Package 'boilerplate'

May 7, 2026

Title Managing and Compiling Manuscript Templates

Version 1.3.0

Author Joseph Bulbulia [aut, cre] (ORCID:
<https://orcid.org/0000-0002-5861-2056>)

Maintainer Joseph Bulbulia <joseph.bulbulia@gmail.com>

Description Managing and generating standardised text for methods and results sections of scientific reports. It handles template variable substitution and supports hierarchical organisation of text through dot-separated paths. The package supports both RDS and JSON database formats, enabling version control and cross-language compatibility.

License MIT + file LICENSE

Encoding UTF-8

Depends R (>= 4.1)

Imports cli, digest, jsonlite, jsonvalidate, tools, utils

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

RoxygenNote 7.3.2

URL https://go-bayes.github.io/boilerplate/,
https://github.com/go-bayes/boilerplate

BugReports https://github.com/go-bayes/boilerplate/issues

NeedsCompilation no

Repository CRAN

Date/Publication 2025-06-24 09:10:06 UTC

Contents

`boilerplate_add_bibliography` *Add Bibliography to Database*

Description

Adds or updates bibliography information in a boilerplate database.

Usage

```
boilerplate_add_bibliography(db, url, local_path = NULL, validate = TRUE)
```

Arguments

db	Database object
url	URL to the bibliography file
local_path	Local filename to use when copying (default: basename of URL)
validate	Whether to validate citations on updates

Value

Updated database object

Examples

```
# Create temporary directory for example
temp_dir <- tempfile()
dir.create(temp_dir)

# Initialise and import
boilerplate_init(data_path = temp_dir, create_dirs = TRUE, confirm = FALSE, quiet = TRUE)
db <- boilerplate_import(data_path = temp_dir, quiet = TRUE)

# Add bibliography
# Using the example bibliography included with the package
example_bib <- system.file("extdata", "example_references.bib", package = "boilerplate")
db <- boilerplate_add_bibliography(
  db,
  url = paste0("file://", example_bib),
  local_path = "references.bib"
)

# Save the updated database
boilerplate_save(db, data_path = temp_dir, confirm = FALSE, quiet = TRUE)

# Clean up
unlink(temp_dir, recursive = TRUE)
```

---

boilerplate_appendix *Access Appendix Content*

Description

This function extracts and returns the appendix portion of a unified database, optionally retrieving a specific appendix section by name using dot notation.

Usage

```
boilerplate_appendix(unified_db, name = NULL)
```

Arguments

unified_db	List. The unified boilerplate database containing appendix content
name	Character. Optional specific appendix section to retrieve using dot notation (e.g., "supplementary.tables" for nested content)

Details

Access Appendix from Unified Database

Value

List or character. The requested appendix database or specific section. If name is NULL, returns the entire appendix database. If name is specified, returns the content at that path.

Examples

```
# Create a temporary directory and initialise database
temp_dir <- tempdir()
data_path <- file.path(temp_dir, "boilerplate_appendix_example", "data")

# Initialise with default appendix content
boilerplate_init(
  categories = "appendix",
  data_path = data_path,
  create_dirs = TRUE,
  create_empty = FALSE,
  confirm = FALSE,
  quiet = TRUE
)

# Import all databases
unified_db <- boilerplate_import(data_path = data_path, quiet = TRUE)

# Get all appendix sections
appendix_db <- boilerplate_appendix(unified_db)
names(appendix_db)

# Get a specific appendix section (if it exists)
if ("appendix" %in% names(unified_db) && length(unified_db$appendix) > 0) {
  section_names <- names(unified_db$appendix)
  if (length(section_names) > 0) {
    first_section <- boilerplate_appendix(unified_db, section_names[1])
  }
}

# Clean up
unlink(file.path(temp_dir, "boilerplate_appendix_example"), recursive = TRUE)
```

`boilerplate_batch_clean` *Batch Clean Fields in boilerplate Database*

Description

This function allows batch cleaning of text fields by removing or replacing specific characters or patterns across multiple entries in a boilerplate database.

Usage

```
boilerplate_batch_clean(
  db,
  field,
  remove_chars = NULL,
  replace_pairs = NULL,
  trim_whitespace = TRUE,
  collapse_spaces = FALSE,
  target_entries = NULL,
  exclude_entries = NULL,
  category = NULL,
  recursive = TRUE,
  preview = FALSE,
  confirm = TRUE,
  quiet = FALSE
)
```

Arguments

`db`	List. The database to clean (can be a single category or unified database).
`field`	Character. The field to clean (e.g., "reference", "description").
`remove_chars`	Character vector. Characters to remove from text fields.
`replace_pairs`	List. Named list for replacements (e.g., list(" " = "_")).
`trim_whitespace`	Logical. Whether to trim leading/trailing whitespace.
`collapse_spaces`	Logical. Whether to collapse multiple spaces to single space.
`target_entries`	Character vector. Entries to clean. Can be: • Specific entry names (e.g., c("ban_hate_speech", "born_nz")) • Patterns with wildcards (e.g., "anxiety*") • "all" to clean all entries • NULL to clean all entries with the specified field
`exclude_entries`	Character vector. Entries to exclude from cleaning. Can use specific names or wildcard patterns like target_entries

category	Character. If db is unified, specifies which category to clean.
recursive	Logical. Whether to search recursively through nested structures.
preview	Logical. If TRUE, shows what would be changed without making changes.
confirm	Logical. If TRUE, asks for confirmation before making changes.
quiet	Logical. If TRUE, suppresses non-essential messages.

