# Package 'framework'

November 7, 2026

**Type** Package

**Title** Structured Data Science Project Scaffolding

**Version** 2.0.0

**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, openssl, lubridate, jsonlite, digest

**Suggests** bit64, gt, gtsummary, flextable, testthat (>= 3.0.0), arrow, aws.s3, aws.signature, BiocManager, cli, curl, cyclocomp, devtools, dplyr, DT, duckdb, ggplot2, haven, htmltools, htmlwidgets, httr2, IRkernel, keyring, jsonvalidate, askpass, knitr, languageserver, odbc, pool, prismjs, R.utils, readxl, remotes, renv, RMariaDB, rlang, rmarkdown, RPostgres, tm, usethis, withr, writexl, data.table, fixest, lattice, lme4, MASS, mgcv, nnet, patchwork, plotly, ragg, survival, svglite, tibble, vegawidget

**Config/testthat/edition** 3

**Depends** R (>= 4.1.0)

**Config/roxygen2/version** 8.1.0

**NeedsCompilation** no

**Author** Erik Westlund [aut, cre, cph]

**Repository** CRAN

**Date/Publication** 2026-11-07 17:30:09 UTC

# Contents

add_project_to_config . . . 4
adopt . . . 5
ai_generate_context . . . 6
ai_regenerate_context . . . 7
ai_skills . . . 8
ai_skills_update . . . 8
ai_sync_context . . . 9
bootstrap . . . 10
cache . . . 11
cache_flush . . . 12
cache_forget . . . 12
cache_get . . . 13
cache_list . . . 13
cache_remember . . . 14
capture_output . . . 15
cloud_login . . . 15
cloud_logout . . . 17
cloud_settings . . . 17
cloud_status . . . 18
cloud_sync . . . 18
configure_global . . . 19
config_sync . . . 20
connections_list . . . 21
connections_s3 . . . 21
data_add . . . 22
data_info . . . 23
data_list . . . 24
data_read . . . 24
data_save . . . 25
data_verify . . . 26
db_connect . . . 26
db_drivers_install . . . 27
db_drivers_status . . . 28
db_execute . . . 28
db_list . . . 29
db_query . . . 30
db_transaction . . . 30
db_with . . . 31
docs_export . . . 32
env_clear . . . 34
env_summary . . . 35
fw_config_dir . . . 35
git_add . . . 36
git_commit . . . 37
git_diff . . . 37
git_hooks_disable . . . 38

git_hooks_enable . . . 38
git_hooks_install . . . 39
git_hooks_list . . . 40
git_hooks_uninstall . . . 41
git_log . . . 41
git_pull . . . 42
git_push . . . 42
git_security_audit . . . 43
git_status . . . 45
gui . . . 45
ledger_list . . . 46
ledger_push . . . 46
load_settings_catalog . . . 47
make_notebook . . . 48
make_presentation . . . 50
make_qmd . . . 51
make_revealjs . . . 52
make_rmd . . . 53
make_script . . . 54
new . . . 55
new_course . . . 56
new_presentation . . . 57
new_project . . . 58
new_project_sensitive . . . 60
now . . . 61
outputs . . . 61
packages_install . . . 62
packages_list . . . 62
packages_restore . . . 63
packages_snapshot . . . 63
packages_status . . . 64
packages_update . . . 64
package_add . . . 65
package_sets_list . . . 66
package_set_add . . . 66
project_create_from_layers . . . 67
project_info . . . 68
project_list . . . 69
publish . . . 69
publish_data . . . 72
publish_dir . . . 73
publish_list . . . 75
publish_notebook . . . 76
quarto_generate_all . . . 77
quarto_regenerate . . . 78
read_frameworkrc . . . 79
read_framework_template . . . 79
remove_project_from_config . . . 80

renv_disable . . . . . 80
renv_enable . . . . . 81
renv_enabled . . . . . 82
reset_framework_template . . . . . 82
result_list . . . . . 83
save_figure . . . . . 83
save_model . . . . . 85
save_notebook . . . . . 86
save_report . . . . . 87
save_table . . . . . 88
scaffold . . . . . 89
scratch_capture . . . . . 91
scratch_clean . . . . . 92
settings . . . . . 92
settings_read . . . . . 93
settings_write . . . . . 94
setup . . . . . 94
skills_sync . . . . . 95
spec_check . . . . . 96
spec_resolve . . . . . 97
standardize_wd . . . . . 98
status . . . . . 99
storage_test . . . . . 99
structure_import . . . . . 100
stubs_list . . . . . 101
stubs_path . . . . . 101
stubs_publish . . . . . 102
view . . . . . 104
write_frameworkrc . . . . . 105
write_framework_template . . . . . 105

**Index** **106**

---

`add_project_to_config` *Add project to global configuration*

---

## Description

Adds a project folder to the registry (`projects.yml` in `fw_config_dir()`). The registry stores only the ID, the path, and the date. The project's name and type are read from its settings.yml when needed.

## Usage

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

### Arguments

| | |
|---|---|
| `project_dir` | Path to project directory |
| `project_name` | Not stored. Kept so older code still runs. |
| `project_type` | Not stored. Kept so older code still runs. |

### Value

Invisibly returns the project ID. For a folder already in the registry, that project's existing ID.

---

## `adopt` *Adopt an Existing Project Folder*

---

### Description

Turns a folder you already work in into a Framework project without moving anything. `adopt()` looks at the folder names (never file names or contents, and at most three levels deep), proposes which Framework directory each one is, and lets you accept, change a role, or leave a folder out. It then writes `settings.yml` (a single file) and `.framework/spec.json` describing the folders as they are. Nothing else is created, moved, or overwritten, and `scaffold()` works afterwards.

### Usage

```
adopt(path = ".", yes = FALSE)
```

### Arguments

| | |
|---|---|
| `path` | The project folder (default: the working directory). |
| `yes` | If TRUE, accept the proposed mapping without asking. Required in a non-interactive session. |

### Details

Folders whose path contains "private" are flagged: consider the Privacy Sensitive structure for that project, and keep those folders out of git.

When you are signed in to Framework Cloud and run `adopt()` interactively, it then offers to register the project (which writes its project token to `.env`, making sure git ignores `.env` first) and to save the layout as a structure. Both default to No.

### Value

Invisibly, a list with `directories` and `extra_directories` as written.

### See Also

`structure_import()` to save the layout as a structure on Framework Cloud.

## Examples

```
if (FALSE) {
adopt("~/projects/old-study")
adopt(yes = TRUE)
}
```

---

`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,
  skills = NULL,
  skills_dir = ".claude/skills"
)
```

## 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) |
| `skills` | Character vector of the skills installed in the project, for the skills table. NULL lists the default skills for `project_type`. |
| `skills_dir` | Folder the skills are installed in, for the paths in the skills table. |

## 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". A section is regenerated only when the file has its marked heading; files made by Framework 2.0 have no "functions" section. |
| `ai_file` | Name of the AI context file (default: `ai.canonical_file` from settings, then "AGENTS.md", then "CLAUDE.md" for older projects) |

## Value

Invisible TRUE on success

## Examples

```
if (file.exists("AGENTS.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 skill files Framework ships for the project's type (the `framework-*` skills and the ggplot2 skills) from the installed package version into `.claude/skills/`. 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

Locally edited skills and account-owned forks are preserved. Unchanged package skills receive the new package version. Custom skills are left alone.

