid_cases$tbl() |>
  mandatory_db_filter(
    granularity_time = "isoweek",
    granularity_geo  = "county",
    age_not          = "total"
  ) |>
  dplyr::collect()

## End(Not run)
```

---

path *Get Results Folder Path*

Description

Constructs the appropriate folder path for surveillance system results, with automatic switching between production and interactive modes.

Usage

```
path(
  ...,
  create_dir = FALSE,
  trailing_slash = FALSE,
  auto = cs9::config$is_auto
)
```

Arguments

...	Character strings specifying the second level directory and beyond
create_dir	Logical value indicating whether to create the directory if it doesn't exist. Defaults to FALSE.
trailing_slash	Logical value indicating whether to add a trailing slash to the returned path. Defaults to FALSE.

auto	Logical value indicating whether this is running in automatic mode (uses base directory) or interactive mode (adds "_interactive" subdirectory). Defaults to the current cs9::config$is_auto setting.

Value

Character string containing the constructed file path

Examples

```
## Not run:
# Get basic output path
path("reports", "daily")

# Create directory if it doesn't exist
path("reports", "daily", create_dir = TRUE)

# Get path with trailing slash
path("reports", "daily", trailing_slash = TRUE)

## End(Not run)
```

---

run_task_sequentially_as_callr_bg_using_load_all

Run a cs9 task in a background process, streaming its output live

---

Description

Positron-friendly counterpart to `run_task_sequentially_as_rstudio_job_using_load_all()`.

Usage

```
run_task_sequentially_as_callr_bg_using_load_all(
  task_name,
  ss_prefix = "global$ss"
)
```

Arguments

task_name	Character string. Name of the task to run.
ss_prefix	Character string. R expression that resolves to the surveillance system object in the child process. Defaults to `"global$ss"`.

Value

Invisibly returns the TaskJob R6 object (already started).

Examples

```
## Not run:
job <- run_task_sequentially_as_callr_bg_using_load_all("my_task")
job$wait()

## End(Not run)
```

`run_task_sequentially_as_rstudio_job_using_load_all`
Run a Task Sequentially as an RStudio Job

Description

Executes a surveillance task as an RStudio job using devtools::load_all() for package development. This function creates a temporary R script that loads the package and runs the specified task sequentially (cores = 1).

Usage

```
run_task_sequentially_as_rstudio_job_using_load_all(
  task_name,
  ss_prefix = "global$ss"
)
```

Arguments

`task_name`	Character string specifying the name of the task to run
`ss_prefix`	Character string specifying the prefix used to access the surveillance system object. Defaults to "global$ss"

Details

This function is primarily used during package development to test tasks interactively. It creates a temporary R script that:

- Loads the package using `devtools::load_all()`
- Sets the task to run with single core (cores = 1)
- Executes the task via the surveillance system

Value

No return value. This function is called for its side effect of launching an RStudio job that executes the surveillance task.

Examples

```
## Not run:
# Run a task as RStudio job during development
run_task_sequentially_as_rstudio_job_using_load_all("covid_analysis")

# Use custom surveillance system prefix
run_task_sequentially_as_rstudio_job_using_load_all(
  "covid_analysis",
  ss_prefix = "my_ss"
)

## End(Not run)
```

SurveillanceSystem_v9 *A Surveillance System Object*

Description

Core R6 class for creating and managing disease surveillance systems. This class orchestrates database tables, tasks, and analyses for real-time epidemiological monitoring and public health surveillance.

Details

SurveillanceSystem_v9 provides infrastructure for:

- Database table management with automated logging
- Task scheduling and parallel execution
- Data validation and schema enforcement
- Configuration and performance monitoring

The surveillance system follows a structured approach:

1. Define database tables with `add_table()`
2. Configure surveillance tasks with `add_task()`
3. Execute tasks with `run_task()` or external schedulers

Public fields

`tables` List of database tables managed by the surveillance system

`partitionedtables` List of partitioned database tables

`tasks` List of surveillance tasks configured for execution

`name` Character string identifying the surveillance system instance

`implementation_version` Character string tracking the analytics code version

Methods

Public methods:

- SurveillanceSystem_v9$new()
- SurveillanceSystem_v9$add_table()
- SurveillanceSystem_v9$add_partitionedtable()
- SurveillanceSystem_v9$add_task()
- SurveillanceSystem_v9$get_task()
- SurveillanceSystem_v9$run_task()
- SurveillanceSystem_v9$shortcut_get_tables()
- SurveillanceSystem_v9$shortcut_get_argset()
- SurveillanceSystem_v9$shortcut_get_data()
- SurveillanceSystem_v9$shortcut_get_plans_argsets_as_dt()
- SurveillanceSystem_v9$shortcut_get_num_analyses()
- SurveillanceSystem_v9$clone()

Method new():

Usage:

```
SurveillanceSystem_v9$new(
  name = "unspecified",
  implementation_version = "unspecified"
)
```

Arguments:

name A string that the user may choose to use to track performance metrics (runtime and RAM usage)

implementation_version A string that the user may choose to use to track performance metrics (runtime and RAM usage)

Method add_table(): Add a table

Usage:

```
SurveillanceSystem_v9$add_table(
  name_access,
  name_grouping = NULL,
  name_variant = NULL,
  field_types,
  keys,
  indexes = NULL,
  validator_field_types = csdb::validator_field_types_blank,
  validator_field_contents = csdb::validator_field_contents_blank
)
```

Arguments:

name_access First part of table name, corresponding to the database where it will be stored.

name_grouping Second part of table name, corresponding to some sort of grouping.

name_variant Final part of table name, corresponding to a distinguishing variant.

`field_types` Named character vector, where the names are the column names, and the values are the column types. Valid types are BOOLEAN, CHARACTER, INTEGER, DOUBLE, DATE, DATETIME

`keys` Character vector, containing the column names that uniquely identify a row of data.

`indexes` Named list, containing indexes.

`validator_field_types` Function corresponding to a validator for the field types.

`validator_field_contents` Function corresponding to a validator for the field contents.

Returns: No return value. This method is called for its side effect of adding a table to the surveillance system.

Examples:

```
\dontrun{
global$ss$add_table(
  name_access = c("anon"),
  name_grouping = "example_weather",
  name_variant = "data",
  field_types =  c(
    "granularity_time" = "TEXT",
    "granularity_geo" = "TEXT",
    "country_iso3" = "TEXT",
    "location_code" = "TEXT",
    "border" = "INTEGER",
    "age" = "TEXT",
    "sex" = "TEXT",

    "isoyear" = "INTEGER",
    "isoweek" = "INTEGER",
    "isoyearweek" = "TEXT",
    "season" = "TEXT",
    "seasonweek" = "DOUBLE",

    "calyear" = "INTEGER",
    "calmonth" = "INTEGER",
    "calyearmonth" = "TEXT",

    "date" = "DATE",

    "temp_max" = "DOUBLE",
    "temp_min" = "DOUBLE",
    "precip" = "DOUBLE"
  ),
  keys = c(
    "granularity_time",
    "location_code",
    "date",
    "age",
    "sex"
  ),
```

```
  validator_field_types = csdb::validator_field_types_csfmt_rts_data_v1,
  validator_field_contents = csdb::validator_field_contents_csfmt_rts_data_v1
)
}
```

Method `add_partitionedtable()`**:** Add a partitioned table to the surveillance system

Usage:

```
SurveillanceSystem_v9$add_partitionedtable(
  name_access,
  name_grouping = NULL,
  name_variant = NULL,
  name_partitions = "default",
  column_name_partition = "partition",
  value_generator_partition = NULL,
  field_types,
  keys,
  indexes = NULL,
  validator_field_types = csdb::validator_field_types_blank,
  validator_field_contents = csdb::validator_field_contents_blank
)
```

Arguments:

`name_access` First part of table name, corresponding to the database where it will be stored

`name_grouping` Second part of table name, corresponding to some sort of grouping

`name_variant` Final part of table name, corresponding to a distinguishing variant

`name_partitions` Character string specifying partition naming scheme

`column_name_partition` Column name used for partitioning

`value_generator_partition` Function to generate partition values

`field_types` Named character vector of column names and types

`keys` Character vector of column names that uniquely identify rows

`indexes` Named list containing index definitions

`validator_field_types` Function to validate field types

`validator_field_contents` Function to validate field contents

Returns: No return value. This method is called for its side effect of adding a partitioned table to the surveillance system.

Method `add_task()`**:** Add a surveillance task to the system

Usage:

```
SurveillanceSystem_v9$add_task(
  name_grouping = NULL,
  name_action = NULL,
  name_variant = NULL,
  cores = 1,
  permission = NULL,
  plan_analysis_fn_name = NULL,
  for_each_plan = NULL,
```