Value

The modified database.

Examples

```
# First create a sample database
unified_db <- list(
  measures = list(
    test1 = list(reference = "@Smith2023[p.45]"),
    test2 = list(reference = "Jones[2022]")
  )
)

# Remove @, [, and ] from all references
unified_db <- boilerplate_batch_clean(
  db = unified_db,
  field = "reference",
  remove_chars = c("@", "[", "]"),
  category = "measures"
)

# Clean all entries EXCEPT specific ones
unified_db <- boilerplate_batch_clean(
  db = unified_db,
  field = "reference",
  remove_chars = c("@", "[", "]"),
  exclude_entries = c("forgiveness", "special_measure"),
  category = "measures"
)

# Clean specific entries only
unified_db <- boilerplate_batch_clean(
  db = unified_db,
  field = "reference",
  remove_chars = c("@", "[", "]"),
  target_entries = c("ban_hate_speech", "born_nz"),
  category = "measures"
)

# Clean all entries starting with "emp_" except "emp_special"
unified_db <- boilerplate_batch_clean(
  db = unified_db,
  field = "reference",
  remove_chars = c("@", "[", "]"),
```

```
  target_entries = "emp_*",
  exclude_entries = "emp_special",
  category = "measures"
)

# Replace characters and clean
unified_db <- boilerplate_batch_clean(
  db = unified_db,
  field = "reference",
  remove_chars = c("@", "[", "]"),
  replace_pairs = list(" " = "_", "." = ""),
  trim_whitespace = TRUE,
  category = "measures"
)

# Preview changes first
boilerplate_batch_clean(
  db = unified_db,
  field = "reference",
  remove_chars = c("@", "[", "]"),
  exclude_entries = "forgiveness",
  category = "measures",
  preview = TRUE
)
```

`boilerplate_batch_edit` *Batch Edit Fields in boilerplate Database*

Description

This function allows batch editing of specific fields across multiple entries in a boilerplate database. It supports pattern matching, explicit lists, and various editing operations.

Usage

```
boilerplate_batch_edit(
  db,
  field,
  new_value,
  target_entries = NULL,
  match_pattern = NULL,
  match_values = NULL,
  category = NULL,
  recursive = TRUE,
  case_sensitive = FALSE,
  preview = FALSE,
```

```
  confirm = TRUE,
  quiet = FALSE
)
```

Arguments

db	List or character. The database to edit (can be a single category or unified database), or a file path to a JSON/RDS database file which will be loaded automatically.
field	Character. The field to edit (e.g., "reference", "description").
new_value	Character. The new value to set for the field.
target_entries	Character vector. Entries to edit. Can be: • Specific entry names (e.g., c("ban_hate_speech", "born_nz")) • Patterns with wildcards (e.g., "anxiety*" matches all entries starting with "anxiety") • "all" to edit all entries • NULL to use pattern/value matching
match_pattern	Character. Pattern to match in the current field values. Only entries with matching values will be updated. Ignored if target_entries is specified.
match_values	Character vector. Specific values to match. Only entries with these exact values will be updated. Ignored if target_entries is specified.
category	Character. If db is unified, specifies which category to edit (e.g., "measures", "methods"). If NULL, attempts to detect.
recursive	Logical. Whether to search recursively through nested structures.
case_sensitive	Logical. Whether pattern matching is case sensitive.
preview	Logical. If TRUE, shows what would be changed without making changes.
confirm	Logical. If TRUE, asks for confirmation before making changes.
quiet	Logical. If TRUE, suppresses non-essential messages.

Value

The modified database (invisibly if preview=TRUE).

Examples

```
# Create a temporary directory and initialise database
temp_dir <- tempdir()
data_path <- file.path(temp_dir, "boilerplate_batch_edit_example", "data")

# Initialise database with default content
boilerplate_init(
  categories = "measures",
  data_path = data_path,
  create_dirs = TRUE,
  create_empty = FALSE,
  confirm = FALSE,
```

```
  quiet = TRUE
)

# Load database
unified_db <- boilerplate_import(data_path = data_path, quiet = TRUE)

# Example 1: Change specific references
unified_db <- boilerplate_batch_edit(
  db = unified_db,
  field = "reference",
  new_value = "example2024",
  target_entries = c("anxiety", "depression"),
  category = "measures",
  confirm = FALSE,
  quiet = TRUE
)

# Check the changes
unified_db$measures$anxiety$reference

# Example 2: Preview changes before applying
boilerplate_batch_edit(
  db = unified_db,
  field = "waves",
  new_value = "1-5",
  target_entries = "anxiety*",  # All entries starting with "anxiety"
  category = "measures",
  preview = TRUE,
  quiet = TRUE
)

# Example 3: Load database directly from file

# First save the database to a JSON file
json_file <- file.path(temp_dir, "boilerplate_unified.json")
boilerplate_save(unified_db, format = "json", data_path = temp_dir, quiet = TRUE)

# Now edit directly from the file
db <- boilerplate_batch_edit(
  db = json_file,  # File path instead of database object
  field = "description",
  new_value = "Updated description",
  target_entries = "anxiety",
  category = "measures",
  confirm = FALSE,
  quiet = TRUE
)


# Clean up
unlink(file.path(temp_dir, "boilerplate_batch_edit_example"), recursive = TRUE)
```