### Value

Invisible character vector of the package skills now in the project (created, updated, or already up to date). Its `status` attribute names what happened to each skill, including preserved ones.

### Examples

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

`ai_sync_context` *Sync AI Assistant Context Files (Deprecated)*

## Description

Deprecated in framework 2.0. Projects now keep one `AGENTS.md`, which `CLAUDE.md` points to, plus skills, so there is nothing to sync.

## 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). A file that only points to the canonical file, such as a CLAUDE.md holding `@AGENTS.md`, is never overwritten. |
| `verbose` | Logical; if TRUE (default), show sync messages |

## Details

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

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)
}
```

---

`bootstrap` *Prepare a Cloned Project*

---

## Description

Gets a project ready to work in after `git clone`, on a new machine or a cluster. Every step is safe to repeat, and none contacts Framework Cloud:

## Usage

```
bootstrap(path = ".", interactive = base::interactive(), packages = TRUE)
```

## Arguments

| | |
|---|---|
| `path` | The project folder (default: the current directory). |
| `interactive` | If `TRUE`, ask for the values of empty `.env` variables the settings use. If `FALSE`, never prompt and list them as to-dos instead. |
| `packages` | If `FALSE`, skip package installation. |

## Details

1. Make sure `.gitignore` ignores `.env`.
2. Create `.env` from `.env.example` when it is missing, and list the empty variables that the project's settings use (through `env("NAME")`, `Sys.getenv("NAME")`, or `${NAME}`). Unused examples are not to-dos.
3. Create the project folders, which git does not carry when they are empty or ignored. Absolute paths (shared drives) are left alone, and so are folders under `outputs/`, which are made when first written to, as in a new project.
4. Create `framework.db`, or add the tables a newer version needs.
5. Install packages: `renv::restore()` when the project has `renv.lock`, otherwise the packages listed in settings.yml.
6. Install the git hooks enabled in settings.yml.

The Python package and `fw bootstrap` follow the same contract. From a shell: `Rscript -e 'framework::bootstrap(inte`
which exits with status 0 when the project is ready (even with to-dos left) and 1 when a step failed.

## Value

Invisibly, a list with `root`, `done` (the steps that changed something), and `todo` (things left to do by hand).

## See Also

new(), git_hooks_install(), packages_restore()

## Examples

```
if (FALSE) {
# After cloning a project
framework::bootstrap()

# In a job script on a cluster
framework::bootstrap(interactive = 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 | Not used. Expiry is set when the value is stored, with cache() or cache_remember(). Kept so older code still runs. |

### 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 "30 minutes", "2 hours", "1 day", or "1 week" are accepted (also seconds). |
| `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)

---

`cloud_login` *Log In to Framework Cloud*

---

## Description

Shows what connecting shares, asks for confirmation, then verifies the token and stores it in the native credential store for your platform – the macOS Keychain (through `/usr/bin/security`, as `fw` does), or Windows Credential Manager or Linux Secret Service (through the `keyring` package), falling back to a permission-restricted file in the Framework config directory. This is the whole onboarding:

## Usage

```
cloud_login(
  token = NULL,
  url = NULL,
  verify = TRUE,
  consent = NULL,
  browser = TRUE
)
```

## Arguments

| | |
|---|---|
| token | Your user token from Framework Cloud (Settings -> API tokens). Omit it to approve this computer in the browser. |
| url | The Framework Cloud server. Defaults to FW_CLOUD_URL, then the server you last logged in to, then https://framework.pub. Plain http is accepted only for localhost and 127.0.0.1. |
| verify | Check the token against the server before saving (default TRUE). |
| consent | Set TRUE to confirm non-interactively (scripts, CI). In an interactive session you are prompted instead. What connecting shares is printed first and documented at the server's /privacy page. |
| browser | Open the approval page when no token is given. FALSE prints the link only. |

## Details

```
framework::cloud_login("fw_...")

# A self-hosted server, or single-user mode on this machine
framework::cloud_login("fw_...", url = "https://framework.example.edu")
framework::cloud_login("fw_...", url = "http://localhost:8080")
```

Tokens are stored per server, and the server you log in to becomes this machine's default for cloud_status(), cloud_sync(), and new(). The FW_CLOUD_URL environment variable overrides that default.

Without a token, cloud_login() opens the server's approval page in your browser. Sign in, or create a free account, and approve this computer; the token comes back to R. With a token from the server's token page, cloud_login(token) stores that token instead.

## Value

Invisibly, where the token was stored ("keyring" or the file path).

## See Also

cloud_status(), cloud_logout(), cloud_settings()

## cloud_logout *Log Out of Framework Cloud*

### Description

Revokes this machine's stored token for a server on that server, then removes it from the system keyring and any token files. When `FW_TOKEN_FILE` is set, only that file is used, because it is the only place `cloud_login()` wrote. A token in `FW_CLOUD_TOKEN` is neither revoked nor removed. When the server can't be reached, the local copy is still removed and the token can be revoked on the server's tokens page. The default server saved by `cloud_login()` is kept.

### Usage

```
cloud_logout(url = NULL)
```

### Arguments

| | |
|---|---|
| `url` | The server to log out of. Defaults to the server `cloud_login()` would use. |

### Value

Invisibly, TRUE.

## cloud_settings *Fetch Your Cloud Settings*

### Description

Retrieves your settings document from Framework Cloud. These settings are overlaid on local defaults when creating projects with `new()`.

### Usage

```
cloud_settings()
```

### Value

A list with `revision`, `schema_version`, and `document`.

## cloud_status *Check Cloud Connection Status*

### Description

Reports which Framework Cloud server is in use and where that setting came from, whether a token is stored for it, and who the token authenticates as. Inside a project set up from a different server, the project's server is reported too.

### Usage

```
cloud_status(url = NULL)
```

### Arguments

| | |
|---|---|
| url | Check this server instead of the default. |

### Value

Invisibly, TRUE if connected, FALSE otherwise.

## cloud_sync *Sync Your Settings with Framework Cloud*

### Description

Your cloud settings document is mirrored to `cloud-settings.yml` in the Framework config directory. Edit it there (or on the website) and sync. Your main `settings.yml` is never touched.

### Usage

```
cloud_sync(direction = c("auto", "pull", "push"))
```

### Arguments

| | |
|---|---|
| direction | One of "auto", "pull", "push". |

### Details

- `pull` fetches the cloud document (backing up local edits first).
- `push` sends your local mirror to the cloud.
- `auto` (default) chooses: pull when the cloud is ahead, push when only your local mirror changed, and stops with instructions when both changed.

## Value

Invisibly, the current revision number.

