Package: framework (via r-universe)

October 31, 2026

Type Package

Title Structured Data Science Project Scaffolding

Version 1.1.1

Maintainer Erik Westlund <erik@framework.pub>

Description Project scaffolding and workflow tools for reproducible data science. Manages packages, tracks data integrity, handles database connections, generates notebooks, and publishes to S3-compatible storage. More information at <https://framework.table1.org>.

License MIT + file LICENSE

URL https://framework.table1.org, https://github.com/frameworktools/framework

BugReports https://github.com/frameworktools/framework/issues

Encoding UTF-8

Imports checkmate, DBI, RSQLite, yaml, fs, readr, dotenv, openssl, lubridate, jsonlite

Suggests testthat (>= 3.0.0), arrow, aws.s3, aws.signature, BiocManager, cli, cyclocomp, devtools, dplyr, DT, duckdb, ggplot2, haven, htmltools, htmlwidgets, knitr, languageserver, odbc, plumber, pool, prismjs, R.utils, readxl, remotes, renv, RMariaDB, rmarkdown, RPostgres, tm, usethis, withr, writexl

Config/testthat/edition 3

Depends R (>= 4.1.0)

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Erik Westlund [aut, cre, cph]

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

Date/Publication 2026-10-31 12:49:01 UTC

RemoteUrl https://github.com/cran/framework

RemoteRef HEAD

RemoteSha 61a772db4b8dd148f6e1bf1917bcf9513a3deb44

Contents

add_project_to_config . . . 4
ai_generate_context . . . 4
ai_regenerate_context . . . 5
ai_skills . . . 6
ai_skills_update . . . 6
ai_sync_context . . . 7
cache . . . 8
cache_flush . . . 9
cache_forget . . . 9
cache_get . . . 10
cache_list . . . 10
cache_remember . . . 11
capture_output . . . 12
configure_global . . . 12
connections_list . . . 14
connections_s3 . . . 14
data_add . . . 14
data_info . . . 15
data_list . . . 16
data_read . . . 17
data_save . . . 18
db_connect . . . 19
db_drivers_install . . . 19
db_drivers_status . . . 20
db_execute . . . 21
db_list . . . 22
db_query . . . 22
db_transaction . . . 23
db_with . . . 24
docs_export . . . 25
env_clear . . . 27
env_summary . . . 27
fw_config_dir . . . 28
git_add . . . 28
git_commit . . . 29
git_diff . . . 30
git_hooks_disable . . . 30
git_hooks_enable . . . 31
git_hooks_install . . . 31
git_hooks_list . . . 32
git_hooks_uninstall . . . 33
git_log . . . 33
git_pull . . . 34
git_push . . . 34
git_security_audit . . . 35
git_status . . . 37

gui . . . 37
load_settings_catalog . . . 38
make_notebook . . . 39
make_presentation . . . 41
make_qmd . . . 42
make_revealjs . . . 43
make_rmd . . . 44
make_script . . . 45
new . . . 46
new_course . . . 47
new_presentation . . . 48
new_project . . . 49
new_project_sensitive . . . 50
now . . . 51
outputs . . . 51
packages_install . . . 52
packages_list . . . 52
packages_restore . . . 53
packages_snapshot . . . 53
packages_status . . . 54
packages_update . . . 54
project_info . . . 55
project_list . . . 55
publish . . . 56
publish_data . . . 57
publish_dir . . . 58
publish_list . . . 59
publish_notebook . . . 59
quarto_generate_all . . . 61
quarto_regenerate . . . 61
read_framework_template . . . 62
read_frameworkrc . . . 62
remove_project_from_config . . . 63
renv_disable . . . 63
renv_enable . . . 64
renv_enabled . . . 64
reset_framework_template . . . 65
result_list . . . 65
save_figure . . . 66
save_model . . . 67
save_notebook . . . 68
save_report . . . 69
save_table . . . 70
scaffold . . . 71
scratch_capture . . . 73
scratch_clean . . . 74
settings . . . 74
settings_read . . . 75

settings_write 76
setup 76
standardize_wd 77
status 78
storage_test 79
stubs_list 79
stubs_path 80
stubs_publish 81
view 82
write_framework_template 83
write_frameworkrc 84

Index **85**

`add_project_to_config` *Add project to global configuration*

Description

Add project to global configuration

Usage

```
add_project_to_config(project_dir, project_name = NULL, project_type = NULL)
```

Arguments

`project_dir`	Path to project directory
`project_name`	Optional project name
`project_type`	Optional project type

Value

Invisibly returns the project ID

`ai_generate_context` *Generate AI Context File*

Description

Generates the canonical AI context file (AGENTS.md) for a project. The file is intentionally thin: it carries project-specific facts (environment, packages, directories) plus an index of the Framework skills installed in `.claude/skills/`, which hold the detailed workflow instructions.

Usage

```
ai_generate_context(
  project_path = ".",
  project_name = NULL,
  project_type = NULL,
  config = NULL
)
```

Arguments

project_path	Path to the project directory (default: current directory)
project_name	Name of the project (for header)
project_type	Project type: "project", "project_sensitive", "course", "presentation"
config	Project configuration (if NULL, reads from settings.yml)

Value

Character string with the complete AI context content

Examples

```
if (FALSE) {
# Generate AI context for current project
content <- ai_generate_context()

# Generate for a specific project type
content <- ai_generate_context(project_type = "project_sensitive")
}
```

---

ai_regenerate_context *Regenerate Dynamic Sections in AI Context File*

Description

Updates only the sections marked with <!-- @framework:regenerate --> in an existing AI context file, preserving user customizations in unmarked sections.

Usage

```
ai_regenerate_context(project_path = ".", sections = NULL, ai_file = NULL)
```

Arguments

project_path	Path to the project directory
sections	Which sections to regenerate. NULL = all regeneratable sections. Options: "environment", "packages", "data", "functions"
ai_file	Name of the AI context file (default: from settings or "CLAUDE.md")

Value

Invisible TRUE on success

Examples

```
if (file.exists("CLAUDE.md")) {
  # Regenerate all dynamic sections
  ai_regenerate_context()

  # Regenerate only packages section
  ai_regenerate_context(sections = "packages")
}
```

---

ai_skills *AI Skills Management*

Description

Framework ships agent "skills" - focused instruction files (SKILL.md) that AI coding assistants load on demand. Skills are installed into a project's `.claude/skills/` directory and indexed from AGENTS.md, so both Claude Code (which discovers them natively) and other agents (which follow the paths listed in AGENTS.md) can use them.

---

ai_skills_update *Update Framework Skills in a Project*

Description

Re-copies the Framework skill files (`.claude/skills/framework-*/SKILL.md`) from the installed package version into the project. Run this after upgrading the framework package to pick up improved skill instructions.

Usage

```
ai_skills_update(project_path = ".")
```

Arguments

`project_path`	Path to the project directory (default: current directory)

Details

Only Framework-owned skills (prefixed `framework-`) are touched; any custom skills you add alongside them are left alone.

Value

Invisible character vector of updated skill names

Examples

```
if (FALSE) {
# Refresh skills after upgrading the framework package
ai_skills_update()
}
```

`ai_sync_context` *Sync AI Assistant Context Files*

Description

Copies content from the canonical AI assistant file to all other AI files, adding a warning header to non-canonical files.

Usage

```
ai_sync_context(config_file = NULL, force = FALSE, verbose = TRUE)
```

Arguments

`config_file`	Path to configuration file (default: auto-detect settings.yml/settings.yml)
`force`	Logical; if TRUE, overwrite even if target is newer (default: FALSE)
`verbose`	Logical; if TRUE (default), show sync messages

Details

This function reads the `ai.canonical_file` setting from settings.yml and copies its content to all other AI assistant instruction files that exist in the project.

The canonical file is copied as-is. Non-canonical files receive a warning header indicating they are auto-synced and should not be edited directly.

Supported files:

- **AGENTS.md** - Cross-platform AI assistants
- **CLAUDE.md** - Claude Code
- **.github/copilot-instructions.md** - GitHub Copilot

Value

Invisible list with sync results

Examples

```
if (FALSE) {
# Sync AI context files
ai_sync_context()

# Force sync even if targets are newer
ai_sync_context(force = TRUE)

# Silent sync (for git hooks)
ai_sync_context(verbose = FALSE)
}
```

---

cache *Cache a value*

Description

Cache a value

Usage

```
cache(name, value, file = NULL, expire_after = NULL)
```

Arguments

name	The cache name
value	The value to cache
file	Optional file path to store the cache (default: cache/{name}.rds)
expire_after	Optional expiration time in hours (default: from config)

Value

The cached value

cache_flush *Clear all cached values*

Description

Clear all cached values

Usage

```
cache_flush()
```

Value

Invisibly returns NULL.

cache_forget *Remove a cached value*

Description

Remove a cached value

Usage

```
cache_forget(name, file = NULL)
```

Arguments

name	The cache name to remove
file	Optional file path of the cache (default: cache/{name}.rds)

Value

Invisibly returns NULL.

cache_get *Get a cached value*

Description

Get a cached value

Usage

```
cache_get(name, file = NULL, expire_after = NULL)
```

Arguments

name	The cache name
file	Optional file path to store the cache (default: cache/{name}.rds)
expire_after	Optional expiration time in hours (default: from config)

Value

The cached value, or NULL if not found, expired, or hash mismatch

cache_list *List all cached values*

Description

Returns a data frame of all cache entries with their names, expiration times, and status (expired or active).

Usage

```
cache_list()
```

Value

A data frame with columns:

name Cache key name

expire_at Expiration timestamp (NA if no expiration)

created_at When the cache was created

updated_at When the cache was last updated

last_read_at When the cache was last read

status Either "active" or "expired"

Returns an empty data frame if no cache entries exist.

Examples

```
if (FALSE) {
# List all cache entries
cache_list()

# Filter to see only expired caches
cache_list() |> dplyr::filter(status == "expired")
}
```

`cache_remember` *Remember a value (get from cache or compute and store)*

Description

Attempts to retrieve a cached value by name. If the cache doesn't exist, is expired, or a refresh is requested, evaluates the expression and caches the result. This is the primary caching interface for expensive computations.