`boilerplate_batch_edit_multi` *Batch Edit Multiple Fields at Once*

Description

This function allows editing multiple fields across multiple entries in a single operation.

Usage

```
boilerplate_batch_edit_multi(
  db,
  edits,
  category = NULL,
  preview = FALSE,
  confirm = TRUE,
  quiet = FALSE
)
```

Arguments

`db`	List. The database to edit.
`edits`	List of lists. Each sub-list should contain: • field: The field to edit • new_value: The new value • target_entries: Which entries to edit (optional) • match_pattern: Pattern to match (optional) • match_values: Values to match (optional)
`category`	Character. Category to edit if db is unified.
`preview`	Logical. If TRUE, shows what would be changed.
`confirm`	Logical. If TRUE, asks for confirmation.
`quiet`	Logical. If TRUE, suppresses messages.

Value

List. The modified database with the batch edits applied.

Examples

```
# First create a sample database
unified_db <- list(
  measures = list(
    ban_hate_speech = list(reference = "old_ref", waves = "1-10"),
    born_nz = list(reference = "old_ref", waves = "1-10")
  )
```

```
)

# Update multiple fields for specific entries
unified_db <- boilerplate_batch_edit_multi(
  db = unified_db,
  edits = list(
    list(
      field = "reference",
      new_value = "sibley2021",
      target_entries = c("ban_hate_speech", "born_nz")
    ),
    list(
      field = "waves",
      new_value = "1-15",
      target_entries = c("ban_hate_speech", "born_nz")
    )
  ),
  category = "measures"
)
```

`boilerplate_check_health` *Check Database Health*

Description

Performs comprehensive health checks on a boilerplate database to identify potential issues such as orphaned variables, empty entries, and structural problems. Can optionally generate a detailed report.

Usage

```
boilerplate_check_health(db, fix = FALSE, report = NULL, quiet = FALSE)
```

Arguments

`db`	List. The database to check (can be unified or single category).
`fix`	Logical. If TRUE, attempts to fix issues where possible. Default FALSE.
`report`	Character or NULL. If a file path is provided, saves a detailed report. If "text", returns the report as a character string. If NULL (default), returns the health check object.
`quiet`	Logical. If TRUE, suppresses non-critical messages. Default FALSE.

Value

Depends on the `report` parameter:

- If `report = NULL`: A list with class "boilerplate_health" containing summary, issues, stats, and fixed items
- If `report = "text"`: A character string containing the detailed report
- If `report` is a file path: Invisibly returns the file path after saving report

Examples

```
# Create temporary directory for example
temp_dir <- tempfile()
dir.create(temp_dir)

# Initialise and import database
boilerplate_init(data_path = temp_dir, create_dirs = TRUE,
                 confirm = FALSE, quiet = TRUE)
db <- boilerplate_import(data_path = temp_dir, quiet = TRUE)

# Check database health
health <- boilerplate_check_health(db)
print(health)

# Generate text report
report_text <- boilerplate_check_health(db, report = "text")
cat(report_text)

# Save report to file
report_file <- file.path(temp_dir, "health_report.txt")
boilerplate_check_health(db, report = report_file)

# Check and fix issues
health <- boilerplate_check_health(db, fix = TRUE)

# Get the fixed database
if (health$summary$issues_fixed > 0) {
  db <- attr(health, "fixed_db")
}

# Clean up
unlink(temp_dir, recursive = TRUE)
```

`boilerplate_copy_bibliography`
Copy Bibliography to Project Directory

Description

Copies the bibliography file from cache to a specified directory, typically for use with Quarto documents.

Usage

```
boilerplate_copy_bibliography(
  db,
  target_dir = ".",
  overwrite = TRUE,
  update_first = FALSE,
  quiet = FALSE
)
```

Arguments

db	Database object containing bibliography information
target_dir	Directory to copy the bibliography file to. Default is current directory.
overwrite	Logical. Whether to overwrite existing file
update_first	Logical. Whether to update from remote before copying
quiet	Logical. Suppress messages

Value

Path to the copied bibliography file, or NULL if operation failed

Examples

```
# Create temporary directory for example
temp_dir <- tempfile()
dir.create(temp_dir)

# Initialise and import
boilerplate_init(data_path = temp_dir, create_dirs = TRUE, confirm = FALSE, quiet = TRUE)
db <- boilerplate_import(data_path = temp_dir, quiet = TRUE)

# Copy bibliography
boilerplate_copy_bibliography(db, temp_dir)

# Clean up
unlink(temp_dir, recursive = TRUE)
```

`boilerplate_copy_from_project` *Copy Elements from One Project to Another*

Description

This function allows selective copying of boilerplate elements from one project to another, enabling controlled sharing of content between projects.