## See Also

cloud_login(), cloud_settings()

---

`configure_global` *Configure Global Framework Settings*

---

## Description

Unified function for reading and writing global Framework settings. They are stored in `settings.yml` and the project registry in `projects.yml`, both in `fw_config_dir()`. 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", "project_sensitive", "presentation", "course", "bare")
  - `notebook_format` - Default notebook format ("quarto", "rmarkdown", "jupyter")
  - `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()
}
```

---

`config_sync` *Sync Settings from the Python Package and fw*

---

## Description

The Python package and the `fw` command-line tool keep their settings in `~/.config/framework`, and the R package keeps its own in `fw_config_dir()`. `config_sync()` brings R in line with them: files that are missing from R's folder, or newer in theirs, are copied into R's folder. This covers user settings, the saved Framework Cloud server, and token files. The cloud settings mirror and its sync state stay separate, because each tool keeps its own. R only reads their folder and never writes to it.

## Usage

```
config_sync(from = NULL, dry_run = FALSE)
```

## Arguments

| | |
|---|---|
| `from` | The folder to sync from. Defaults to `$XDG_CONFIG_HOME/framework`, or `~/.config/framework` when `XDG_CONFIG_HOME` is unset. |
| `dry_run` | Logical. List what would be copied without copying it. |

### Details

Logins stored in the system keyring are already shared and need no sync.

### Value

Invisibly, the relative paths copied (or that would be copied).

### See Also

`fw_config_dir()`, `cloud_login()`

### Examples

```
if (FALSE) {
config_sync(dry_run = TRUE)
config_sync()
}
```

---

## 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, a file in a configured inputs directory is named after the directory's key and the file name: `inputs/private/raw/participants.csv` in `inputs_private_raw` becomes `inputs.private.raw.participants`. Files elsewhere use the file name alone. |
| `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()
}
```

## 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:<br>• Dot notation: `inputs.raw.filename` resolves to inputs/raw/filename.rds. A name may end in the extension: `inputs.raw.filename.csv` is the name `inputs.raw.filename` saved as CSV.<br>• Direct path with a `/`: "inputs/raw/filename.csv". A relative path starts at the project root, not the working directory. framework.db records it relative to the project root with forward slashes, so "./inputs/raw/filename.csv" and "inputs/raw/filename.csv" are the same record.<br>A bare file name such as "filename.csv" is an error.<br>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.

## data_verify *Verify a Dataset Against the Integrity Ledger*

### Description

Hashes the file behind a catalog name (or a file path) and compares it against the local framework.db record and the project's cloud ledger on Framework Cloud. Proves "the data I have is bit-identical to what was recorded" without the data ever leaving the machine.

### Usage

```
data_verify(name)
```

### Arguments

| | |
|---|---|
| name | Data catalog name (e.g. "inputs.raw.enrollment") or a file path (e.g. "outputs/public/summary.csv"), as used with `data_read()` / `data_save()`. A path is relative to the project root. |

### Value

Invisibly, TRUE when no available record disagrees with the file. It prints which records were compared. With no framework.db record and no cloud ledger entry, nothing is compared: it says "Not verified" and still returns TRUE, as framework-py does.

## db_connect *Get a database connection*

### Description

Gets a database connection based on the connection name in settings.yml (or settings/connections.yml). SQLite and DuckDB paths are relative to the project root. 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 settings (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 settings.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 settings.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 settings.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 or other formats. This powers the searchable documentation website.

## Usage

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

## 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 functions that are not exported or are tagged `@keywords internal` (such as `.has_column()`). Default: FALSE |
| verbose | Print progress messages. Default: TRUE |
| reference_path | If set, also write the reference for the docs site (`docs/site/reference.json`) to this path, in the format of `inst/schema/docs-reference.v1.json`. Default: NULL |

## 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(tempfile(fileext = ".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. Framework itself stays attached.

## Usage

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

## Arguments

| | |
|---|---|
| `keep` | Character vector of object names to keep (default: empty) |
| `envir` | The environment to clear. Required, so nothing is removed by accident. Use `globalenv()` for your workspace. |

## Value

Invisibly returns NULL

## Examples

```
if (FALSE) {
# Clean your workspace
env_clear(envir = globalenv())

# 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. Use `globalenv()` for your workspace. |

## 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, which holds user settings, the saved Framework Cloud server, and token files. 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()
```

## Details

The Python package and the `fw` command-line tool keep their own folder (`~/.config/framework`). `config_sync()` brings their settings into R's.

## 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. Stops with git's message when nothing could be staged (a missing or ignored file, for example).

## 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: "data_security", "check_sensitive_dirs", or "check_notebook_outputs" |
| 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: "data_security", "check_sensitive_dirs", or "check_notebook_outputs" |
| 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("check_sensitive_dirs")
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

Installs Framework's pre-commit script where git looks for it (worktrees and `core.hooksPath` included). The script is shared with the Python package and `fw`: it finds an R or Python runtime and runs the hooks enabled in settings.yml, so enabling or disabling a hook later needs no reinstall. A pre-commit hook Framework wrote earlier is replaced; any other hook is kept unless `force = TRUE`. The enabled hooks:

- **data_security**: Run security audit to catch data leaks
- **check_sensitive_dirs**: Warn about unignored sensitive directories
- **check_notebook_outputs**: Block staged Jupyter notebooks that still hold cell outputs, which can contain data

Hook behavior is controlled by `git.hooks.*` settings in settings.yml. The `ai_sync` hook was removed in 2.0: skills and AGENTS.md replace it, and an `ai_sync` setting left in settings.yml is ignored.

## 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). When the branch has no upstream yet, it is pushed with `--set-upstream`, so later pulls and pushes need no arguments. |

## 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. Each is a row of `summary` (named by its `check`), and its details are in `findings` under the key in brackets:

- **gitignore_coverage** (`gitignore_issues`): Verifies all private data files are in .gitignore
- **git_history** (`git_history_issues`): Scans git history for accidentally committed data files
- **orphaned_files** (`orphaned_files`): Finds data files outside configured directories
- **private_data_exposure** (`private_data_exposure`): Checks if private data is tracked by git
- **secrets_exposure** (`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) |

### Details

For a project inside another git repository (the project folder is not the repository's top folder), it says so: the status is the enclosing repository's.

### Value

Invisibly returns the status output as a character vector

### Examples

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

## gui *Framework GUI (deprecated)*

### Description

The local GUI is retired. Framework Cloud at https://framework.pub manages projects, defaults, and settings.

### Usage

```
gui(...)
```

### Arguments

| | |
|---|---|
| ... | Ignored. |

### Value

Invisibly NULL, with a deprecation warning.

### Examples

```
if (FALSE) {
gui()
}
```

---

## ledger_list *List the Project's Cloud Ledger*

---

### Description

List the Project's Cloud Ledger

### Usage

```
ledger_list(name = NULL)
```

### Arguments

| | |
|---|---|
| name | Optional data catalog name to filter by. |

### Value

A data.frame of ledger entries, newest first.