Usage

```
cache_remember(
  name,
  expr,
  file = NULL,
  expire_after = NULL,
  refresh = FALSE,
  expire = NULL
)
```

Arguments

`name`	The cache name (non-empty string identifier)
`expr`	The expression to evaluate and cache if cache miss occurs. Expression is evaluated in the parent frame.
`file`	Optional file path to store the cache (default: `cache/{name}.rds`)
`expire_after`	Optional expiration time in hours (default: from config). Character durations like "1 day" or "2 hours" are accepted.
`refresh`	Optional boolean or function that returns boolean to force refresh. If TRUE or if function returns TRUE, cache is invalidated and expression is re-evaluated.
`expire`	Optional alias for `expire_after` (accepts the same formats)

Value

The cached value (if cache hit) or the result of evaluating expr (if cache miss or refresh requested)

Examples

```
if (FALSE) {
# Cache expensive computation
result <- cache_remember("my_analysis", {
  expensive_computation()
})

# Force refresh when data changes
result <- cache_remember("analysis", {
  run_analysis()
}, refresh = file.mtime("data.csv") > cache_time)
}
```

`capture_output` *Capture console output and errors from an expression*

Description

Capture console output and errors from an expression

Usage

```
capture_output(expr)
```

Arguments

`expr`	Expression to evaluate

Value

List containing status (boolean), console_output (character vector), and result (return value or error)

`configure_global` *Configure Global Framework Settings*

Description

Unified function for reading and writing global Framework settings to ~/.frameworkrc.json. This function provides a single source of truth for global configuration, used by both the CLI and GUI interfaces.

Usage

```
configure_global(settings = NULL, validate = TRUE)
```

Arguments

settings	List. Settings to update (partial updates supported)
validate	Logical. Validate settings before saving (default: TRUE)

Details

Global Settings Structure:

- author - Author information (name, email, affiliation)
- defaults - Project defaults
  - project_type - Default project type ("project", "presentation", "course")
  - notebook_format - Default notebook format ("quarto", "rmarkdown")
  - ide - IDE preference ("vscode", "rstudio", "both", "none")
  - use_git - Initialize git repositories by default
  - use_renv - Enable renv by default
  - seed - Default random seed
  - seed_on_scaffold - Set seed during scaffold()
  - ai_support - Enable AI assistant support
  - ai_assistants - List of AI assistants ("claude", "agents", etc.)
  - ai_canonical_file - Canonical AI instruction file
  - packages - Default package list
  - directories - Default directory structure
  - git_hooks - Git hook preferences
- projects - Registered projects list
- active_project - Currently active project path

Value

Invisibly returns updated global configuration

Examples

```
if (FALSE) {
# Update author information
configure_global(settings = list(
  author = list(
    name = "Jane Doe",
    email = "jane@example.com"
  )
))

# Update default project type
configure_global(settings = list(
  defaults = list(
    project_type = "presentation"
  )
))
```

```
# Get current settings (read-only)
current <- configure_global()
}
```

`connections_list` *List all connections (databases and object storage)*

Description

Prints both database connections defined under `connections:` and object storage profiles (S3-compatible buckets). Use this to see everything Framework can talk to from your config.

Usage

```
connections_list()
```

Value

Invisibly returns NULL after printing summaries.

`connections_s3` *S3 Connection Functions*

Description

Functions for connecting to and interacting with S3-compatible storage.

`data_add` *Add an existing file to the data catalog*

Description

Registers an existing data file with the Framework data catalog. This allows you to track files that were created outside of Framework (e.g., downloaded from external sources, copied from other projects) and use them with `data_read()` using dot notation.

Usage

```
data_add(
  file_path,
  name = NULL,
  type = NULL,
  delimiter = "comma",
  locked = TRUE,
  update_config = TRUE
)
```

Arguments

file_path	Path to the existing file (must exist)
name	Optional dot notation name for the data catalog (e.g., inputs.raw.survey_data). If NULL, derives name from file path relative to project root.
type	Optional type override. Auto-detected from file extension if NULL.
delimiter	Delimiter for CSV files ("comma", "tab", "semicolon", "space")
locked	Whether the file should be locked (hash-verified on read)
update_config	If TRUE (default), also updates the YAML config with the data spec

Value

Invisibly returns the data spec that was created

Examples

```
if (FALSE) {
# Add a downloaded CSV file to the catalog
data_add("inputs/raw/survey_results.csv", name = "inputs.raw.survey_results")

# Now you can read it with dot notation
data_read("inputs.raw.survey_results")

# Add with auto-generated name
data_add("inputs/intermediate/cleaned_data.rds")
# Name will be derived as "inputs.intermediate.cleaned_data"
}
```

---

data_info *Get data specification from config*

Description

Gets the data specification for a given dot notation path from settings.yml. Supports dot notation (e.g., "source.private.example"), relative paths, and absolute paths. Auto-detects file type from extension and applies intelligent defaults for common formats.

Usage

```
data_info(path)
```

Arguments

path	Dot notation path (e.g. "source.private.example"), relative path, or absolute path to a data file

Value

A list with data specification including:

- path - Full file path
- type - File type (csv, rds, stata, spss, sas, etc.)
- delimiter - Delimiter for CSV files (comma, tab, etc.)
- locked - Whether file is locked for integrity checking
- private - Whether file is in private data directory
- description - Optional description of the dataset (displayed when loading)

Examples

```
if (FALSE) {
# Get info from dot notation
info <- data_info("source.private.my_data")

# Get info from file path
info <- data_info("data/public/example.csv")
}
```

---

data_list *List all data entries from config*

Description

Lists all data specifications defined in the configuration, showing the data key, path, type, and description (if available).

Usage

```
data_list()
```

Value

A data frame with columns: name, path, type, locked, description

Examples

```
if (FALSE) {
# List all data entries
data_list()

# Use the alias
list_data()
}
```

`data_read` *Read data using dot notation path or direct file path*

Description

Supports CSV, TSV, RDS, Excel (.xlsx, .xls), Stata (.dta), SPSS (.sav, .zsav, .por), and SAS (.sas7bdat, .xpt) file formats.

Usage

```
data_read(path, delim = NULL, keep_attributes = FALSE, ...)
```

Arguments

`path`	Dot notation path (e.g. "source.private.example") or direct file path
`delim`	Optional delimiter for CSV files ("comma", "tab", "semicolon", "space")
`keep_attributes`	Logical flag to preserve special attributes (e.g., haven labels). Default: FALSE (strips attributes)
`...`	Additional arguments passed to read functions (readr::read_delim, readxl::read_excel, haven::read_*, etc.)

Value

The loaded data, typically a data frame or tibble.

`data_save` *Save data using dot notation or file path*

Description

Save data using dot notation or file path

Usage

```
data_save(
  data,
  path,
  type = NULL,
  delimiter = "comma",
  locked = TRUE,
  force = FALSE
)
```

Arguments

`data`	Data frame to save
`path`	Either: • Dot notation: `inputs.raw.filename` resolves to inputs/raw/filename.rds • Direct path: "inputs/raw/filename.csv" uses path as-is Dot notation uses your configured directories (e.g., `inputs.raw`, `inputs.intermediate`, `outputs.private`).
`type`	Type of data file ("csv" or "rds"). Auto-detected from extension if path includes one.
`delimiter`	Delimiter for CSV files ("comma", "tab", "semicolon", "space")
`locked`	Whether the file should be locked after saving
`force`	If TRUE, creates missing directories. If FALSE (default), errors if directory doesn't exist.

Value

Invisibly returns the saved data.

db_connect *Get a database connection*

Description

Gets a database connection based on the connection name in config.yml. For most use cases, prefer `db_query()` or `db_execute()` which handle connection lifecycle automatically.

Usage

```
db_connect(name)
```

Arguments

`name`	Character. Name of the connection in config.yml (e.g., "postgres")

Value

A database connection object (DBIConnection)

Examples

```
if (FALSE) {
# Preferred: use db_query() which auto-disconnects
users <- db_query("SELECT * FROM users", "postgres")

# Manual connection management (remember to disconnect!)
conn <- db_connect("postgres")
DBI::dbListTables(conn)
DBI::dbDisconnect(conn)
}
```

db_drivers_install *Install database drivers*

Description

Interactive helper to install one or more database drivers. Provides helpful instructions and handles special cases (like ODBC).

Usage

```
db_drivers_install(drivers = NULL, repos = getOption("repos"))
```

Arguments

drivers	Character vector. Database driver names to install (e.g., "postgres", "mysql", "duckdb"). If NULL, shows interactive menu.
repos	Character. CRAN repository URL. Default: getOption("repos")

Value

NULL (invisible). Installs packages as side effect.

Examples

```
if (FALSE) {
# Install specific drivers
db_drivers_install(c("postgres", "mysql"))

# Interactive mode
db_drivers_install()
}
```

---

db_drivers_status *Check if database drivers are installed*

Description

Checks which database drivers are currently available on the system. Returns a data frame showing the status of all supported database drivers.

Usage

```
db_drivers_status(quiet = FALSE)
```

Arguments

quiet	Logical. If TRUE, suppresses messages. Default: FALSE

Value

A data frame with columns:

- driver: Database driver name
- package: Required R package
- installed: Whether the package is installed
- version: Package version (if installed)

Examples

```
# Check all drivers
db_drivers_status()

# Quiet mode (no messages)
db_drivers_status(quiet = TRUE)
```

`db_execute` *Execute a database statement*

Description

Executes a SQL statement on a database without returning results. The connection is created, used, and automatically closed.

Usage

```
db_execute(query, connection_name, ...)
```

Arguments

`query`	SQL statement to execute
`connection_name`	Name of the connection in config.yml
`...`	Additional arguments passed to DBI::dbExecute

Value

Number of rows affected

Examples

```
if (FALSE) {
rows <- db_execute("DELETE FROM cache WHERE expired = TRUE", "my_db")
}
```

db_list *List all database connections from configuration*

Description

Lists all database connections defined in the configuration, showing the connection name, driver, host, and database name (if applicable).