```
  for_each_analysis = NULL,
  universal_argset = NULL,
  upsert_at_end_of_each_plan = FALSE,
  insert_at_end_of_each_plan = FALSE,
  action_fn_name,
  data_selector_fn_name = NULL,
  tables = NULL
)
```

Arguments:

`name_grouping` Name of the task (grouping)

`name_action` Name of the task (action)

`name_variant` Name of the task (variant)

`cores` Number of CPU cores

`permission` A permission R6 instance

`plan_analysis_fn_name` The name of a function that returns a named list `list(for_each_plan = list(), for_each_analysis = NULL)`.

`for_each_plan` A list, where each unit corresponds to one data extraction. Generally recommended to use `plnr::expand_list`.

`for_each_analysis` A list, where each unit corresponds to one analysis within a plan (data extraction). Generally recommended to use `plnr::expand_list`.

`universal_argset` A list, where these argsets are applied to all analyses univerally

`upsert_at_end_of_each_plan` Do you want to upsert your results automatically at the end of each plan?

`insert_at_end_of_each_plan` Do you want to insert your results automatically at the end of each plan?

`action_fn_name` The name of the function that will be called for each analysis with arguments `data`, `argset`, `schema`

`data_selector_fn_name` The name of a function that will be called to obtain the data for each analysis. The function must have the arguments `argset`, `schema` and must return a named list.

`tables` A named list that maps `cs9::config$schemas` for use in `action_fn_name` and `data_selector_fn_name`

Returns: No return value. This method is called for its side effect of adding a task to the surveillance system.

Method `get_task()`**:** Get a surveillance task by name

Usage:

```
SurveillanceSystem_v9$get_task(task_name)
```

Arguments:

`task_name` Character string specifying the task name

Returns: A Task R6 object representing the surveillance task

Method `run_task()`**:** Execute a surveillance task by name

Usage:

```
SurveillanceSystem_v9$run_task(task_name)
```

Arguments:

`task_name` Character string specifying the task name to run

Returns: No return value. This method is called for its side effect of executing the task.

Method `shortcut_get_tables()`**:** Get database tables associated with a task

Usage:

```
SurveillanceSystem_v9$shortcut_get_tables(task_name)
```

Arguments:

`task_name` Character string specifying the task name

Returns: A named list of database table objects used by the task

Method `shortcut_get_argset()`**:** Get argument set for a specific plan and analysis

Usage:

```
SurveillanceSystem_v9$shortcut_get_argset(
  task_name,
  index_plan = 1,
  index_analysis = 1
)
```

Arguments:

`task_name` Character string specifying the task name

`index_plan` Integer specifying which plan to access

`index_analysis` Integer specifying which analysis to access

Returns: A named list containing the argument set for the specified plan and analysis

Method `shortcut_get_data()`**:** Get data for a specific plan

Usage:

```
SurveillanceSystem_v9$shortcut_get_data(task_name, index_plan = 1)
```

Arguments:

`task_name` Character string specifying the task name

`index_plan` Integer specifying which plan to access

Returns: A named list containing the data extracted for the specified plan

Method `shortcut_get_plans_argsets_as_dt()`**:** Get plans and argsets as a data.table

Usage:

```
SurveillanceSystem_v9$shortcut_get_plans_argsets_as_dt(task_name)
```

Arguments:

`task_name` Character string specifying the task name

Returns: A data.table containing plan and analysis information with columns including index_plan and index_analysis

Method `shortcut_get_num_analyses()`**:** Get the total number of analyses for a task

Usage:

```
SurveillanceSystem_v9$shortcut_get_num_analyses(task_name)
```

Arguments:

`task_name` Character string specifying the task name

Returns: Integer value representing the total number of analyses across all plans for the task

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

Usage:

```
SurveillanceSystem_v9$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
## Not run:
# Create surveillance system
ss <- SurveillanceSystem_v9$new(
  name = "covid_surveillance",
  implementation_version = "1.0"
)

# Add database table
ss$add_table(
  name_access = "anon",
  name_grouping = "covid",
  name_variant = "cases",
  field_types = c("date" = "DATE", "cases" = "INTEGER"),
  keys = c("date")
)

# Add surveillance task
ss$add_task(
  name_grouping = "covid",
  name_action = "import",
  name_variant = "daily_data",
  action_fn_name = "import_covid_data",
  data_selector_fn_name = "select_covid_sources"
)

# Run task
ss$run_task("covid_import_daily_data")

## End(Not run)


## ------------------------------------------------
## Method `SurveillanceSystem_v9$add_table`
## ------------------------------------------------

## Not run:
global$ss$add_table(
```

```
  name_access = c("anon"),
  name_grouping = "example_weather",
  name_variant = "data",
  field_types =  c(
    "granularity_time" = "TEXT",
    "granularity_geo" = "TEXT",
    "country_iso3" = "TEXT",
    "location_code" = "TEXT",
    "border" = "INTEGER",
    "age" = "TEXT",
    "sex" = "TEXT",

    "isoyear" = "INTEGER",
    "isoweek" = "INTEGER",
    "isoyearweek" = "TEXT",
    "season" = "TEXT",
    "seasonweek" = "DOUBLE",