---

## ledger_push *Import the Local Ledger into the Cloud*

---

### Description

Reads the project's local `framework.db` (the SQLite ledger of record that `data_save()` always maintains) and appends any digests the cloud ledger does not yet have. This is the default way project history reaches Framework Cloud: an explicit, reviewable action rather than a network call on every save.

### Usage

```
ledger_push()
```

## Details

For per-save streaming instead – tamper-evident, since each entry lands before the file could be altered – set `FW_LEDGER=live` or `options: cloud_ledger: live` in the project's `settings.yml`.

## Value

Invisibly, a list with `pushed` and `up_to_date` counts.

## See Also

`data_verify()`, `ledger_list()`

---

# 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:<br>• .qmd: Quarto notebook (default if no extension)<br>• .Rmd: RMarkdown notebook<br>• .R: R script Examples: `1-init`, `1-init.qmd`, `analysis.Rmd`, `script.R` |
| `type` | Character. File type: "quarto", "rmarkdown", "script", or "jupyter" (an .ipynb for the R kernel). Auto-detected from extension if provided. If NULL (default):<br>1. Checks the project's `scaffold.notebook_format` (or legacy `default_notebook_format` and `options$default_notebook_format`), ignoring "quarto" when the project turned the Quarto tool off<br>2. Falls back to "quarto", unless the project turned the Quarto tool off: then "jupyter" when the Jupyter tool is on, "rmarkdown" when rmarkdown is installed, and "script" otherwise |
| `dir` | Character. Directory to create the file in. Uses your project's configured `directories$notebooks` setting. Default: "notebooks/". A relative path starts at the project root, not the working directory. |
| `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")       # Creates .qmd
make_rmd("report")         # Creates .Rmd
# An extension in the name wins over the alias, as in the Python package:
# make_qmd("report.Rmd") creates an .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/". A relative path starts at the project root, not the working directory. |
| `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). Another extension wins: `"x.Rmd"` creates a .Rmd file. |
| `dir` | Character. Directory to create the file in. Uses your project's configured `directories$notebooks` setting. Default: "notebooks/". A relative path starts at the project root, not the working directory. |
| `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/". A relative path starts at the project root, not the working directory. |
| 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). Another extension wins: `"x.qmd"` creates a .qmd file. |
| `dir` | Character. Directory to create the file in. Uses your project's configured `directories$notebooks` setting. Default: "notebooks/". A relative path starts at the project root, not the working directory. |
| `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/". A relative path starts at the project root, not the working directory. |
| `stub` | Character. Name of the stub template to use. Defaults to "default", the only script stub Framework ships. Create custom stubs in `stubs/script-{stub}.R`, or save your default with `write_framework_template("script", ...)`. |
| `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 a custom stub: save it as stubs/script-etl.R first
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 = NULL,
  browse = interactive(),
  key = NULL,
  url = NULL,
  ...
)
```

## 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, as in `new_project()`. NULL (default): signed in to Framework Cloud, your default blueprint is used. Not signed in, an interactive session asks, with the Standard Project ("project") first, and a non-interactive one makes a "project". Pass `type = "bare"` for a bare project. |
| `browse` | Whether to open the project folder after creation (default: TRUE in interactive sessions) |

| | |
|---|---|
| key | A Framework Cloud project token. When given, the project's definition is fetched from the cloud (as with setup("fw_...")) and name and type come from it. The token is saved in the new project's .env for publish() and the ledger. |
| url | With key: the Framework Cloud server that issued the token. Defaults to FW_CLOUD_URL, then the server you last logged in to with cloud_login(), then https://framework.pub. |
| ... | Ignored, with a warning. |

## Details

new_project() is the preferred name. new() remains available for existing scripts. Its name can mask methods::new() when attached.

## 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")

# A project defined on Framework Cloud
new(key = "1|fw_abc123...", location = "~/projects/study")
}
```

---

# 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) |
| ... | Ignored, with a warning. |

### 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) |
| ... | Ignored, with a warning. |

## 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 your global settings (or your Framework Cloud defaults) as defaults, prompts for missing required values (name and location).

## Usage

```
new_project(
  name = NULL,
  location = NULL,
  type = NULL,
  browse = interactive(),
  tools = NULL,
  package_sets = NULL,
  skill_sets = NULL,
  workflow = NULL,
  offline = FALSE,
  key = NULL,
  url = NULL,
  ...
)
```

## 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 Framework's structures, "project", "project_sensitive", "course", "presentation", or "bare". NULL (default) uses your default blueprint on Framework Cloud when you are signed in, and "project" otherwise. |
| browse | Whether to open the project folder after creation (default: TRUE in interactive sessions) |
| tools | Tools to turn on, such as `"quarto"`, `"quarto_slides"`, `"targets"`, `"renv"`, or `"jupyter"`: a character vector, or a named logical that can also turn a structure's tool off, as in `c(quarto = FALSE, targets = TRUE)`. Tools a tool needs are added. NULL (default) keeps the structure's tools: Quarto for every structure except bare. |
| package_sets | Character vector of package sets to add, such as `"tidyverse-analysis"`. |
| skill_sets | Character vector of AI skill sets to install, such as `"framework"` and `"ggplot2"`. NULL (default) installs both. A sensitive project also gets the sensitive-data skill. |
| workflow | How notebooks are written: `"quarto"`, `"jupyter"` (Quarto stays on for rendering), or `"scripts"` (plain R scripts, no notebook tools). NULL (default) keeps your default, or the structure's: Quarto for every structure except bare. |
| offline | When you are signed in to Framework Cloud, `new_project()` sends your choices to it and builds the project it resolves. If the cloud can't be reached, your saved cloud defaults are used when you made no choices beyond the name. Otherwise an interactive session asks before using Framework's built-in defaults, and a non-interactive one stops. Successful projects are registered with their choices on Cloud and linked through a project token in the ignored `.env` file. `offline = TRUE` skips the cloud: saved defaults when there are any, else the built-ins. Not signed in, the built-ins are used. |
| key | Optional Framework Cloud project token. |
| url | Optional Framework Cloud server URL for `key`. |
| ... | Ignored, with a warning. |

## Details

This function is designed for the streamlined workflow:

```
remotes::install_github("table1/framework")
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

`cloud_login()` to use your Framework Cloud defaults, `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) |
| `...` | Ignored, with a warning. |

## 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 package status*

### Description

With Framework's renv integration on, a wrapper around `renv::status()`. With renv off, there is no lockfile to compare with, so it reports which packages in settings.yml are installed instead of stopping.

### Usage

```
packages_status()
```

### Value

With renv on, the status object returned by `renv::status()`. With renv off, invisibly a data frame with columns `package`, `installed`, and `version`, one row per package in settings.yml.

## 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"))
}
```

package_add *Add a package to the project*

## Description

Records a package in the project's settings (in `settings/packages.yml` when the project uses split settings files, otherwise in `settings.yml`) and installs it if it isn't installed yet. Recording the package is what makes it reproducible: `scaffold()` and `packages_install()` install everything listed, and auto-attached packages are loaded by `scaffold()`.