Usage

```
db_list()
```

Value

Invisibly returns NULL after printing connection list

Examples

```
if (FALSE) {
# List all connections
db_list()
}
```

db_query *Get data from a database query*

Description

Gets data from a database using a query and connection name. The connection is created, used, and automatically closed.

Usage

```
db_query(query, connection_name, ...)
```

Arguments

query	SQL query to execute
connection_name	Name of the connection in config.yml
...	Additional arguments passed to DBI::dbGetQuery

Value

A data frame with the query results

Examples

```
if (FALSE) {
users <- db_query("SELECT * FROM users", "my_db")
}
```

`db_transaction` *Execute code within a database transaction*

Description

Wraps code execution in a database transaction with automatic commit on success and rollback on error. This ensures atomicity of multiple database operations.

Usage

```
db_transaction(conn, code)
```

Arguments

`conn`	Database connection
`code`	Expression or code block to execute within the transaction

Details

The function automatically:

- Begins a transaction with `DBI::dbBegin()`
- Executes the provided code
- Commits the transaction on success with `DBI::dbCommit()`
- Rolls back the transaction on error with `DBI::dbRollback()`

Transactions are essential for maintaining data integrity when performing multiple related operations. If any operation fails, all changes are rolled back.

Value

The result of the code expression

Examples

```
if (FALSE) {
conn <- db_connect("postgres")

# Basic transaction
db_transaction(conn, {
  DBI::dbExecute(conn, "INSERT INTO users (name, age) VALUES ('Alice', 30)")
  DBI::dbExecute(conn, "INSERT INTO users (name, age) VALUES ('Bob', 25)")
})

# Transaction with error handling - auto-rollback on error
tryCatch({
  db_transaction(conn, {
    DBI::dbExecute(conn, "INSERT INTO users (name) VALUES ('Alice')")
    stop("Something went wrong")  # This will trigger rollback
  })
}, error = function(e) {
  message("Transaction failed: ", e$message)
})

DBI::dbDisconnect(conn)
}
```

---

db_with *Execute code with a managed database connection*

Description

Provides automatic connection lifecycle management. The connection is automatically closed when the code block finishes, even if an error occurs. This prevents connection leaks and ensures proper resource cleanup.

Usage

```
db_with(connection_name, code)
```

Arguments

connection_name	Character. Name of the connection in config.yml
code	Expression to evaluate with the connection (use conn to access the connection)

Value

The result of evaluating code

Examples

```
if (FALSE) {
# Safe - connection auto-closes
users <- db_with("my_db", {
  DBI::dbGetQuery(conn, "SELECT * FROM users WHERE active = TRUE")
})

# Multiple operations with same connection
result <- db_with("my_db", {
  DBI::dbExecute(conn, "INSERT INTO users (name) VALUES ('Alice')")
  DBI::dbGetQuery(conn, "SELECT * FROM users")
})

# Connection closes even on error
tryCatch(
  db_with("my_db", {
    stop("Something went wrong")  # Connection still closes
  }),
  error = function(e) message(e$message)
)
}
```

---

docs_export *Export Package Documentation to Database*

Description

Parses roxygen2-generated .Rd files and exports structured documentation to SQLite (for GUI) or other formats. This enables searchable documentation in the Framework GUI and powers the public documentation website.

Usage

```
docs_export(
  output_path = "docs.db",
  man_dir = "man",
  package_name = "framework",
  package_version = NULL,
  include_internal = FALSE,
  verbose = TRUE
)
```

Arguments

output_path	Path to SQLite database file. Default: "docs.db"
man_dir	Directory containing .Rd files. Default: "man"

package_name	Package name for metadata. Default: "framework"
package_version	Package version for metadata. Default: NULL (auto-detect)
include_internal	Include internal/non-exported functions. Default: FALSE
verbose	Print progress messages. Default: TRUE

Details

The exporter reads all .Rd files from the man/ directory and extracts:

- Function name, title, description, details
- Arguments/parameters with descriptions
- Usage signatures
- Examples (with dontrun detection)
- See Also references
- Custom sections and subsections
- Keywords

The SQLite output includes FTS5 full-text search for fast querying.

Value

Invisibly returns the database connection path

Examples

```
if (FALSE) {
# Export to default location (exported functions only)
docs_export()

# Export to custom location
docs_export("inst/gui/docs.db")

# Include internal/private functions too
docs_export("all_docs.db", include_internal = TRUE)

# Query the exported docs
con <- DBI::dbConnect(RSQLite::SQLite(), "docs.db")
DBI::dbGetQuery(con, "SELECT name, title FROM functions WHERE name LIKE 'data_%'")
DBI::dbDisconnect(con)
}
```

env_clear *Clear R environment*

Description

Cleans up the R environment by removing objects, closing plots, detaching packages, and running garbage collection. Does not clear the console.

Usage

```
env_clear(keep = character(), envir)
```

Arguments

keep	Character vector of object names to keep (default: empty)
envir	The environment to clear

Value

Invisibly returns NULL

Examples

```
if (FALSE) {
# Clean a specific environment
env_clear(envir = my_env)

# Keep specific objects
env_clear(keep = c("config", "data"), envir = my_env)
}
```

env_summary *Summarize R environment*

Description

Displays a summary of the current R environment including loaded packages, objects in the global environment, and memory usage.

Usage

```
env_summary(envir)
```

Arguments

envir	The environment to summarize

Value

Invisibly returns a list with environment information

Examples

```
if (FALSE) {
env_summary(envir = my_env)
}
```

`fw_config_dir` *Get Framework config directory path*

Description

Returns the path to Framework's global configuration directory. Uses `tools::R_user_dir("framework", "config")` by default (CRAN compliant). Can be overridden with the `FW_CONFIG_HOME` environment variable.

Usage

```
fw_config_dir()
```

Value

Character string with the config directory path

`git_add` *Stage Files for Commit*

Description

Add file contents to the staging area.

Usage

```
git_add(files = ".")
```

Arguments

`files`	Character vector of file paths to stage, or "." for all (default)

Value

Invisibly returns TRUE on success

Examples

```
if (FALSE) {
git_add()              # Stage all changes
git_add("README.md")   # Stage specific file
git_add(c("R/foo.R", "R/bar.R"))
}
```

`git_commit` *Commit Staged Changes*

Description

Record changes to the repository with a commit message.

Usage

```
git_commit(message, all = FALSE)
```

Arguments

`message`	Commit message (required)
`all`	Logical; if TRUE, automatically stage modified/deleted files (default: FALSE)

Value

Invisibly returns TRUE on success

Examples

```
if (FALSE) {
git_commit("Fix bug in data loading")
git_commit("Update README", all = TRUE)  # Stage and commit
}
```

git_diff *Show Changes (Diff)*

Description

Show changes between commits, working tree, etc.

Usage

```
git_diff(staged = FALSE, file = NULL)
```

Arguments

staged	Logical; if TRUE, show staged changes (default: FALSE shows unstaged)
file	Optional file path to show diff for specific file

Value

Invisibly returns the diff output as a character vector

Examples

```
if (FALSE) {
git_diff()             # Show unstaged changes
git_diff(staged = TRUE) # Show staged changes
git_diff(file = "R/foo.R")
}
```

git_hooks_disable *Disable Specific Git Hook*

Description

Disables a specific hook in settings and reinstalls the pre-commit hook.

Usage

```
git_hooks_disable(hook_name, config_file = NULL, verbose = TRUE)
```

Arguments

hook_name	Name of hook: "ai_sync", "data_security", or "check_sensitive_dirs"
config_file	Path to configuration file (default: auto-discover settings.yml or settings.yml)
verbose	Logical; if TRUE (default), show messages

Value

Invisible TRUE on success

`git_hooks_enable` *Enable Specific Git Hook*

Description

Enables a specific hook in settings and reinstalls the pre-commit hook.

Usage

```
git_hooks_enable(hook_name, config_file = NULL, verbose = TRUE)
```

Arguments

`hook_name`	Name of hook: "ai_sync", "data_security", or "check_sensitive_dirs"
`config_file`	Path to configuration file (default: auto-discover settings.yml or settings.yml)
`verbose`	Logical; if TRUE (default), show messages

Value

Invisible TRUE on success

Examples

```
if (FALSE) {
git_hooks_enable("ai_sync")
git_hooks_enable("data_security")
}
```

`git_hooks_install` *Install Git Pre-commit Hook*

Description

Creates a pre-commit hook that runs Framework checks based on settings.yml settings.

Usage

```
git_hooks_install(config_file = NULL, force = FALSE, verbose = TRUE)
```

Arguments

config_file	Path to configuration file (default: "settings.yml")
force	Logical; if TRUE, overwrite existing hook (default: FALSE)
verbose	Logical; if TRUE (default), show installation messages

Details

Creates or updates `.git/hooks/pre-commit` to run enabled Framework hooks:

- **ai_sync**: Sync AI assistant context files before commit
- **data_security**: Run security audit to catch data leaks
- **check_sensitive_dirs**: Warn about unignored sensitive directories

Hook behavior is controlled by `git.hooks.*` settings in settings.yml.

Value

Invisible TRUE on success, FALSE on failure

Examples

```
if (FALSE) {
# Install hooks based on settings.yml
git_hooks_install()

# Force reinstall (overwrites existing hook)
git_hooks_install(force = TRUE)
}
```

---

git_hooks_list *List Git Hook Status*

Description

Shows which hooks are enabled and their current status.

Usage

```
git_hooks_list(config_file = NULL)
```

Arguments

config_file	Path to configuration file (default: auto-discover settings.yml or settings.yml)

Value

Data frame with hook information

git_hooks_uninstall *Uninstall Git Pre-commit Hook*

Description

Removes the Framework-managed pre-commit hook.

Usage

```
git_hooks_uninstall(verbose = TRUE)
```

Arguments

verbose	Logical; if TRUE (default), show messages

Value

Invisible TRUE if hook was removed, FALSE otherwise

git_log *Show Commit Log*

Description

Show recent commit history.