Usage

```
boilerplate_copy_from_project(
  from_project,
  to_project,
  paths = NULL,
  prefix = NULL,
  data_path = NULL,
  merge_strategy = c("skip", "overwrite", "rename"),
  confirm = TRUE,
  quiet = FALSE
)
```

Arguments

`from_project`	Character. Name of the source project.
`to_project`	Character. Name of the destination project.
`paths`	Character vector. Paths to copy (e.g., c("measures.anxiety", "methods.sampling")). If NULL, copies everything (requires confirmation).
`prefix`	Character. Optional prefix to add to copied entries to avoid conflicts. For example, prefix = "colleague_" would rename "anxiety" to "colleague_anxiety".
`data_path`	Character. Base path for boilerplate data. If NULL, uses default location.
`merge_strategy`	Character. How to handle conflicts: "skip", "overwrite", or "rename".
`confirm`	Logical. If TRUE, asks for confirmation. Default is TRUE.
`quiet`	Logical. If TRUE, suppresses messages. Default is FALSE.

Value

Invisibly returns the updated database from the destination project.

Examples

```
# Create temporary directory for example
temp_dir <- tempfile()
dir.create(temp_dir)
```

```
# Initialise two projects
boilerplate_init(data_path = temp_dir, project = "colleague_measures",
                 create_dirs = TRUE, confirm = FALSE, quiet = TRUE)
boilerplate_init(data_path = temp_dir, project = "my_research",
                 create_dirs = TRUE, confirm = FALSE, quiet = TRUE)

# Add some content to source project
source_db <- boilerplate_import(data_path = temp_dir,
                                project = "colleague_measures", quiet = TRUE)
source_db$measures$anxiety <- list(name = "Anxiety Scale", items = 10)
source_db$measures$depression <- list(name = "Depression Scale", items = 20)
boilerplate_save(source_db, data_path = temp_dir,
                 project = "colleague_measures", confirm = FALSE, quiet = TRUE)

# Copy specific measures from colleague's project
boilerplate_copy_from_project(
  from_project = "colleague_measures",
  to_project = "my_research",
  paths = c("measures.anxiety", "measures.depression"),
  prefix = "smith_",
  data_path = temp_dir,
  confirm = FALSE,
  quiet = TRUE
)

# Clean up
unlink(temp_dir, recursive = TRUE)
```

`boilerplate_discussion` *Access Discussion Content*

Description

This function extracts and returns the discussion portion of a unified database, optionally retrieving a specific discussion section by name using dot notation.

Usage

```
boilerplate_discussion(unified_db, name = NULL)
```

Arguments

`unified_db`	List. The unified boilerplate database containing discussion content
`name`	Character. Optional specific discussion section to retrieve using dot notation (e.g., "limitations.statistical" for nested content)

Details

Access Discussion from Unified Database

Value

List or character. The requested discussion database or specific section. If name is NULL, returns the entire discussion database. If name is specified, returns the content at that path.

Examples

```
# Create a temporary directory and initialise database
temp_dir <- tempdir()
data_path <- file.path(temp_dir, "boilerplate_discussion_example", "data")

# Initialise with default discussion content
boilerplate_init(
  categories = "discussion",
  data_path = data_path,
  create_dirs = TRUE,
  create_empty = FALSE,
  confirm = FALSE,
  quiet = TRUE
)

# Import all databases
unified_db <- boilerplate_import(data_path = data_path, quiet = TRUE)

# Get all discussion sections
discussion_db <- boilerplate_discussion(unified_db)
names(discussion_db)

# Get a specific discussion section (if it exists)
if ("discussion" %in% names(unified_db) && length(unified_db$discussion) > 0) {
  section_names <- names(unified_db$discussion)
  if (length(section_names) > 0) {
    first_section <- boilerplate_discussion(unified_db, section_names[1])
  }
}

# Clean up
unlink(file.path(temp_dir, "boilerplate_discussion_example"), recursive = TRUE)
```

---

boilerplate_export *Export boilerplate Database*

Description

This function exports a boilerplate database or selected elements to disk. It supports exporting entire databases, specific categories, or selected elements using dot notation paths. Can export in RDS, JSON, or both formats.