## Usage

```
package_add(package, auto_attach = FALSE, install = TRUE)
```

## Arguments

| | |
|---|---|
| package | Package specification: `"janitor"`, a pinned CRAN version (`"dplyr@1.1.0"`), a GitHub repository (`"tidyverse/dplyr@main"`), or a Bioconductor package (`"bioc::DESeq2"`). |
| auto_attach | Logical. Whether `scaffold()` loads the package. Default FALSE. |
| install | Logical. Install the package now if it isn't installed (or the installed version doesn't match a pin). Default TRUE. If the install fails, the settings are left as they were and `package_add()` stops. |

## Details

Adding a package that is already listed replaces its entry, so this is also how to pin a version or turn auto-attach on or off.

## Value

Invisibly, the package entry that was recorded.

## See Also

`packages_list()`, `packages_install()`

## Examples

```
if (FALSE) {
package_add("janitor")
package_add("forcats", auto_attach = TRUE)
package_add("tidyverse/dplyr@main")
package_add("dplyr@1.1.0")
}
```

## package_sets_list *List package sets*

### Description

Framework's built-in package sets for R projects. Add one to a project with `package_set_add()`, or at creation with `new_project(package_sets = ...)`.

### Usage

```
package_sets_list()
```

### Value

A data frame with `id`, `name`, `description`, and `packages` (the package names, comma-separated).

### See Also

`package_set_add()`

### Examples

```
package_sets_list()
```

## package_set_add *Add a package set to the project*

### Description

Adds a package set's packages to an existing project. Packages the project already lists are left as they are. The rest are recorded as `package_add()` records them, with each package's source, repository, and auto-attach setting from the set. Sets also stack at creation with `new_project(package_sets = ...)`.

### Usage

```
package_set_add(set, install = TRUE)
```

### Arguments

| | |
|---|---|
| `set` | Package set id or Framework Cloud handle. |
| `install` | Logical. Install the added packages now. Default TRUE. A failed install leaves the package recorded and warns. |

### Details

A set is one of Framework's built-in sets for R, such as `"tidyverse-analysis"` or `"bayes-r"`. See `package_sets_list()`. When you are signed in to Framework Cloud, a handle such as `"@yourname/lab-tools"` adds one of your account's sets.

### Value

Invisibly, a list with `added` and `present`: the names of the packages added and those the project already listed.

### See Also

`package_sets_list()`, `package_add()`

### Examples

```
if (FALSE) {
package_set_add("bayes-r")
package_set_add("tidy-omop", install = FALSE)
}
```

---

## project_create_from_layers

*Create a Project from a Layers File*

---

### Description

Reads a `.framework/layers.json` file (the format framework.pub sends), resolves it with `spec_resolve()`, and creates the project in `path`. This is the entry point the framework.pub GitHub Action runs, and it can also be used locally.

### Usage

```
project_create_from_layers(layers_file, path = ".", interactive = FALSE)
```

### Arguments

| | |
|---|---|
| `layers_file` | Path to a layers file (see `inst/schema/layers.v1.json`). |
| `path` | Directory to create the project in (default: current directory). |
| `interactive` | If `FALSE`, run headless as described above. |

### Details

With `interactive = FALSE` it runs headless, for automation: it never prompts, does not initialize git or make a commit (the caller owns the repository), does not add the project to the local registry, and refuses to write under `.github/workflows/`. In that mode `path` may be an existing git checkout that contains only `.git/`, `.github/`, `.framework/`, `README.md`, and `LICENSE`. Anything else in it is an error, so a misconfigured run cannot overwrite real work.

## Value

Invisibly, a list with `success`, `path`, and the resolved `spec`.

## See Also

`spec_resolve()`, `project_create()`

## Examples

```
if (FALSE) {
# In the framework.pub GitHub Action
project_create_from_layers(".framework/layers.json", path = ".", interactive = FALSE)
}
```

---

`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` | Has no effect. Every section is always shown, including the project name and type that `verbose` adds in the Python package. |

## Value

A data frame with directory information (invisibly)

## Examples

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

}
```

## project_list *List all projects in global configuration*

### Description

Lists the projects in the registry. A project whose folder no longer exists (such as one made in a temporary folder) is marked `exists = FALSE` and a message names it; remove it with `remove_project_from_config()`.

### Usage

```
project_list()
```

### Value

Data frame with columns `id`, `path`, `created`, and `exists`

## publish *Publishing Functions*

### Description

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

Publish files, directories, and R objects to named Cloud, GitHub, folder, or S3 destinations. `publish_notebook()` renders Quarto documents. `publish_data()` retains the low-level bucket export API.

### Usage