Usage

```
git_log(n = 10, oneline = TRUE)
```

Arguments

n	Number of commits to show (default: 10)
oneline	Logical; if TRUE, show condensed one-line format (default: TRUE)

Value

Invisibly returns the log output as a character vector

Examples

```
if (FALSE) {
git_log()
git_log(n = 5)
git_log(oneline = FALSE)  # Full format
}
```

git_pull *Pull from Remote*

Description

Fetch and integrate changes from the remote repository.

Usage

```
git_pull(remote = "origin", branch = NULL)
```

Arguments

remote	Remote name (default: "origin")
branch	Branch name (default: current branch)

Value

Invisibly returns TRUE on success

Examples

```
if (FALSE) {
git_pull()
git_pull(remote = "origin", branch = "main")
}
```

git_push *Push to Remote*

Description

Push commits to the remote repository.

Usage

```
git_push(remote = "origin", branch = NULL)
```

Arguments

remote	Remote name (default: "origin")
branch	Branch name (default: current branch)

Value

Invisibly returns TRUE on success

Examples

```
if (FALSE) {
git_push()
git_push(remote = "origin", branch = "main")
}
```

`git_security_audit` *Security audit for Framework projects*

Description

Performs a comprehensive security audit of data files in Framework projects, checking for unignored data files, git history leaks, and orphaned data files outside configured directories.

Usage

```
git_security_audit(
  config_file = NULL,
  check_git_history = TRUE,
  history_depth = "all",
  auto_fix = FALSE,
  verbose = TRUE,
 extensions = c("csv", "rds", "tsv", "txt", "dat", "xlsx", "xls", "sqlite", "db", "dta",
   "sav", "zsav", "por", "sas7bdat", "sas7bcat", "xpt", "parquet", "feather", "arrow",
    "json", "xml", "h5", "hdf5")
)
```

Arguments

`config_file`	Path to configuration file (default: auto-detect settings.yml/settings.yml)
`check_git_history`	Logical; if TRUE (default), check git history for leaked data files
`history_depth`	Character or numeric. "all" for full history, "shallow" for recent 100 commits, or numeric for specific commit count (default: "all")
`auto_fix`	Logical; if TRUE, automatically update .gitignore (default: FALSE)
`verbose`	Logical; if TRUE (default), show progress messages
`extensions`	Character vector of data file extensions to detect (default: common data formats)

Details

The security audit performs the following checks:

- **gitignore_coverage**: Verifies all private data files are in .gitignore
- **git_history**: Scans git history for accidentally committed data files
- **orphaned_files**: Finds data files outside configured directories
- **private_data_exposure**: Checks if private data is tracked by git
- **secrets_exposure**: Checks if `.env` files (which hold passwords and tokens) are tracked by git

Status levels:

- **pass**: No issues found
- **warning**: Potential issues that should be reviewed
- **fail**: Critical security issues requiring immediate action

Value

A structured list containing:

summary Data frame with check names, status (pass/warning/fail), and counts

findings List of data frames with detailed findings for each check

recommendations Character vector of actionable recommendations

audit_metadata List with audit timestamp, Framework version, and config info

Examples

```
if (FALSE) {
# Basic audit (report only)
audit <- git_security_audit()
print(audit$summary)
View(audit$findings$orphaned_files)

# Quick scan without git history
audit <- git_security_audit(check_git_history = FALSE)

# Verbose with limited git history
audit <- git_security_audit(history_depth = 100, verbose = TRUE)

# Auto-fix mode (updates .gitignore)
audit <- git_security_audit(auto_fix = TRUE)
}
```

git_status *Show Git Status*

Description

Display the working tree status from the R console.

Usage

```
git_status(short = FALSE)
```

Arguments

short	Logical; if TRUE, show short format (default: FALSE)

Value

Invisibly returns the status output as a character vector

Examples

```
if (FALSE) {
git_status()
git_status(short = TRUE)
}
```

gui *Launch Framework GUI*

Description

Opens a beautiful web-based interface for Framework with documentation, project management, and settings configuration.

Usage

```
gui(port = 8080, host = "127.0.0.1", browse = TRUE, route = NULL)
```

Arguments

port	Port number to use (default: 8080)
host	Host address to bind to. Default "127.0.0.1" for local access only. Use "0.0.0.0" to allow connections from other machines (requires appropriate network security).
browse	Automatically open browser (default: TRUE)
route	Initial route to open (default: NULL for home page)

Details

The GUI requires the optional 'plumber' package, which in turn depends on 'sodium' (and the system libsodium library). Everything else in Framework works without it.

Value

Invisibly returns the plumber server object

See Also

`setup()` for first-time configuration

Examples

```
if (FALSE) {
# Launch the GUI
framework::gui()

# Launch on specific port
framework::gui(port = 8888)

# Open directly to settings
framework::gui(route = "#/settings/basics")

# Run as standalone server (no browser, accessible from network)
framework::gui(port = 8080, host = "0.0.0.0", browse = FALSE)
}
```

---

load_settings_catalog *Read the Framework settings catalog*

Description

The catalog defines metadata (labels, hints) and default values for settings sections. Users can override the packaged defaults by placing a `settings-catalog.yml` file in their Framework config directory (`tools::R_user_dir("framework", "config")`). When an override exists it is merged on top of the packaged catalog.

Usage

```
load_settings_catalog(include_user = TRUE, validate = TRUE)
```

Arguments

`include_user`	Logical indicating whether to merge user overrides. Defaults to `TRUE`.
`validate`	Logical indicating whether to perform basic validation on the catalog structure. Defaults to `TRUE`.

Value

A nested list representing the settings catalog.

`make_notebook` *Create a Notebook or Script from Stub Template*

Description

Creates a new Quarto (.qmd), RMarkdown (.Rmd) notebook, or R script (.R) from stub templates. Searches for user-provided stubs first (in `stubs/` directory), then falls back to framework defaults.

Usage

```
make_notebook(
  name,
  type = NULL,
  dir = NULL,
  stub = "default",
  overwrite = FALSE,
  subdir = NULL
)
```

Arguments

`name`	Character. The file name. Extension determines type: • .qmd: Quarto notebook (default if no extension) • .Rmd: RMarkdown notebook • .R: R script Examples: `1-init`, `1-init.qmd`, `analysis.Rmd`, `script.R`
`type`	Character. File type: "quarto", "rmarkdown", or "script". Auto-detected from extension if provided. If NULL (default): 1. Checks config `default_notebook_format` (or legacy `options$default_notebook_format`) 2. Falls back to "quarto" (Framework is Quarto-first)
`dir`	Character. Directory to create the file in. Uses your project's configured `directories$notebooks` setting. Default: "notebooks/".
`stub`	Character. Name of the stub template to use. Defaults to "default". User can create custom stubs in `stubs/notebook-{stub}.qmd`, `stubs/notebook-{stub}.Rmd`, or `stubs/script-{stub}.R`.
`overwrite`	Logical. Whether to overwrite existing file. Default FALSE.
`subdir`	Optional subdirectory under `dir` (e.g., "analyses/exploratory").

Details

Convenient aliases: Use `make_qmd()` or `make_rmd()` for explicit Quarto or RMarkdown notebook creation. Use `make_revealjs()` or `make_presentation()` for reveal.js presentations.

Default Output:

Notebooks are created in the `notebooks/` directory by default:

```
notebooks/
  1-data-cleaning.qmd
  2-analysis.qmd
  3-visualization.qmd
```

Extension Normalization:

- If name includes `.qmd` or `.Rmd`, type is auto-detected
- If no extension provided, `.qmd` is used (Quarto-first)
- Use `type = "rmarkdown"` to default to `.Rmd`

Stub Template Resolution:

The function searches for stub templates in this order:

1. User stubs: `stubs/notebook-{stub}.qmd` or `stubs/notebook-{stub}.Rmd`
2. Framework stubs: `inst/stubs/notebook-{stub}.qmd` or `inst/stubs/notebook-{stub}.Rmd`

Custom stub templates can use placeholders:

- `{filename}` - The notebook filename without extension
- `{date}` - Current date (YYYY-MM-DD)

Value

Invisible path to created notebook

See Also

`make_qmd()`, `make_rmd()`, `make_revealjs()`, `make_presentation()`

Examples

```
if (FALSE) {
# Create notebooks/1-init.qmd (defaults to Quarto)
make_notebook("1-init")

# Create notebooks/analysis.Rmd (RMarkdown, extension-based)
make_notebook("analysis.Rmd")

# Explicit type parameter
make_notebook("report", type = "rmarkdown")

# Use custom stub template
make_notebook("report", stub = "minimal")

# Create in specific directory
```

```
make_notebook("explore", dir = "work")

# Convenient aliases (recommended for explicit types)
make_qmd("analysis")       # Always creates .qmd
make_rmd("report")         # Always creates .Rmd
make_revealjs("slides")    # Creates reveal.js presentation
make_presentation("deck")  # Alias for make_revealjs()
}
```

`make_presentation` *Create a Presentation*

Description

Alias for `make_revealjs()`. Creates a Quarto reveal.js presentation.

Usage

```
make_presentation(name, dir = NULL, overwrite = FALSE, subdir = NULL)
```

Arguments

`name`	Character. The presentation name (with or without .qmd extension)
`dir`	Character. Directory to create the file in. Uses your project's configured `directories$notebooks` setting. Default: "notebooks/".
`overwrite`	Logical. Whether to overwrite existing file. Default FALSE.
`subdir`	Optional subdirectory under `dir` (e.g., "slides/week-01").

Value

Invisible path to created presentation

See Also

`make_notebook()`, `make_revealjs()`

Examples

```
if (FALSE) {
# Create notebooks/deck.qmd with reveal.js format
make_presentation("deck")
}
```

make_qmd *Create a Quarto Notebook*

Description

Convenient alias for `make_notebook(type = "quarto")`. Creates a .qmd file from stub templates.

Usage

```
make_qmd(name, dir = NULL, stub = "default", overwrite = FALSE, subdir = NULL)
```

Arguments

`name`	Character. The file name (with or without .qmd extension)
`dir`	Character. Directory to create the file in. Uses your project's configured `directories$notebooks` setting. Default: "notebooks/".
`stub`	Character. Name of the stub template to use. Default "default".
`overwrite`	Logical. Whether to overwrite existing file. Default FALSE.
`subdir`	Optional subdirectory under `dir` (e.g., "slides/week-01").