Usage

```
boilerplate_export(
  db,
  output_file = NULL,
  select_elements = NULL,
  data_path = NULL,
  format = "rds",
  confirm = TRUE,
  create_dirs = FALSE,
  quiet = FALSE,
  pretty = TRUE,
  save_by_category = TRUE,
  project = "default"
)
```

Arguments

db	List. The database to export (unified or single category).
output_file	Character. Optional output filename. If NULL, uses default naming.
select_elements	Character vector. Optional paths to export (supports wildcards).
data_path	Character. Base path for data directory. If NULL (default), uses tools::R_user_dir("boilerplate", "data").
format	Character. Format to save: "rds" (default), "json", or "both".
confirm	Logical. If TRUE (default), asks for confirmation before overwriting.
create_dirs	Logical. If TRUE, creates directories if they don't exist.
quiet	Logical. If TRUE, suppresses all CLI alerts. Default is FALSE.
pretty	Logical. If TRUE (default), pretty-print JSON for readability.
save_by_category	Logical. If TRUE (default), saves unified databases by category.
project	Character. Project name for organizing databases. Default is "default". Projects are stored in separate subdirectories to allow multiple independent boilerplate collections.

Value

Invisible TRUE if successful.

Examples

```
# Create a temporary directory and initialise databases
temp_dir <- tempdir()
data_path <- file.path(temp_dir, "boilerplate_export_example", "data")

# Initialise and import databases
boilerplate_init(
```

```
  categories = c("methods", "measures"),
  data_path = data_path,
  create_dirs = TRUE,
  confirm = FALSE,
  quiet = TRUE
)

unified_db <- boilerplate_import(data_path = data_path, quiet = TRUE)

# Export entire database
export_path <- file.path(temp_dir, "export")
boilerplate_export(
  db = unified_db,
  data_path = export_path,
  create_dirs = TRUE,
  confirm = FALSE,
  quiet = TRUE
)

# Export selected elements
boilerplate_export(
  db = unified_db,
  select_elements = "methods.*",
  output_file = "methods_only.rds",
  data_path = export_path,
  confirm = FALSE,
  quiet = TRUE
)

# Clean up
unlink(file.path(temp_dir, "boilerplate_export_example"), recursive = TRUE)
```

`boilerplate_find_chars` *Find Entries with Specific Characters in Fields*

Description

This helper function finds all entries containing specific characters in a field. Useful for identifying which entries need cleaning.

Usage

```
boilerplate_find_chars(
  db,
  field,
  chars,
  exclude_entries = NULL,
  category = NULL
)
```

Arguments

db	List. The database to search.
field	Character. The field to check.
chars	Character vector. Characters to search for.
exclude_entries	Character vector. Entries to exclude from results.
category	Character. Category if db is unified.

Value

A named list of entries containing the specified characters.

Examples

```
# First create a sample database
unified_db <- list(
  measures = list(
    test1 = list(reference = "@Smith2023[p.45]"),
    test2 = list(reference = "Jones (2022)"),
    test3 = list(reference = "Brown[2021]")
  )
)

# Find all entries with @, [, or ] in references
entries_to_clean <- boilerplate_find_chars(
  db = unified_db,
  field = "reference",
  chars = c("@", "[", "]"),
  category = "measures"
)

# Find entries but exclude specific ones
entries_to_clean <- boilerplate_find_chars(
  db = unified_db,
  field = "reference",
  chars = c("@", "[", "]"),
  exclude_entries = c("forgiveness", "special_*"),
  category = "measures"
)
```

---

boilerplate_generate_measures

Generate Formatted Text for Measures

Description

This function generates formatted markdown text describing measures in a study. It creates a simple output with customisable heading levels, focusing on presenting measure information in a clean, consistent format.

Usage

```
boilerplate_generate_measures(
  variable_heading,
  variables,
  db,
  heading_level = 3,
  subheading_level = 4,
  print_waves = FALSE,
  print_keywords = FALSE,
  appendices_measures = NULL,
  label_mappings = NULL,
  quiet = FALSE,
  sample_items = FALSE,
  table_format = FALSE,
  check_completeness = FALSE,
  extract_scale_info = TRUE
)
```

Arguments

variable_heading	Character. Heading for the variable section (e.g., "Exposure Variable", "Outcome Variables").
variables	Character vector. Names of the variables to include.
db	List. Measures database. Can be either a measures database or a unified database. If a unified database is provided, the measures category will be extracted.
heading_level	Integer. Heading level for the section header (e.g., 2 for ##, 3 for ###). Default is 3.
subheading_level	Integer. Heading level for individual variables (e.g., 3 for ###, 4 for ####). Default is 4.
print_waves	Logical. Whether to include wave information in the output. Default is FALSE.
print_keywords	Logical. Whether to include keyword information in the output. Default is FALSE.
appendices_measures	Character. Optional reference to appendices containing measure details.
label_mappings	Named character vector. Mappings to transform variable names in the output.
quiet	Logical. If TRUE, suppresses all CLI alerts. Default is FALSE.
sample_items	Integer or FALSE. If numeric (1-3), shows only that many sample items. Default is FALSE (show all).

`table_format` Logical. If TRUE, formats output as markdown tables. Default is FALSE.

`check_completeness` Logical. If TRUE, adds notes about missing information. Default is FALSE.

`extract_scale_info` Logical. If TRUE, attempts to extract scale information from descriptions. Default is TRUE.

Value

Character string with formatted text describing the measures.