```
publish(
  source,
  dest = NULL,
  connection = NULL,
  overwrite = TRUE,
  self_contained = TRUE,
  slug = NULL,
  include_private = FALSE,
  to = NULL,
  title = NULL,
  description = NULL,
  kind = NULL,
  password = NULL,
  visibility = NULL,
  collection = NULL,
  share_unlisted = FALSE
)
```

## Arguments

`source` A local file or rendered directory path, or an R object. `publish()` converts these objects to files:

- Data frames, tibbles, data.tables, matrices, and `table` or `xtabs` contingency tables become a table with every row, a profile of each column, and variable labels from `attr(x, "label")`. Row names that aren't 1 to n become a first column named `row`.
- `anova()` results and `summary(aov)` become an ANOVA table.
- Hypothesis tests (`htest` objects from `t.test()`, `chisq.test()`, `wilcox.test()`, `cor.test()`, `prop.test()`, `fisher.test()`, `shapiro.test()`, and similar) become a test summary with the method, statistic, parameters, p-value, estimates, and confidence interval. The data name is left out.
- `lm`, `glm`, `aov`, `MASS::glm.nb`, `MASS::polr`, `nnet::multinom`, lme4 (`lmer`, `glmer`), `survival::coxph`, `survival::survreg`, fixest, and `mgcv::gam` models become coefficients with standard errors and 95% Wald intervals, fit statistics, and a description of the model. Mixed models add random effects and gam models add smooth terms. Models never carry observations, residuals, fitted values, or the model object itself.
- ggplot and patchwork plots, base graphics saved with `recordPlot()`, and lattice plots become an SVG figure (7 by 5 inches, white background) when svglite is installed, otherwise a 300 dpi PNG. An SVG larger than 5 MB is replaced by a PNG.
- plotly and other htmlwidgets become a self-contained HTML file, which needs pandoc (Quarto's copy is used when pandoc isn't on the PATH).
- gt, gtsummary, and flextable tables become HTML when their packages are installed.

Framework Cloud receives tables, tests, and models as structured JSON. Legacy storage uploads receive CSV tables, HTML summaries for `lm` and `glm`, and JSON for other models and tests. Named destinations receive typed bundles with offline readers and SHA-256 file manifests.

`dest` Character or NULL. Destination path in S3 bucket. If NULL, derives from source filename. On Framework Cloud, use `slug` instead; `dest` still names the document there when `slug` isn't a string, for older scripts.

`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.

`self_contained` Logical. When publishing to Framework Cloud, whether to bundle an HTML file with its resources. Ignored for S3. Default TRUE.

`slug` Framework Cloud only. A string names the document, for example `slug = "main-results"`. It does not change visibility: a new artifact is unlisted and an existing one keeps its visibility. `TRUE` makes the artifact public under the name derived from `dest` or the file name, and `FALSE` makes it unlisted. `NULL` (the default) publishes under the derived name. Ignored, with a warning, for S3. With `to = "github"`, a string names the folder under `published/`.

include_private
: Logical. Skip the private-path check. When the absolute path of any file to upload contains "private" in any case (apart from the /private/ macOS adds to /tmp and /var), Framework warns and, in an interactive session, asks before publishing (default No). In a non-interactive session it stops unless include_private = TRUE.

to
: Named destination from publish.destinations, or a driver name. publish.default selects the default destination. "folder" needs a named destination with a path. "s3" and "bucket" retain legacy storage uploads. "cloud" selects Framework Cloud even when a bucket is configured. "bucket" selects storage. "github" publishes to the project's GitHub Pages site instead of a storage bucket or Framework Cloud: the document is bundled under published/<slug>/. The artifact, version archives, and site index are committed and pushed. The site deploys from the default branch once "Publish with GitHub Pages" is on for the project in Framework Cloud, and it is public even when the repository is private, so an interactive session asks first. slug names the folder. Runs of characters outside lowercase letters and digits become -. The default is the file name lowercased, with other characters as - ("Final Report (v2).html" becomes final-report-v2). A folder needs an index.html at its top. dest, connection, and a TRUE/FALSE slug are errors here.

title
: Artifact title. Default: a Quarto or R Markdown document's YAML title, else the file name.

description
: Artifact description.

kind
: Optional artifact kind override.

password
: Password required to view the artifact on Framework Cloud.

visibility
: Artifact visibility: "public", "unlisted", or "private". Set "public" for a readable URL. An explicit visibility wins over a TRUE or FALSE slug. Without one, a new artifact is unlisted. Unsupported access controls fail before delivery. Private folder and S3 destinations use visibility: private and destination access controls. private: true remains a compatibility alias. S3 requires API confirmation of a non-public bucket policy and Public Access Block settings.

collection
: Existing project collection ID or slug. Follows the latest artifact version.

share_unlisted
: Explicit consent to share an unlisted artifact with collection readers.

## Details

With no connection and no storage bucket in settings.yml, a project set up from Framework Cloud publishes to its server instead (the FW_CLOUD_URL and FW_PROJECT_TOKEN in the project's .env). Each publish to Framework Cloud adds a new version of the document. Cloud provides owner-only access, public and unlisted links, and password protection. A new artifact is unlisted. Set visibility = "public" for a readable URL (/@handle/project/slug).

## Value

Character. Published URLs, or the local path for a folder without a URL.

## 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")

# Framework Cloud: a new version with a readable URL
publish("outputs/report.html", slug = "main-results", visibility = "public")

# GitHub Pages: commits published/main-results/ and pushes it
publish("outputs/report.html", to = "github", slug = "main-results")
}
```

---

`publish_data` *Publish data to S3 or a named destination*

---

## Description

Uploads a data frame or existing data file. Like `publish()`, it follows `to`, then `publish.default` in settings.yml. S3 destinations, an explicit `connection`, and projects without `publish.default` get a plain upload to the storage bucket at `dest`. Folder, GitHub, and Cloud destinations publish the file with `publish()`. The slug is `dest` without its extension, with each `/` turned into `-`: `"tables/demo.csv"` publishes as `tables-demo`. The message says which slug was used.

## Usage

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

### Arguments

| | |
|---|---|
| data | Data frame or character path to existing file. |
| dest | Character. Destination path in S3 (required for data frames). For other destinations, its file name names the file, and the path without its extension, with / turned into -, is the slug. |
| 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. |
| include_private | Logical. Skip the private-path check. When the absolute path of any file to upload contains "private" in any case (apart from the /private/ macOS adds to /tmp and /var), Framework warns and, in an interactive session, asks before publishing (default No). In a non-interactive session it stops unless include_private = TRUE. |
| to | Character or NULL. Named destination from publish.destinations, or a driver name, as in publish(). Default: publish.default. |

### Value

Character. Public URL of the published data (for a folder destination without a url, its location).

### 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 or a named destination*

---

### Description

Recursively uploads all files in a directory to S3. Like publish(), it follows to, then publish.default in settings.yml. S3 destinations, an explicit connection, and projects without publish.default get a file-by-file upload under dest. Folder, GitHub, and Cloud destinations publish the directory (after pattern and recursive) with publish(), using the last part of dest (or the directory name) as the slug.

## Usage

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

## 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. |
| include_private | Logical. Skip the private-path check. When the absolute path of any file to upload contains "private" in any case (apart from the /private/ macOS adds to /tmp and /var), Framework warns and, in an interactive session, asks before publishing (default No). In a non-interactive session it stops unless include_private = TRUE. |
| to | Character or NULL. Named destination from publish.destinations, or a driver name, as in publish(). Default: publish.default. |

## Value

Character vector. Public URLs of uploaded files (for other destinations, the published artifact's URL or location).

## 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 or a folder destination*

## Description

Lists files in an S3 bucket/prefix. Like publish(), it follows to, then publish.default in settings.yml. For a folder destination it lists the published artifacts, one row per slug. GitHub and Cloud destinations can't be listed here.

## Usage

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

## 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. |
| to | Character or NULL. Named destination from publish.destinations, or a driver name, as in publish(). Default: publish.default. |

## Value

Data frame with columns: key, size, last_modified, url.

## 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 or Framework Cloud*

## 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",
  slug = NULL,
  include_private = FALSE,
  to = NULL,
  collection = NULL,
  share_unlisted = FALSE,
  title = NULL,
  description = NULL,
  ...
)
```

## 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"). On Framework Cloud, use slug instead. |
| 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". |
| slug | Framework Cloud only: name the document. It does not make the document public. See publish(). With to = "github", names the folder under published/. |
| include_private | Logical. Skip the private-path check. When the absolute path of any file to upload contains "private" in any case (apart from the /private/ macOS adds to /tmp and /var), Framework warns and, in an interactive session, asks before publishing (default No). In a non-interactive session it stops unless include_private = TRUE. |

| | |
|---|---|
| to | Character or NULL. "cloud" selects Framework Cloud and "bucket" selects storage. "github" publishes the rendered HTML to the project's GitHub Pages site. See publish(). |
| collection | Existing project collection ID or slug. Follows the latest artifact version. |
| share_unlisted | Explicit consent to share an unlisted artifact with collection readers. |
| title | Character or NULL. The published title. Default: the document's YAML title, else its file name. |
| description | Character or NULL. A short description of the published document. |
| ... | Additional arguments passed to quarto render. |

## Details

Publishing follows publish(): a named connection or a configured storage bucket comes first, then Framework Cloud when the project was set up from it. Framework Cloud takes HTML output only, adds a new version on each publish, and uses slug as publish() does.

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 a project's Quarto configuration. Writes one root _quarto.yml with the shared settings and an output folder that keeps each render folder's path (notebooks/ renders to outputs/notebooks/). A folder that needs its own format, such as a course's slides/, gets a _metadata.yml, which inherits from the root. A render target outside the shared output folder keeps its own _quarto.yml.

### 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 | Ignored. Kept so older calls still work. The root output folder is the one most `render_dirs` share, as in the Python package. Setting it from `quarto.render_dir` would nest renders, such as `outputs/notebooks/notebooks/x.html`. |

### 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_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

## read_framework_template *Read the contents of a Framework template*

### Description

Templates are your versions of Framework's starter files, kept in the `templates/` folder of your Framework settings. Three are used: `"notebook"` by `make_notebook()` and `make_qmd()` with the default stub, `"script"` by `make_script()`, and `"presentation"` by `make_revealjs()`. A project's own `stubs/` folder still comes first. The gitignore and AI templates are kept for older settings and are not used by project creation.

### Usage