Value

Invisible path to created notebook

See Also

`make_notebook()`, `make_rmd()`

Examples

```
if (FALSE) {
# Create notebooks/analysis.qmd
make_qmd("analysis")

# Use custom stub
make_qmd("report", stub = "minimal")

# Create in specific directory
make_qmd("explore", dir = "work")
}
```

make_revealjs *Create a Reveal.js Presentation*

Description

Convenient alias for creating reveal.js presentations. Always creates a Quarto notebook with the revealjs stub template.

Usage

```
make_revealjs(name, dir = NULL, overwrite = FALSE, subdir = NULL)
```

Arguments

name	Character. The presentation name (with or without .qmd extension)
dir	Character. Directory to create the file in. Uses your project's configured directories$notebooks setting. Default: "notebooks/".
overwrite	Logical. Whether to overwrite existing file. Default FALSE.
subdir	Optional subdirectory under dir (e.g., "slides/week-01").

Value

Invisible path to created presentation

See Also

make_notebook(), make_qmd(), make_presentation()

Examples

```
if (FALSE) {
# Create notebooks/slides.qmd with reveal.js format
make_revealjs("slides")

# Create in specific directory
make_revealjs("presentation", dir = "presentations")
}
```

make_rmd *Create an RMarkdown Notebook*

Description

Convenient alias for `make_notebook(type = "rmarkdown")`. Creates a .Rmd file from stub templates.

Usage

```
make_rmd(name, dir = NULL, stub = "default", overwrite = FALSE, subdir = NULL)
```

Arguments

`name`	Character. The file name (with or without .Rmd extension)
`dir`	Character. Directory to create the file in. Uses your project's configured `directories$notebooks` setting. Default: "notebooks/".
`stub`	Character. Name of the stub template to use. Default "default".
`overwrite`	Logical. Whether to overwrite existing file. Default FALSE.
`subdir`	Optional subdirectory under `dir` (e.g., "analyses/exploratory").

Value

Invisible path to created notebook

See Also

`make_notebook()`, `make_qmd()`

Examples

```
if (FALSE) {
# Create notebooks/analysis.Rmd
make_rmd("analysis")

# Use custom stub
make_rmd("report", stub = "minimal")

# Create in specific directory
make_rmd("explore", dir = "work")
}
```

make_script *Create an R Script from Stub Template*

Description

Convenience wrapper for `make_notebook()` that creates R scripts (.R files). This is identical to calling `make_notebook("name.R")`.

Usage

```
make_script(name, dir = NULL, stub = "default", overwrite = FALSE)
```

Arguments

`name`	Character. The script name (with or without .R extension). Examples: "process-data", "process-data.R"
`dir`	Character. Directory to create the script in. Uses your project's configured `directories$scripts` setting. Default: "scripts/".
`stub`	Character. Name of the stub template to use. Defaults to "default". User can create custom stubs in `stubs/script-{stub}.R`.
`overwrite`	Logical. Whether to overwrite existing file. Default FALSE.

Details

This function is a convenience wrapper that:

1. Ensures the name ends with .R extension
2. Uses `script_dir` config option instead of `notebook_dir`
3. Calls `make_notebook()` with `type = "script"`

Default Output:

Scripts are created in the `scripts/` directory by default:

```
scripts/
  process-data.R
  build-features.R
  run-model.R
```

Value

Invisible path to created script

See Also

`make_notebook()` for creating Quarto/RMarkdown notebooks

Examples

```
if (FALSE) {
# Create script (extension optional)
make_script("process-data")
make_script("process-data.R")

# Use custom stub
make_script("etl-pipeline", stub = "etl")

# Create in specific directory
make_script("analysis", dir = "analysis/")
}
```

---

new *Create a New Project (Master Wrapper)*

Description

Flexible project creation interface. Alias for `new_project()` that accepts type as a parameter.

Usage

```
new(
  name = NULL,
  location = NULL,
  type = "project",
  browse = interactive(),
  ...
)
```

Arguments

`name`	Project name. If NULL (default), prompts interactively.
`location`	Directory path where project will be created. If NULL (default), prompts interactively.
`type`	Project type. One of "project" (default), "project_sensitive", "course", or "presentation".
`browse`	Whether to open the project folder after creation (default: TRUE in interactive sessions)
`...`	Additional arguments passed to 'project_

Value

Invisibly returns the result from `project_create()`

See Also

new_project(), new_project_sensitive(), new_presentation(), new_course()

Examples

```
if (FALSE) {
# Create different project types
new("analysis", "~/projects/analysis")
new("study", "~/projects/study", type = "project_sensitive")
new("slides", "~/projects/slides", type = "presentation")
new("course-materials", "~/projects/course", type = "course")
}
```

---

new_course *Create a Course Project*

Description

Shorthand for new_project(..., type = "course"). Creates a project structured for teaching materials with slides, assignments, and modules.

Usage

```
new_course(name = NULL, location = NULL, browse = interactive(), ...)
```

Arguments

name	Project name. If NULL (default), prompts interactively.
location	Directory path where project will be created. If NULL (default), prompts interactively.
browse	Whether to open the project folder after creation (default: TRUE in interactive sessions)
...	Additional arguments passed to 'project_

Value

Invisibly returns the result from project_create()

See Also

new_project()

Examples

```
if (FALSE) {
new_course("stats-101", "~/projects/stats-101")
}
```

`new_presentation` *Create a Presentation Project*

Description

Shorthand for `new_project(..., type = "presentation")`. Creates a project optimized for RevealJS presentations.

Usage

```
new_presentation(name = NULL, location = NULL, browse = interactive(), ...)
```

Arguments

`name`	Project name. If NULL (default), prompts interactively.
`location`	Directory path where project will be created. If NULL (default), prompts interactively.
`browse`	Whether to open the project folder after creation (default: TRUE in interactive sessions)
`...`	Additional arguments passed to 'project_

Value

Invisibly returns the result from `project_create()`

See Also

`new_project()`

Examples

```
if (FALSE) {
new_presentation("quarterly-review", "~/projects/q4-review")
}
```

`new_project` *Create a New Framework Project*

Description

Convenience wrapper for creating Framework projects from the command line. Uses global settings configured via `setup()` as defaults, prompts for missing required values (name and location).

Usage

```
new_project(
  name = NULL,
  location = NULL,
  type = "project",
  browse = interactive(),
  ...
)
```

Arguments

`name`	Project name. If NULL (default), prompts interactively.
`location`	Directory path where project will be created. If NULL (default), prompts interactively.
`type`	Project type. One of "project" (default), "project_sensitive", "course", or "presentation".
`browse`	Whether to open the project folder after creation (default: TRUE in interactive sessions)
`...`	Additional arguments passed to 'project_

Details

This function is designed for the streamlined workflow:

```
remotes::install_github("frameworktools/framework")
framework::setup()        # One-time global configuration
framework::new_project()  # Create projects using saved defaults
```

Global settings from `tools::R_user_dir("framework", "config")` are used for:

- Author information (name, email, affiliation
- Default packages
- Directory structure
- Git settings
- AI assistant configuration
- Quarto format preferences

Value

Invisibly returns the result from `project_create()` (list with success, path, and project_id)

See Also

`setup()` for initial configuration, `project_create()` for full control

Examples

```
if (FALSE) {
# Interactive - prompts for name and location
new_project()

# With name and location specified
new_project("my-analysis", "~/projects/my-analysis")

# Create a sensitive data project
new_project("medical-study", "~/projects/medical", type = "project_sensitive")
}
```

---

new_project_sensitive *Create a Sensitive Data Project*

Description

Shorthand for `new_project(..., type = "project_sensitive")`. Creates a project with additional privacy protections for handling sensitive data.

Usage

```
new_project_sensitive(
  name = NULL,
  location = NULL,
  browse = interactive(),
  ...
)
```

Arguments

`name`	Project name. If NULL (default), prompts interactively.
`location`	Directory path where project will be created. If NULL (default), prompts interactively.
`browse`	Whether to open the project folder after creation (default: TRUE in interactive sessions)
`...`	Additional arguments passed to 'project_

Value

Invisibly returns the result from `project_create()`

See Also

`new_project()`

Examples

```
if (FALSE) {
new_project_sensitive("medical-study", "~/projects/medical")
}
```

`now` *Get current datetime*

Description

Get current datetime

Usage

```
now()
```

Value

Current datetime as an ISO 8601 formatted character string

`outputs` *Output Save Functions*

Description

First-class functions for saving tables, figures, models, and reports. These functions implement lazy directory creation with console feedback.

packages_install *Install packages from configuration*

Description

Installs all packages defined in the configuration that are not already installed. This is the same logic used in scaffold(), but exposed as a standalone function.

Usage

```
packages_install()
```

Value

Invisibly returns TRUE on success

Examples

```
if (FALSE) {
# Install all configured packages
packages_install()
}
```

packages_list *List all packages from configuration*

Description

Lists all packages defined in the configuration, showing the package name, version pin (if specified), and source (CRAN or GitHub).

Usage

```
packages_list()
```

Value

Invisibly returns NULL after printing package list

Examples

```
if (FALSE) {
# List all packages
packages_list()
}
```

packages_restore *Restore packages from renv.lock*

Description

Wrapper around `renv::restore()` that requires Framework's renv integration to be enabled first.

Usage

```
packages_restore(prompt = FALSE)
```

Arguments

`prompt`	Logical. If TRUE, renv prompts before restoring.

Value

Invisibly returns TRUE on success.

packages_snapshot *Snapshot current package library (renv)*

Description

Wrapper around `renv::snapshot()` that requires Framework's renv integration to be enabled first.

Usage

```
packages_snapshot(prompt = FALSE)
```

Arguments

`prompt`	Logical. If TRUE, renv prompts before writing the snapshot.

Value

Invisibly returns TRUE on success.

packages_status *Show renv package status*

Description

Wrapper around `renv::status()` that requires Framework's renv integration.