Examples

```
# Create a temporary directory and initialise databases
temp_dir <- tempdir()
data_path <- file.path(temp_dir, "boilerplate_measures_example", "data")

# Initialise measures database with default content
boilerplate_init(
  categories = "measures",
  data_path = data_path,
  create_dirs = TRUE,
  create_empty = FALSE,
  confirm = FALSE,
  quiet = TRUE
)

# Import the unified database
unified_db <- boilerplate_import(data_path = data_path, quiet = TRUE)
measures_db <- unified_db$measures

# Generate with sample items only
exposure_text <- boilerplate_generate_measures(
  variable_heading = "Exposure Variable",
  variables = "anxiety",
  db = measures_db,
  sample_items = 2,  # Show only first 2 items
  quiet = TRUE
)

# Check the output
cat(substr(exposure_text, 1, 150), "...\n")

# Generate with table format for multiple variables
outcome_text <- boilerplate_generate_measures(
  variable_heading = "Outcome Variables",
  variables = c("anxiety", "depression"),
  db = measures_db,
  table_format = TRUE,
  quiet = TRUE
)
```

```
# Clean up
unlink(file.path(temp_dir, "boilerplate_measures_example"), recursive = TRUE)
```

`boilerplate_generate_text` *Generate Text from boilerplate*

Description

This function generates text by retrieving and combining text from a boilerplate database. It allows for template variable substitution and customisation through overrides. Supports arbitrarily nested section paths using dot notation.

Usage

```
boilerplate_generate_text(
 category = c("measures", "methods", "results", "discussion", "appendix", "template"),
  sections,
  global_vars = list(),
  section_vars = list(),
  text_overrides = list(),
  db = NULL,
  data_path = NULL,
  warn_missing = TRUE,
  add_headings = FALSE,
  heading_level = "###",
  custom_headings = list(),
  quiet = FALSE,
  create_dirs = FALSE,
  confirm = TRUE,
  copy_bibliography = FALSE,
  bibliography_path = "."
)
```

Arguments

`category`	Character. Category of text to generate.
`sections`	Character vector. The sections to include (can use dot notation for nesting).
`global_vars`	List. Variables available to all sections.
`section_vars`	List. Section-specific variables.
`text_overrides`	List. Direct text overrides for specific sections.
`db`	List. Optional database to use. Can be either a category-specific database or a unified database. If a unified database is provided, the appropriate category will be extracted.

data_path	Character. Path to the directory where database files are stored. If NULL (default), uses tools::R_user_dir("boilerplate", "data").
warn_missing	Logical. Whether to warn about missing template variables.
add_headings	Logical. Whether to add markdown headings to sections. Default is FALSE.
heading_level	Character. The heading level to use (e.g., "###"). Default is "###".
custom_headings	List. Custom headings for specific sections. Names should match section names.
quiet	Logical. If TRUE, suppresses all CLI alerts. Default is FALSE.
create_dirs	Logical. If TRUE, creates directories that don't exist. Default is FALSE.
confirm	Logical. If TRUE, asks for confirmation before creating directories. Default is TRUE.
copy_bibliography	Logical. If TRUE, copies bibliography file to the project. Default is FALSE.
bibliography_path	Character. Directory to copy bibliography file to. Default is "." (current directory).

Value

Character. The combined text with optional headings.

Examples

```
# Create a temporary directory and initialise databases
temp_dir <- tempdir()
data_path <- file.path(temp_dir, "boilerplate_text_example", "data")

# Initialise with default content
boilerplate_init(
  categories = c("methods", "results"),
  data_path = data_path,
  create_dirs = TRUE,
  create_empty = FALSE,
  confirm = FALSE,
  quiet = TRUE
)

# Import the databases
unified_db <- boilerplate_import(data_path = data_path, quiet = TRUE)

# Basic usage with methods sections
methods_text <- boilerplate_generate_text(
  category = "methods",
  sections = c("sample"),
  global_vars = list(
    exposure_var = "political_conservative",
    population = "university students",
    timeframe = "2023-2024"
  ),
```

```
  db = unified_db,
  quiet = TRUE
)

# Check the output
cat(substr(methods_text, 1, 100), "...\n")

# Using with headings
methods_text <- boilerplate_generate_text(
  category = "methods",
  sections = c("sample"),
  global_vars = list(
    exposure_var = "treatment",
    timeframe = "2023-2024"
  ),
  db = unified_db,
  add_headings = TRUE,
  heading_level = "##",
  quiet = TRUE
)

# Clean up
unlink(file.path(temp_dir, "boilerplate_text_example"), recursive = TRUE)
```

`boilerplate_import` *Import boilerplate Database(s)*

Description

This function imports one or more boilerplate databases from disk. It automatically detects the file format (RDS or JSON) based on file extension. The function can import from a directory (original behaviour) or from a specific file path (new).