```
read_framework_template(name)
```

### Arguments

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

### Value

Character scalar containing template contents

## 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:

- Initializes renv if not already initialized
- Copies framework into the project's renv folder from the R installation in use. It installs framework from GitHub only when no installed copy is found.
- Installs the packages listed in settings.yml (with `sync = TRUE`)
- Writes renv.lock with every package in the project's renv folder, framework and the configured packages included
- Updates .gitignore to exclude renv cache
- Creates the `.framework_renv_enabled` marker file

## Usage

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

## Arguments

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

## Details

If a step fails, Framework undoes the steps before it: the marker file, the files renv created, and the `.gitignore` and `.renvignore` changes. The project is then as it was before the call. A package from settings.yml that fails to install gives a warning and is left out of renv.lock.

## 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, a recorded plot from grDevices::recordPlot(), 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 |

## Details

With plot = NULL, Framework saves the plot on the active graphics device, base graphics or ggplot2. Some devices keep no record of what they drew, such as the Rplots.pdf device in Rscript. There, a ggplot drawn last is saved with ggplot2::last_plot(), and a base plot drawn last is an error. Call dev.control("enable") before plotting, or pass recordPlot().

## 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 a base R plot
plot(mtcars$mpg, mtcars$hp)
save_figure(name = "mpg_vs_hp_base")
}
```

---

`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, R Markdown, or Jupyter 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, .Rmd, or .ipynb 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 |

## Details

Quarto renders a Jupyter notebook (.ipynb) from the outputs saved in it, without running its cells, so run the notebook before saving it.

## 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")

# A Jupyter notebook, rendered from its saved outputs
save_notebook("notebooks/analysis.ipynb")

# 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
8. **Sources scaffold.R** - Runs the project's `scaffold.R`, if there is one
9. **Makes the first git commit** - In a git repository with no commits yet, stages every file `.gitignore` allows and commits it as "Project initialized." This covers projects whose creation skipped that commit, for example when git had no user name or email. Repositories with commits are left alone.

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 and sourcing functions. No `config` object is created; read settings with `settings()` or `settings_read()`.

## 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*

### Description

Deletes the files at the top of the scratch directory, dotfiles included. Subfolders and their contents stay.

### 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 *Set up a Framework Cloud project, or Framework's global settings*

---

### Description

- `setup("fw_...")` with a **project token** from Framework Cloud pulls that project's definition and creates it locally. The project's `.env` records the token and its server (`FW_CLOUD_URL`), so `publish()` and the ledger go back to the server the project came from.
- `setup()` with no token creates Framework's global settings file if it is missing. Manage your defaults at `https://framework.pub` after `cloud_login()`.

### Usage

```
setup(token = NULL, browse = TRUE, location = NULL, url = NULL)
```

### Arguments

| | |
|---|---|
| token | A project token from Framework Cloud. NULL (default) sets up the global settings only. |
| browse | Open the new project's folder (default: TRUE) |
| location | Where to create the project (prompted interactively when NULL) |
| url | The Framework Cloud server that issued the token. Defaults to FW_CLOUD_URL, then the server you last logged in to with cloud_login(), then https://framework.pub. |

### Value

With a token, invisibly, the project_create() result. Without one, invisibly NULL.

### See Also

new(), cloud_login()

### Examples

```
if (FALSE) {
# Set up a cloud-defined project by its token
framework::setup("1|fw_abc123...")

# From a self-hosted server
framework::setup("1|fw_abc123...", url = "https://framework.example.edu")
}
```

---

## skills_sync *Synchronize a Skill Directory*

---

### Description

Synchronize a versioned skill bundle with Framework Cloud. Divergent changes return a conflict without replacing files. Companion files and executable modes are included. Untracked skills become private account skills.

### Usage

```
skills_sync(
  path,
  project_path = ".",
  direction = "auto",
  dry_run = FALSE,
  skill_id = NULL,
  variant = "r",
  resolve = NULL
)
```

### Arguments

| | |
|---|---|
| path | Project-relative skill directory. |
| project_path | Project root directory. |
| direction | One of auto, push, pull, or status. |
| dry_run | Report the action without writing files or Cloud versions. |
| skill_id | Explicit Cloud skill identity for adoption or an initial pull. |
| variant | Language variant: r, python, or stata. |
| resolve | Optional conflict choice: local or remote. Replaced files are backed up. |

### Value

A list with the action, skill identity, hashes, and version.

---

## spec_check *Check a Resolved Project Spec*

---

### Description

Checks a resolved spec for the problems `project_create()` cannot recover from: missing sections, unknown values, and paths that would write outside the project. The JSON Schema in `inst/schema/spec.v1.json` describes the same shape for other tools.

### Usage

```
spec_check(spec)
```

### Arguments

| | |
|---|---|
| spec | A resolved spec, as returned by `spec_resolve()`. |

### Value

A character vector of problems, empty when the spec is valid.

### See Also

`spec_resolve()`

spec_resolve *Resolve a Project Spec from Layers*

## Description

Merges spec layers into one complete project spec, the input to `project_create()`. This is the only implementation of the merge: the framework.pub cloud and the `fw` CLI send unmerged layers, and R resolves them.

## Usage

```
spec_resolve(layers = list(), name, author = NULL, check = TRUE)
```

## Arguments

| | |
|---|---|
| `layers` | A list of layers, bottom to top. Each element is either a layer (a named list in spec shape) or a list with `kind`, `key`, `version`, and `layer`, as in `.framework/layers.json`. |
| `name` | Project name. Overrides any `name` in the layers. |
| `author` | Optional author list (`name`, `email`, `affiliation`, and optionally `orcid`, `url`, and `affiliations`). Non-empty fields override the layers' `author`; `affiliations` replaces the whole list. |
| `check` | If `TRUE` (default), stop when the resolved spec is invalid. |

## Details

The package defaults for the project type are always the bottom layer. The given layers are applied on top, in order. The project type is taken from the last layer that sets `type` (default `"project"`).