Usage

```
packages_status()
```

Value

The status object returned by `renv::status()`.

packages_update *Update packages from configuration*

Description

Updates packages defined in the configuration. If renv is enabled, uses renv::update(). Otherwise, reinstalls packages using standard installation methods.

Usage

```
packages_update(packages = NULL)
```

Arguments

packages	Character vector of specific packages to update, or NULL to update all

Value

Invisibly returns TRUE on success

Examples

```
if (FALSE) {
# Update all packages
packages_update()

# Update specific packages
packages_update(c("dplyr", "ggplot2"))
}
```

project_info *Display project structure information*

Description

Shows configured directories and their status (created or pending lazy creation). Useful for understanding the project structure and discovering available paths.

Usage

```
project_info(verbose = FALSE)
```

Arguments

verbose	If TRUE, shows additional details about each directory

Value

A data frame with directory information (invisibly)

Examples

```
if (FALSE) {
# Show project structure
project_info()

# Get detailed info
project_info(verbose = TRUE)
}
```

project_list *List all projects in global configuration*

Description

List all projects in global configuration

Usage

```
project_list()
```

Value

Data frame with project information

publish *Publishing Functions*

Description

Functions for publishing notebooks, data, and files to S3 storage.

Upload files or directories to an S3 bucket. This is the generic publishing function - use `publish_notebook()` for Quarto documents or `publish_data()` for data files.

Usage

```
publish(source, dest = NULL, connection = NULL, overwrite = TRUE)
```

Arguments

`source`	Character. Local file or directory path to upload.
`dest`	Character or NULL. Destination path in S3 bucket. If NULL, derives from source filename.
`connection`	Character or NULL. S3 connection name from config.yml. If NULL, uses the connection marked with `default: true`.
`overwrite`	Logical. Whether to overwrite existing files. Default TRUE.

Value

Character. The public URL(s) of uploaded file(s).

Examples

```
if (FALSE) {
# Upload a single file
publish("outputs/report.html")
# -> https://bucket.s3.region.amazonaws.com/prefix/report.html

# Upload with custom destination
publish("outputs/report.html", dest = "reports/q4-2024.html")

# Upload a directory
publish("outputs/charts/", dest = "reports/charts/")

# Use specific connection
publish("data.csv", connection = "s3_backup")
}
```

publish_data *Publish data to S3*

Description

Uploads a data frame or existing data file to S3.

Usage

```
publish_data(data, dest, format = "csv", connection = NULL, compress = FALSE)
```

Arguments

data	Data frame or character path to existing file.
dest	Character. Destination path in S3 (required for data frames).
format	Character. Output format when data is a data frame: "csv", "rds", "parquet", or "json". Default "csv".
connection	Character or NULL. S3 connection name, or NULL for default.
compress	Logical. Whether to gzip compress. Default FALSE.

Value

Character. Public URL of the published data.

Examples

```
if (FALSE) {
# Publish a data frame
publish_data(my_df, "datasets/customers.csv")

# Publish as RDS
publish_data(my_df, "datasets/customers.rds", format = "rds")

# Publish existing file
publish_data("outputs/model.rds", "models/v2/model.rds")
}
```

publish_dir *Publish a directory to S3*

Description

Recursively uploads all files in a directory to S3.

Usage

```
publish_dir(
  dir,
  dest = NULL,
  connection = NULL,
  pattern = NULL,
  recursive = TRUE
)
```

Arguments

dir	Character. Local directory path.
dest	Character or NULL. Destination prefix in S3. If NULL, uses the directory name.
connection	Character or NULL. S3 connection name, or NULL for default.
pattern	Character or NULL. Optional regex pattern to filter files.
recursive	Logical. Whether to include subdirectories. Default TRUE.

Value

Character vector. Public URLs of uploaded files.

Examples

```
if (FALSE) {
# Upload entire directory
publish_dir("outputs/dashboard/")

# Upload to specific location
publish_dir("outputs/dashboard/", dest = "dashboards/v2/")

# Upload only HTML files
publish_dir("outputs/", pattern = "\\.html$")
}
```

publish_list *List published files in S3*

Description

Lists files in an S3 bucket/prefix.

Usage

```
publish_list(prefix = NULL, connection = NULL, max = 1000L)
```

Arguments

prefix	Character or NULL. Prefix to filter by. If NULL, lists all files under the connection's configured prefix.
connection	Character or NULL. S3 connection name, or NULL for default.
max	Integer. Maximum number of files to list. Default 1000.

Value

Data frame with columns: key, size, last_modified.

Examples

```
if (FALSE) {
# List all published files
publish_list()

# List files under a prefix
publish_list("reports/")

# List from specific connection
publish_list(connection = "s3_backup")
}
```

publish_notebook *Publish a Quarto notebook to S3*

Description

Renders a Quarto document and uploads it to S3. The notebook is rendered to a temporary directory, uploaded, then cleaned up.

Usage

```
publish_notebook(
  file,
  dest = NULL,
  connection = NULL,
  self_contained = TRUE,
  format = "html",
  ...
)
```

Arguments

file	Character. Path to .qmd file.
dest	Character or NULL. Destination path in S3 (without extension). If NULL, derives from filename (e.g., "analysis.qmd" -> "analysis").
connection	Character or NULL. S3 connection name, or NULL for default.
self_contained	Logical. Whether to embed all resources. Default TRUE. Ignored if static_hosting: false (always renders self-contained).
format	Character. Output format. Default "html".
...	Additional arguments passed to quarto render.

Details

The URL format depends on the S3 connection's static_hosting setting:

- static_hosting: true -> uploads to dest/index.html, returns dest/
- static_hosting: false (default) -> uploads as dest.html, returns dest.html

Value

Character. Public URL of the published notebook.

Examples

```
if (FALSE) {
# With static_hosting: true -> returns /analysis/
# With static_hosting: false -> returns /analysis.html
publish_notebook("notebooks/analysis.qmd")

# Publish to specific location
publish_notebook("notebooks/analysis.qmd", dest = "reports/2024/q4")

# Publish non-self-contained (only with static_hosting: true)
publish_notebook("notebooks/analysis.qmd", self_contained = FALSE)
}
```

quarto_generate_all *Generate Quarto Configurations for Project*

Description

Main entry point for generating all `_quarto.yml` files in a project. Generates root config and directory-specific configs based on project type.

Usage

```
quarto_generate_all(
  project_path,
  project_type,
  render_dirs = NULL,
  quarto_settings = NULL,
  directories = NULL,
  root_output_dir = NULL
)
```

Arguments

`project_path`	Character. Path to project root
`project_type`	Character. One of "project", "project_sensitive", "course", "presentation"
`render_dirs`	Named list. Render directories with their paths
`quarto_settings`	List. Quarto settings (html and revealjs configs)
`directories`	Named list. Source directories keyed the same as render_dirs
`root_output_dir`	Optional output directory to set on the root _quarto.yml

Value

List with success status and paths of generated files

quarto_regenerate *Regenerate Quarto Configurations*

Description

Regenerates all `_quarto.yml` files in a project. **WARNING: This will overwrite any manual edits.** Should only be called when user explicitly requests regeneration.

Usage

```
quarto_regenerate(project_path, backup = TRUE)
```

Arguments

project_path	Character. Path to project root
backup	Logical. If TRUE, backs up existing files before overwriting. Default TRUE.

Value

List with success status, backed up files, and regenerated files

---

read_framework_template *Read the contents of a Framework template*

Description

Read the contents of a Framework template

Usage

```
read_framework_template(name)
```

Arguments

name	Template identifier (e.g., "notebook", "gitignore", "ai_claude")

Value

Character scalar containing template contents

---

read_frameworkrc *Read global Framework configuration*

Description

Read global Framework configuration

Usage

```
read_frameworkrc(use_defaults = TRUE)
```

Arguments

use_defaults	Whether to merge with default structure (default: TRUE)

Value

List containing global configuration

remove_project_from_config *Remove project from global configuration*

Description

Remove project from global configuration

Usage

```
remove_project_from_config(project_id)
```

Arguments

project_id	Project ID to remove

Value

Invisibly returns NULL

renv_disable *Disable renv for this project*

Description

Deactivates renv integration while preserving renv.lock for future use. Removes the `.framework_renv_enabled` marker file.

Usage

```
renv_disable(keep_renv = TRUE)
```

Arguments

keep_renv	Logical; if TRUE (default), keep renv.lock and renv/ directory

Value

Invisibly returns TRUE on success

Examples

```
if (FALSE) {
renv_disable()
}
```

renv_enable *Enable renv for this project*

Description

Initializes renv integration for the current Framework project. This:

- Creates `.framework_renv_enabled` marker file
- Initializes renv if not already initialized
- Syncs packages from settings.yml to renv.lock
- Updates .gitignore to exclude renv cache

Usage

```
renv_enable(sync = TRUE)
```

Arguments

sync	Logical; if TRUE (default), sync packages from settings.yml

Value

Invisibly returns TRUE on success

Examples

```
if (FALSE) {
renv_enable()
}
```

renv_enabled *Check if renv is enabled for this project*

Description

Determines whether renv integration is active by checking for the `.framework_renv_enabled` marker file in the project root.

Usage

```
renv_enabled()
```

Value

Logical indicating whether renv is enabled

Examples

```
if (renv_enabled()) {
  message("Using renv for package management")
}
```

---

reset_framework_template *Reset a Framework template back to the packaged default*

Description

Reset a Framework template back to the packaged default

Usage

```
reset_framework_template(name)
```

Arguments

name	Template identifier (e.g., "notebook", "gitignore", "ai_claude")

Value

Invisibly returns the file path of the reset template.

---

result_list *List saved results from the framework database*

Description

Retrieves a list of all saved results (tables, figures, models, reports, notebooks) that have been tracked via the save_* functions.

Usage

```
result_list(type = NULL, public = NULL)
```

Arguments

type	Optional filter by type: "table", "figure", "model", "report", "notebook"
public	Optional filter: TRUE for public results only, FALSE for private only

Value

A data frame with columns: name, type, public, comment, hash, created_at, updated_at. Returns an empty data frame if no results found or database unavailable.