Usage

```
boilerplate_import(
  category = NULL,
  data_path = NULL,
  quiet = FALSE,
  project = "default"
)
```

Arguments

`category`	Character or character vector. Category of database to import. Options include "measures", "methods", "results", "discussion", "appendix", "template". If NULL (default), imports all available categories as a unified database. Ignored if data_path points to a specific file.

data_path	Character. Can be either: • A directory path containing database files (original behaviour) • A specific file path to import (e.g., "data/methods_db_20240115_143022.rds") If NULL (default), uses tools::R_user_dir("boilerplate", "data").
quiet	Logical. If TRUE, suppresses all CLI alerts. Default is FALSE.
project	Character. Project name for organizing databases. Default is "default". Projects are stored in separate subdirectories to allow multiple independent boilerplate collections.

Value

List. The imported database(s). If a single category was requested, returns that database. If multiple categories were requested, returns a unified database with each category as a named element.

Examples

```
# Create a temporary directory and initialise databases
temp_dir <- tempdir()
data_path <- file.path(temp_dir, "boilerplate_import_example", "data")

# Initialise some databases first
boilerplate_init(
  categories = c("methods", "measures"),
  data_path = data_path,
  create_dirs = TRUE,
  create_empty = FALSE,
  confirm = FALSE,
  quiet = TRUE
)

# Import all databases as unified (new default)
unified_db <- boilerplate_import(data_path = data_path, quiet = TRUE)
names(unified_db)

# Access specific category
methods_db <- unified_db$methods
names(methods_db)

# Import from a specific file (e.g., timestamped or backup)
# First, save with timestamp to create a timestamped file
boilerplate_save(
  methods_db,
  category = "methods",
  data_path = data_path,
  timestamp = TRUE,
  confirm = FALSE,
  quiet = TRUE
)

# List files to see what's available
list.files(data_path, pattern = "methods.*\\.rds")
```

```
# Import the timestamped file directly
timestamped_file <- list.files(
  data_path,
  pattern = "methods.*_\\d{8}_\\d{6}\\.rds",
  full.names = TRUE
)[1]
if (length(timestamped_file) > 0 && file.exists(timestamped_file)) {
  methods_timestamped <- boilerplate_import(
    data_path = timestamped_file,
    quiet = TRUE
  )
}

# Clean up
unlink(file.path(temp_dir, "boilerplate_import_example"), recursive = TRUE)
```

`boilerplate_init` *Initialise boilerplate Database*

Description

This function initialises a boilerplate database. By default, it creates a single unified JSON database containing all categories. Legacy support for separate RDS files is maintained through the unified parameter.

Usage

```
boilerplate_init(
 categories = c("measures", "methods", "results", "discussion", "appendix", "template"),
  merge_strategy = c("keep_existing", "merge_recursive", "overwrite_all"),
  data_path = NULL,
  quiet = FALSE,
  dry_run = FALSE,
  create_dirs = FALSE,
  confirm = TRUE,
  create_empty = TRUE,
  format = "json",
  project = "default"
)
```

Arguments

`categories`	Character vector. Categories to include in the database. Default is all categories: "measures", "methods", "results", "discussion", "appendix", "template".
`merge_strategy`	Character. How to merge with existing databases: "keep_existing", "merge_recursive", or "overwrite_all".

data_path	Character. Base path for data directory. If NULL (default), uses tools::R_user_dir("boilerplate", "data").
quiet	Logical. If TRUE, suppresses all CLI alerts. Default is FALSE.
dry_run	Logical. If TRUE, simulates the operation without writing files. Default is FALSE.
create_dirs	Logical. If TRUE, creates directories that don't exist. Default is FALSE.
confirm	Logical. If TRUE, asks for confirmation before making changes. Default is TRUE.
create_empty	Logical. If TRUE, creates empty database structures with just the template headings. Default is TRUE. Set to FALSE to use default content.
format	Character. Format to save: "json" (default), "rds", or "both".
project	Character. Project name for organizing databases. Default is "default". Projects are stored in separate subdirectories to allow multiple independent boilerplate collections.

Value

Invisibly returns TRUE if successful.

Examples

```
# Create a temporary directory for examples
temp_dir <- tempdir()
data_path <- file.path(temp_dir, "boilerplate_example", "data")

# Initialise unified JSON database (new default)
boilerplate_init(
  data_path = data_path,
  create_dirs = TRUE,
  create_empty = TRUE,
  confirm = FALSE,
  quiet = TRUE
)

# Check that unified JSON file was created
list.files(data_path)

# Initialise with default content in both formats
boilerplate_init(
  data_path = data_path,
  create_empty = FALSE,
  format = "both",
  confirm = FALSE,
  quiet = TRUE
)

# Clean up
unlink(file.path(temp_dir, "boilerplate_example"), recursive = TRUE)
```

`boilerplate_list_files` *List Available boilerplate Database Files*

Description

Lists all boilerplate database files in a directory, organised by type (standard, timestamped, or backup).

Usage

```
boilerplate_list_files(data_path = NULL, pattern = NULL, category = NULL)
```

Arguments

`data_path`	Character. Path to the directory containing database files. If NULL (default), uses tools::R_user_dir("boilerplate", "data").
`pattern`	Character. Optional regex pattern to filter files.
`category`	Character. Optional category to filter results. Options include "measures", "methods", "results", "discussion", "appendix", "template", or "unified".