Merge rules:

- Objects merge by key, and a later layer wins.
- A key set to `NULL` (JSON `null`) removes it.
- Entries of `files`, `skills`, `extra_directories`, `packages$default_packages`, `connections$databases`, and `connections$storage_buckets` are replaced whole, never merged field by field.
- `git$gitignore_content` is concatenated, with a comment naming the layer each block came from. The base gitignore template is chosen after merging, from the project type and `security$sensitive`.
- Empty lists are no-ops, so an empty JSON object and an empty JSON array mean the same thing.

## Value

The resolved spec as a named list. Attribute `"conflicts"` holds a data frame of keys set by more than one layer (`path`, `layers`).

## See Also

`project_create()`

## Examples

```
spec <- spec_resolve(
  list(list(security = list(sensitive = TRUE))),
  name = "My study"
)
spec$security$sensitive
```

---

`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")
}
```

---

`structure_import` *Save a Folder's Layout as a Structure on Framework Cloud*

---

## Description

Maps a folder's layout the same way as `adopt()` and saves it as a structure on Framework Cloud, private to you, so new projects can use it. Only the folder paths and the roles you confirm are sent; file names and contents never are. Changes nothing in the folder.

## Usage

```
structure_import(path = ".", name = NULL, yes = FALSE)
```

## Arguments

| | |
|---|---|
| `path` | The folder whose layout to import (default: the working directory). |
| `name` | The structure's name. Default: the folder's name. |
| `yes` | If TRUE, accept the proposed mapping without asking. |

## Value

Invisibly, the cloud's response: `key`, `name`, and `url`.

## Examples

```
if (FALSE) {
structure_import("~/projects/old-study", name = "Lab layout")
}
```

## 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` for the project's `stubs/`, `personal` for your templates, or `framework`), in the order `make_notebook()` searches. The first source hides later ones.

### 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:<br>• "user" - User's project stubs directory (stubs/)<br>• "framework" - Framework's built-in stubs directory<br>• "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:<br>• "notebooks" - Quarto/RMarkdown notebook stubs<br>• "scripts" - R script stubs<br>• "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_frameworkrc *Write global Framework configuration*

### Description

Writes the global settings file. When `config` has a `projects` element, the project registry (`projects.yml`) is written too; otherwise it stays as it is.

### Usage

```
write_frameworkrc(config)
```

### Arguments

| | |
|---|---|
| `config` | A named list, such as one from `read_frameworkrc()`. `NULL` or an empty list is an error, so a slip can't wipe the global settings. |

### Value

Invisibly returns NULL

## write_framework_template *Overwrite a Framework template with new contents*

### Description

See `read_framework_template()` for which templates are used.

### 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.

# Index


add_project_to_config, 4
adopt, 5
adopt(), *100*
ai_generate_context, 6
ai_regenerate_context, 7
ai_skills, 8
ai_skills_update, 8
ai_sync_context, 9

bootstrap, 10

cache, 11
cache(), *13*
cache_flush, 12
cache_forget, 12
cache_get, 13
cache_list, 13
cache_remember, 14
cache_remember(), *13*
capture_output, 15
cloud_login, 15
cloud_login(), *17*, *19*, *21*, *56*, *60*, *94, 95*
cloud_logout, 17
cloud_logout(), *16*
cloud_settings, 17
cloud_settings(), *16*, *19*
cloud_status, 18
cloud_status(), *16*
cloud_sync, 18
cloud_sync(), *16*
config_sync, 20
config_sync(), *36*
configure_global, 19
connections_list, 21
connections_s3, 21

data_add, 22
data_info, 23
data_list, 24
data_read, 24
data_read(), *26*
data_save, 25
data_save(), *26*
data_verify, 26
data_verify(), *47*
db_connect, 26
db_drivers_install, 27
db_drivers_status, 28
db_execute, 28
db_list, 29
db_query, 30
db_transaction, 30
db_with, 31
docs_export, 32

env_clear, 34
env_summary, 35

fw_config_dir, 35
fw_config_dir(), *4*, *19–21*

ggplot2::last_plot(), *84*
git_add, 36
git_commit, 37
git_diff, 37
git_hooks_disable, 38
git_hooks_enable, 38
git_hooks_install, 39
git_hooks_install(), *11*
git_hooks_list, 40
git_hooks_uninstall, 41
git_log, 41
git_pull, 42
git_push, 42
git_security_audit, 43
git_status, 45
grDevices::recordPlot(), *84*
gui, 45

ledger_list, 46

ledger_list(), *47*
ledger_push, 46
load_settings_catalog, 47

make_notebook, 48
make_notebook(), *50–54*, *79*, *103*
make_presentation, 50
make_presentation(), *49*, *52*
make_qmd, 51
make_qmd(), *49*, *52, 53*, *79*
make_revealjs, 52
make_revealjs(), *49, 50*, *79*
make_rmd, 53
make_rmd(), *49*, *51*
make_script, 54
make_script(), *79*, *103*

new, 55
new(), *11*, *16, 17*, *95*
new_course, 56
new_course(), *56*
new_presentation, 57
new_presentation(), *56*
new_project, 58
new_project(), *55–58*, *61*
new_project_sensitive, 60
new_project_sensitive(), *56*
now, 61

outputs, 61

package_add, 65
package_add(), *66, 67*
package_set_add, 66
package_set_add(), *66*
package_sets_list, 66
package_sets_list(), *67*
packages_install, 62
packages_install(), *65*
packages_list, 62
packages_list(), *65*
packages_restore, 63
packages_restore(), *11*
packages_snapshot, 63
packages_status, 64
packages_update, 64
project_create(), *60*, *68*, *90*, *97*
project_create_from_layers, 67
project_info, 68
project_list, 69
publish, 69
publish(), *56*, *72–77*, *94*
publish_data, 72
publish_dir, 73
publish_list, 75
publish_notebook, 76

quarto_generate_all, 77
quarto_regenerate, 78

read_framework_template, 79
read_framework_template(), *105*
read_frameworkrc, 79
read_frameworkrc(), *105*
remove_project_from_config, 80
remove_project_from_config(), *69*
renv_disable, 80
renv_enable, 81
renv_enabled, 82
reset_framework_template, 82
result_list, 83

save_figure, 83
save_model, 85
save_notebook, 86
save_report, 87
save_table, 88
saveRDS(), *85*
scaffold, 89
scaffold(), *65*
scratch_capture, 91
scratch_clean, 92
settings, 92
settings(), *90*
settings_read, 93
settings_read(), *90*
settings_write, 94
setup, 94
skills_sync, 95
spec_check, 96
spec_resolve, 97
spec_resolve(), *67, 68*, *96*
standardize_wd, 98
standardize_wd(), *90*
status, 99
storage_test, 99
structure_import, 100
structure_import(), *5*

stubs_list, 101
stubs_list(), *103*
stubs_path, 101
stubs_path(), *103*
stubs_publish, 102

view, 104

write_framework_template, 105
write_frameworkrc, 105