Examples

```
if (FALSE) {
# List all results
result_list()

# List only tables
result_list(type = "table")

# List only public figures
result_list(type = "figure", public = TRUE)
}
```

`save_figure` *Save a figure to the outputs directory*

Description

Saves a ggplot2 plot or base R graphics to the configured figures directory. The directory is created lazily on first use.

Usage

```
save_figure(
  plot = NULL,
  name,
  format = "png",
  width = 8,
  height = 6,
  dpi = 300,
  public = FALSE,
  overwrite = TRUE,
  ...
)
```

Arguments

`plot`	A ggplot2 object, or NULL to save the current plot
`name`	The name for the output file (without extension)
`format`	Output format: "png" (default), "pdf", "svg", or "jpg"
`width`	Width in inches (default: 8)
`height`	Height in inches (default: 6)
`dpi`	Resolution in dots per inch (default: 300)
`public`	If TRUE, saves to public outputs directory (for project_sensitive type)
`overwrite`	If TRUE, overwrites existing files (default: TRUE)
`...`	Additional arguments passed to ggsave or the graphics device

Value

The path to the saved file (invisibly)

Examples

```
if (FALSE) {
# Save a ggplot
p <- ggplot(mtcars, aes(mpg, hp)) + geom_point()
save_figure(p, "mpg_vs_hp")

# Save as PDF for publication
save_figure(p, "mpg_vs_hp", format = "pdf", width = 10, height = 8)

# Save to public directory
save_figure(p, "summary_plot", public = TRUE)
}
```

---

save_model *Save a model to the outputs directory*

Description

Saves a fitted model object to the configured models directory. The directory is created lazily on first use.

Usage

```
save_model(model, name, public = FALSE, overwrite = TRUE, ...)
```

Arguments

model	A fitted model object (lm, glm, tidymodels workflow, etc.)
name	The name for the output file (without extension)
public	If TRUE, saves to public outputs directory (for project_sensitive type)
overwrite	If TRUE, overwrites existing files (default: TRUE)
...	Additional arguments passed to saveRDS()

Value

The path to the saved file (invisibly)

Examples

```
if (FALSE) {
# Fit and save a model
model <- lm(mpg ~ hp + wt, data = mtcars)
save_model(model, "mpg_regression")
}
```

`save_notebook` *Save a rendered notebook to the outputs directory*

Description

Renders a Quarto or R Markdown notebook and saves the output to the configured notebooks output directory. The directory is created lazily on first use.

Usage

```
save_notebook(
  file,
  name = NULL,
  format = "html",
  public = FALSE,
  overwrite = TRUE,
  embed_resources = TRUE,
  ...
)
```

Arguments

`file`	Path to the .qmd or .Rmd file to render
`name`	Optional new name for the output file (without extension). If NULL, uses the original notebook name.
`format`	Output format: "html" (default), "pdf", or "docx"
`public`	If TRUE, saves to public outputs directory (for project_sensitive type)
`overwrite`	If TRUE, overwrites existing files (default: TRUE)
`embed_resources`	If TRUE, creates a self-contained file with embedded resources (default: TRUE for html format)
`...`	Additional arguments passed to quarto render

Value

The path to the saved file (invisibly)

Examples

```
if (FALSE) {
# Render and save a notebook
save_notebook("notebooks/analysis.qmd")

# Save with a custom name
save_notebook("notebooks/analysis.qmd", name = "final_analysis")

# Render to PDF
save_notebook("notebooks/analysis.qmd", format = "pdf")

# Save to public directory (for sensitive projects)
save_notebook("notebooks/analysis.qmd", public = TRUE)
}
```

---

save_report *Save a report to the outputs directory*

Description

Copies or moves a rendered report (HTML, PDF, etc.) to the configured reports directory. The directory is created lazily on first use.

Usage

```
save_report(file, name = NULL, public = FALSE, overwrite = TRUE, move = FALSE)
```

Arguments

file	Path to the report file to save
name	Optional new name for the file (without extension). If NULL, uses original name.
public	If TRUE, saves to public outputs directory (for project_sensitive type)
overwrite	If TRUE, overwrites existing files (default: TRUE)
move	If TRUE, moves the file instead of copying (default: FALSE)

Value

The path to the saved file (invisibly)

Examples

```
if (FALSE) {
# Save a rendered HTML report
save_report("notebooks/analysis.html", "final_analysis")

# Save to public directory
save_report("notebooks/summary.pdf", "public_summary", public = TRUE)
}
```

`save_table` *Save a table to the outputs directory*

Description

Saves a data frame or tibble to the configured tables directory. The directory is created lazily on first use.

Usage

```
save_table(data, name, format = "csv", public = FALSE, overwrite = TRUE, ...)
```

Arguments

`data`	A data frame, tibble, or other tabular data
`name`	The name for the output file (without extension)
`format`	Output format: "csv" (default), "rds", "xlsx", or "parquet"
`public`	If TRUE, saves to public outputs directory (for project_sensitive type)
`overwrite`	If TRUE, overwrites existing files (default: TRUE)
`...`	Additional arguments passed to the underlying write function

Value

The path to the saved file (invisibly)

Examples

```
if (FALSE) {
# Save a simple table
save_table(my_results, "regression_results")

# Save as Excel
save_table(my_results, "regression_results", format = "xlsx")

# Save to public directory (for sensitive projects)
save_table(summary_stats, "summary", public = TRUE)
```

```
}
```

`scaffold` *Initialize and load the project environment*

Description

The primary entry point for working with Framework projects. Call this at the start of every notebook or script to set up your environment with all configured packages, functions, and settings.

Usage

```
scaffold(config_file = NULL)
```

Arguments

`config_file`	Path to configuration file. If NULL (default), automatically discovers settings.yml or config.yml in the project.

Details

`scaffold()` performs the following steps in order:

1. **Standardizes working directory** - Finds and sets the project root, even when called from notebooks in subdirectories
2. **Loads environment variables** - Reads secrets from `.env` file
3. **Loads configuration** - Parses settings.yml for project settings
4. **Sets random seed** - For reproducibility (if `seed` is configured)
5. **Installs packages** - Any missing packages from the `packages` list
6. **Loads packages** - Attaches all configured packages
7. **Sources functions** - Loads all `.R` files from `functions/` directory

After `scaffold()` completes, you have access to:

- All packages listed in settings.yml
- All functions from your `functions/` directory
- Settings via `settings("key")` helper function
- Database connections configured in your project

Value

Invisibly returns NULL. The main effects are side effects: loading packages, sourcing functions, and creating the `config` object.

Project Discovery

When called without arguments, `scaffold()` searches for a Framework project by:

- Looking for settings.yml or config.yml in current and parent directories
- Checking for .Rproj or .code-workspace files with nearby settings
- Recognizing common Framework subdirectories (notebooks/, scripts/, etc.)

This means you can call `scaffold()` from any subdirectory within your project.

Configuration

The settings.yml file controls what `scaffold()` loads. Key settings include:

- `packages`: List of R packages to install and load
- `seed`: Random seed for reproducibility
- `directories`: Custom directory paths
- `connections`: Database connection configurations

See Also

- `project_create()` to create a new Framework project
- `standardize_wd()` for just the working directory standardization
- `settings()` to access configuration values after scaffolding

Examples

```
if (FALSE) {
# At the top of every notebook or script:
library(framework)
scaffold()

# Now you can use your configured packages and functions
# Access settings via the settings() helper:
settings("directories.notebooks")
settings("seed")
}
```

`scratch_capture` *Capture and Save Data to File*

Description

Saves data to a file in various formats based on the object type and specified format. If no name is provided, uses the name of the object passed in. If no location is provided, uses the scratch directory from the configuration.

Usage

```
scratch_capture(x, name = NULL, to = NULL, location = NULL, n = Inf)
```

Arguments

`x`	The object to save
`name`	Optional character string specifying the name of the file (without extension). If not provided, will use the name of the object passed in.
`to`	Optional character string indicating the output format. One of: "text", "rds", "csv", "tsv". If not provided, will choose based on object type.
`location`	Optional character string specifying the directory where the file should be saved. If NULL, uses the scratch directory from the configuration.
`n`	Optional number of rows to capture for data frames (default: all rows)

Value

The input object `x` invisibly.

Examples

```
if (FALSE) {
# Save a character vector as text
scratch_capture(c("hello", "world"))

# Save a data frame as TSV
scratch_capture(mtcars)

# Save an R object as RDS
scratch_capture(list(a = 1, b = 2), to = "rds")
}
```

scratch_clean *Clean up the scratch directory by deleting all files*

Description

Clean up the scratch directory by deleting all files

Usage

```
scratch_clean()
```

Value

Invisibly returns NULL. Called for side effect of removing scratch files.

settings *Get settings value by dot-notation key*

Description

Framework's primary configuration helper that supports both flat and hierarchical key access using dot notation. Automatically checks common locations for directory settings. Pretty-prints nested structures in interactive sessions.

Usage

```
settings(key = NULL, default = NULL, settings_file = NULL)
```

Arguments

key	Character. Dot-notation key path (e.g., "notebooks" or "directories.notebooks" or "connections.db.host"). If NULL, returns entire settings.
default	Optional default value if key is not found (default: NULL)
settings_file	Settings file path (default: checks "settings.yml" then "settings.yml")

Details

For directory settings, the function checks multiple locations:

- Direct: `settings("notebooks")` checks `directories$notebooks`, then `options$notebook_dir`
- Explicit: `settings("directories.notebooks")` checks only `directories$notebooks`

File Discovery:

- Checks `settings.yml` first (Framework's preferred convention)
- Falls back to `settings.yml` if settings.yml not found

- You can override with explicit `settings_file` parameter

Output Behavior:

- Interactive sessions: Pretty-prints nested lists/structures and returns invisibly
- Non-interactive (scripts): Returns raw value without printing
- Simple values: Always returned directly without modification

Value

The settings value, or default if not found. In interactive sessions, nested structures are pretty-printed and returned invisibly.