    "calyear" = "INTEGER",
    "calmonth" = "INTEGER",
    "calyearmonth" = "TEXT",

    "date" = "DATE",

    "temp_max" = "DOUBLE",
    "temp_min" = "DOUBLE",
    "precip" = "DOUBLE"
  ),
  keys = c(
    "granularity_time",
    "location_code",
    "date",
    "age",
    "sex"
  ),
  validator_field_types = csdb::validator_field_types_csfmt_rts_data_v1,
  validator_field_contents = csdb::validator_field_contents_csfmt_rts_data_v1
)

## End(Not run)
```

TaskJob *TaskJob R6 Class*

Description

Run a cs9 task in a background R process with live output streaming back to the current R console via `later::later()` polling. Equivalent in spirit to `run_task_sequentially_as_rstudio_job_using_load_all()` but does not depend on RStudio's job API (which Positron does not implement).

Methods: $start() spawn background process and begin polling $wait(t) block this session, draining the event loop, until done $is_alive() TRUE while the background process is running $status() summary $tail(n) last n lines of the log file (snapshot, not live) $kill() terminate the background process

Public fields

`task_name` Character string. Name of the task being run.

`ss_prefix` Character string. R expression used to access the surveillance system in the child process.

`script_path` Character string. Path to the temporary R script that is sourced in the background process.

`log_path` Character string. Path to the log file where background process output is written.

`started_at` POSIXct. Time at which `$start()` was called.

`finished_at` POSIXct. Time at which the background process exited.

Methods

Public methods:

- `TaskJob$new()`
- `TaskJob$start()`
- `TaskJob$is_alive()`
- `TaskJob$status()`
- `TaskJob$wait()`
- `TaskJob$kill()`
- `TaskJob$tail()`
- `TaskJob$clone()`

`TaskJob$new()`: Create a new TaskJob.

Usage:

```
TaskJob$new(
  task_name,
  ss_prefix = "global$ss",
  log_dir = tempdir(check = TRUE)
)
```

Arguments:

`task_name` Character string. Name of the task to run, as registered in the surveillance system (e.g. `ss$tasks[["my_task"]]`).

`ss_prefix` Character string. R expression that evaluates to the surveillance system object in the child process. Defaults to `"global$ss"`.

`log_dir` Character string. Directory where the temporary script and log file are written. Defaults to `tempdir()`.

Returns: A new `TaskJob` object. Call `$start()` to launch the background process.

`TaskJob$start()`: Spawn the background process and begin polling its output.

Usage:

```
TaskJob$start()
```

Returns: Invisibly returns the TaskJob object.

`TaskJob$is_alive()`: Check whether the background process is still running.

Usage:

```
TaskJob$is_alive()
```

Returns: Logical. TRUE while the background process is alive.

`TaskJob$status()`: Print a brief status summary (task name, start time, alive status, log path).

Usage:

```
TaskJob$status()
```

Returns: Invisibly returns the TaskJob object.

`TaskJob$wait()`: Block the current session until the background task finishes or the timeout expires, draining the **later** event loop while waiting.

Usage:

```
TaskJob$wait(timeout_s = 600)
```

Arguments:

timeout_s Numeric. Maximum number of seconds to wait. Defaults to 600 (10 minutes).

Returns: Invisibly returns the TaskJob object.

`TaskJob$kill()`: Terminate the background process.

Usage:

```
TaskJob$kill()
```

Returns: Invisibly returns the TaskJob object.

`TaskJob$tail()`: Print the last n lines of the task log file (snapshot, not live).

Usage:

```
TaskJob$tail(n = 20)
```

Arguments:

n Integer. Number of trailing lines to show. Defaults to 20.

Returns: Invisibly returns the TaskJob object.

`TaskJob$clone()`: The objects of this class are cloneable with this method.

Usage:

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

Arguments:

deep Whether to make a deep clone.

Examples

```
## Not run:
job <- TaskJob$new("my_task")
job$start()
job$status()
job$wait(timeout_s = 120)
job$tail(n = 30)

## End(Not run)
```

`update_config_log` *Update Configuration Log*

Description

Logs configuration updates with relevant metadata such as timestamp, session state, task name, and a custom message. The function inserts the log entry into the `config_log` table in the current configuration.

Usage

```
update_config_log(ss = "unspecified", task = "unspecified", ...)
```

Arguments

`ss`	Character. Surveillance system identifier. Defaults to `"unspecified"`.
`task`	Character. Name of the task being logged. Defaults to `"unspecified"`.
`...`	Character. Custom message describing the log entry. Must not be `NULL`.

Details

The function records the type of interaction (automatic or interactive), session state, task description, and a user-provided message in the configuration log. It throws an error if the `message` argument is `NULL`.

Value

No return value; this function is called for its side effect of inserting a log entry into the `config_log` table.

Examples

```
## Not run:
update_config_log(ss = "weather", task = "data_import", message = "Imported dataset successfully.")

## End(Not run)
```

Index