Examples

```
if (FALSE) {
# Get notebook directory (checks both locations)
settings("notebooks")

# Get explicit nested setting
settings("directories.notebooks")
settings("connections.db.host")

# Get entire section
settings("directories")  # Returns all directory settings
settings("connections")  # Returns all connection settings

# View entire settings
settings()  # Returns full configuration

# With default value
settings("missing_key", default = "fallback")
}
```

`settings_read` *Read project settings*

Description

Reads the project settings from settings.yml or config.yml with environment-aware merging and split file resolution. Auto-discovers the settings file if not specified.

Usage

```
settings_read(settings_file = NULL, environment = NULL)
```

Arguments

settings_file	Path to settings file (default: auto-discover settings.yml or config.yml)
environment	Active environment name (default: R_CONFIG_ACTIVE or "default")

Value

The settings as a list

---

settings_write *Write project settings*

Description

Writes the project settings to settings.yml or config files

Usage

```
settings_write(settings, settings_file = NULL, section = NULL)
```

Arguments

settings	The settings list to write
settings_file	The settings file path (default: auto-detect settings.yml/config.yml)
section	Optional section to update (e.g. "data")

Value

Invisibly returns NULL.

---

setup *Setup Framework (First-Time Configuration)*

Description

Initializes Framework's global configuration and launches the GUI for first-time setup. This is the recommended entry point for new users.

Usage

```
setup(port = 8080, browse = TRUE)
```

Arguments

port	Port number to use (default: 8080)
browse	Automatically open browser (default: TRUE)

Details

Use this function after installing Framework to:

- Set your author name and email
- Configure default packages for new projects
- Set IDE preferences (VS Code, RStudio)
- Configure other global defaults

Value

Invisibly returns the plumber server object

See Also

`gui()` for launching the GUI without initialization check

Examples

```
if (FALSE) {
# First-time setup
framework::setup()
}
```

`standardize_wd` *Standardize Working Directory for Framework Projects*

Description

This function helps standardize the working directory when working with framework projects, especially useful in Quarto/RMarkdown documents that may be rendered from subdirectories.

Usage

```
standardize_wd(project_root = NULL)
```

Arguments

`project_root`	Character string specifying the project root directory. If NULL (default), the function will attempt to find it automatically.

Details

The function looks for common framework project indicators:

- settings.yml or settings.yml file
- .Rprofile file
- Being in common subdirectories (scratch, work)

It sets both the regular working directory and knitr's root.dir option if knitr is available.

Value

Invisibly returns the standardized project root path.

Examples

```
if (FALSE) {
library(framework)
standardize_wd()
scaffold()
}
```

`status` *Show Framework project status*

Description

Displays comprehensive information about the current Framework project including:

- Framework package version
- Project configuration
- Git status
- AI assistant configuration
- Git hooks status
- Package dependencies
- Directory structure

Usage

```
status()
```

Value

No return value, called for side effect of printing project status.

Examples

```
if (FALSE) {
status()
}
```

`storage_test` *Test storage connection*

Description

Validates that S3/storage credentials and bucket access are working.

Usage

```
storage_test(connection = NULL)
```

Arguments

`connection`	Character or NULL. Connection name, or NULL for default.

Value

Logical. TRUE if connection is valid.

Examples

```
if (FALSE) {
# Test default storage connection
storage_test()

# Test specific connection
storage_test("my_s3_backup")
}
```

`stubs_list` *List Available Stubs*

Description

Shows all available stub templates that can be used with `make_notebook()`.

Usage

```
stubs_list(type = NULL)
```

Arguments

type	Character. Filter by type: "quarto", "rmarkdown", "script", or NULL (all).

Value

Data frame with columns: name, type, source (user/framework)

Examples

```
if (FALSE) {
# List all stubs
stubs_list()

# List only Quarto stubs
stubs_list("quarto")

# List only script stubs
stubs_list("script")
}
```

---

stubs_path *Get Path to Stub Templates Directory*

Description

Returns the path to the user's stubs directory, or the framework stubs directory if no user stubs exist.

Usage

```
stubs_path(which = "auto")
```

Arguments

which	Character. Which directory to return: • "user" - User's project stubs directory (stubs/) • "framework" - Framework's built-in stubs directory • "auto" (default) - User directory if it exists, otherwise framework

Value

Character path to stubs directory

Examples

```
if (FALSE) {
# Get active stubs directory
stubs_path()

# Get framework stubs directory
stubs_path("framework")

# Get user stubs directory
stubs_path("user")
}
```

`stubs_publish` *Publish Stub Templates for Customization*

Description

Copies framework stub templates to your project's `stubs/` directory, allowing you to customize them. Similar to Laravel's `artisan vendor:publish` command.

Usage

```
stubs_publish(type = "all", overwrite = FALSE, stubs = NULL)
```

Arguments

`type`	Character vector. Which stub types to publish: • "notebooks" - Quarto/RMarkdown notebook stubs • "scripts" - R script stubs • "all" - All stubs (default)
`overwrite`	Logical. Whether to overwrite existing stubs. Default FALSE.
`stubs`	Character vector. Specific stub names to publish (e.g., "default", "minimal"). If NULL (default), publishes all stubs of the specified type.

Details

Stub Customization Workflow:

1. Publish stubs to your project: `stubs_publish()`
2. Edit stubs in `stubs/` directory to match your preferences
3. Use `make_notebook()` or `make_script()` - your custom stubs are used automatically

Stub Naming Convention:

Stubs follow this naming pattern:

- Notebooks: `stubs/notebook-{name}.qmd` or `stubs/notebook-{name}.Rmd`

- Scripts: `stubs/script-{name}.R`

Framework searches user stubs first, then falls back to built-in stubs.

Available Placeholders:

Stubs can use these placeholders:

- `{filename}` - File name without extension
- `{date}` - Current date (YYYY-MM-DD)

Value

Invisible list of published file paths

See Also

`make_notebook()`, `make_script()`, `stubs_list()`, `stubs_path()`

Examples

```
if (FALSE) {
# Publish all stubs
stubs_publish()

# Publish only notebook stubs
stubs_publish("notebooks")

# Publish specific stub
stubs_publish(stubs = "default")

# Overwrite existing stubs
stubs_publish(overwrite = TRUE)
}
```

`view` *View data in an interactive browser viewer*

Description

Opens an interactive, browser-based viewer for R objects. This is the primary function for viewing data frames, plots, lists, and other R objects with enhanced formatting.

Usage

```
view(x, title = NULL, max_rows = 5000)
```

Arguments

x	The data to view (data.frame, plot, list, function, or other R object)
title	Optional title for the view. If NULL, uses the object name.
max_rows	Maximum number of rows to display for data frames (default: 5000).

Value

Invisibly returns NULL. Opens a browser window.

Examples

```
if (FALSE) {
# View a data frame
view(mtcars)

# View with a title
view(iris, title = "Iris Dataset")

# View a ggplot
library(ggplot2)
p <- ggplot(mtcars, aes(mpg, hp)) + geom_point()
view(p)
}
```

---

write_framework_template

Overwrite a Framework template with new contents

Description

Overwrite a Framework template with new contents

Usage

```
write_framework_template(name, contents)
```

Arguments

name	Template identifier (e.g., "notebook", "gitignore", "ai_claude")
contents	Character string to write to the template file.

Value

Invisibly returns the file path of the written template.

`write_frameworkrc` *Write global Framework configuration*

Description

Write global Framework configuration

Usage

```
write_frameworkrc(config)
```

Arguments

`config`	List containing configuration to write

Value

Invisibly returns NULL

Index

add_project_to_config, 4
ai_generate_context, 4
ai_regenerate_context, 5
ai_skills, 6
ai_skills_update, 6
ai_sync_context, 7

cache, 8
cache_flush, 9
cache_forget, 9
cache_get, 10
cache_list, 10
cache_remember, 11
capture_output, 12
configure_global, 12
connections_list, 14
connections_s3, 14

data_add, 14
data_info, 15
data_list, 16
data_read, 17
data_save, 18
db_connect, 19
db_drivers_install, 19
db_drivers_status, 20
db_execute, 21
db_list, 22
db_query, 22
db_transaction, 23
db_with, 24
docs_export, 25

env_clear, 27
env_summary, 27

fw_config_dir, 28

git_add, 28
git_commit, 29
git_diff, 30
git_hooks_disable, 30
git_hooks_enable, 31
git_hooks_install, 31
git_hooks_list, 32
git_hooks_uninstall, 33
git_log, 33
git_pull, 34
git_push, 34
git_security_audit, 35
git_status, 37
gui, 37
gui(), *77*

load_settings_catalog, 38

make_notebook, 39
make_notebook(), *41–45*, *82*
make_presentation, 41
make_presentation(), *40*, *43*
make_qmd, 42
make_qmd(), *40*, *43*, *44*
make_revealjs, 43
make_revealjs(), *40, 41*
make_rmd, 44
make_rmd(), *40*, *42*
make_script, 45
make_script(), *82*

new, 46
new_course, 47
new_course(), *47*
new_presentation, 48
new_presentation(), *47*
new_project, 49
new_project(), *47, 48*, *51*
new_project_sensitive, 50
new_project_sensitive(), *47*
now, 51

outputs, 51

packages_install, 52
packages_list, 52
packages_restore, 53
packages_snapshot, 53
packages_status, 54
packages_update, 54
project_create(), *50*, *72*
project_info, 55
project_list, 55
publish, 56
publish_data, 57
publish_dir, 58
publish_list, 59
publish_notebook, 59

quarto_generate_all, 61
quarto_regenerate, 61

read_framework_template, 62
read_frameworkrc, 62
remove_project_from_config, 63
renv_disable, 63
renv_enable, 64
renv_enabled, 64
reset_framework_template, 65
result_list, 65

save_figure, 66
save_model, 67
save_notebook, 68
save_report, 69
save_table, 70
saveRDS(), *67*
scaffold, 71
scratch_capture, 73
scratch_clean, 74
settings, 74
settings(), *72*
settings_read, 75
settings_write, 76
setup, 76
setup(), *38*, *50*
standardize_wd, 77
standardize_wd(), *72*
status, 78
storage_test, 79
stubs_list, 79
stubs_list(), *82*
stubs_path, 80
stubs_path(), *82*
stubs_publish, 81

view, 82

write_framework_template, 83
write_frameworkrc, 84