Value

A list with class "boilerplate_files" containing:

`standard`	Data frame of standard database files
`timestamped`	Data frame of timestamped versions
`backups`	Data frame of backup files
`other`	Data frame of other files that match the pattern

Each data frame contains columns: file, path, size, modified, format, base_name, timestamp, type, and category.

See Also

`boilerplate_import` for importing the listed files, `boilerplate_restore_backup` for restoring from backups.

Examples

```
# Create temporary directory for example
temp_dir <- tempfile()
dir.create(temp_dir)

# Initialise with some content
boilerplate_init(data_path = temp_dir, categories = "methods",
                 create_dirs = TRUE, confirm = FALSE, quiet = TRUE)
```

```
# Save with timestamp to create some files
db <- boilerplate_import(data_path = temp_dir, quiet = TRUE)
boilerplate_save(db, data_path = temp_dir, timestamp = TRUE,
                 confirm = FALSE, quiet = TRUE)

# List all database files
files <- boilerplate_list_files(data_path = temp_dir)
print(files)

# List only methods files
files <- boilerplate_list_files(data_path = temp_dir, category = "methods")

# List files matching a pattern
files <- boilerplate_list_files(data_path = temp_dir, pattern = "202")

# Clean up
unlink(temp_dir, recursive = TRUE)
```

`boilerplate_list_projects`
List Available Projects

Description

Lists all available boilerplate projects in the specified data path.

Usage

```
boilerplate_list_projects(data_path = NULL, details = FALSE)
```

Arguments

`data_path`	Character. Base path for boilerplate data. If NULL, uses default location.
`details`	Logical. If TRUE, shows additional information about each project.

Value

Character vector of project names.

Examples

```
# Create temporary directory for example
temp_dir <- tempfile()
dir.create(temp_dir)

# Initialise some projects
```

```
boilerplate_init(data_path = temp_dir, project = "project1",
                 create_dirs = TRUE, confirm = FALSE, quiet = TRUE)
boilerplate_init(data_path = temp_dir, project = "project2",
                 create_dirs = TRUE, confirm = FALSE, quiet = TRUE)

# List all projects
projects <- boilerplate_list_projects(data_path = temp_dir)
print(projects)

# List with details
boilerplate_list_projects(data_path = temp_dir, details = TRUE)

# Clean up
unlink(temp_dir, recursive = TRUE)
```

`boilerplate_measures` *Access Measures from Unified Database*

Description

This function extracts and returns the measures portion of a unified database, optionally retrieving a specific measure by name.

Usage

```
boilerplate_measures(unified_db, name = NULL)
```

Arguments

`unified_db`	List. The unified boilerplate database
`name`	Character. Optional specific measure to retrieve

Value

List. The requested measures database or specific measure

Examples

```
# Create a temporary directory and initialise database
temp_dir <- tempdir()
data_path <- file.path(temp_dir, "boilerplate_measures_example", "data")

# Initialise with default measures
boilerplate_init(
  categories = "measures",
  data_path = data_path,
  create_dirs = TRUE,
```

```
  create_empty = FALSE,
  confirm = FALSE,
  quiet = TRUE
)

# Import all databases
unified_db <- boilerplate_import(data_path = data_path, quiet = TRUE)

# Get all measures
measures_db <- boilerplate_measures(unified_db)
names(measures_db)

# Get a specific measure
if ("anxiety" %in% names(measures_db)) {
  anxiety_measure <- boilerplate_measures(unified_db, "anxiety")
  anxiety_measure$description
}

# Clean up
unlink(file.path(temp_dir, "boilerplate_measures_example"), recursive = TRUE)
```

`boilerplate_measures_report` *Report on Measure Database Quality*

Description

Analyses a measures database and reports on completeness and consistency.

Usage

```
boilerplate_measures_report(db, measure_names = NULL, return_report = FALSE)
```

Arguments

`db`	List. Measures database to analyse.
`measure_names`	Character vector. Specific measures to analyse. If NULL, analyses all.
`return_report`	Logical. If TRUE, returns a data frame report. Default is FALSE.

Value

If return_report is TRUE, returns a data frame with quality metrics.

Examples

```
# Create temporary directory for example
temp_dir <- tempfile()
dir.create(temp_dir)

# Initialise and import
boilerplate_init(data_path = temp_dir, categories = "measures",
                 create_dirs = TRUE, confirm = FALSE, quiet = TRUE)
unified_db <- boilerplate_import(data_path = temp_dir, quiet = TRUE)

# Get a quality report
report <- boilerplate_measures_report(unified_db$measures, return_report = TRUE)

# Just print summary
boilerplate_measures_report(unified_db$measures)

# Clean up
unlink(temp_dir, recursive = TRUE)
```

---

boilerplate_methods *Access Methods from Unified Database*

Description

This function extracts and returns the methods portion of a unified database, optionally retrieving a specific method by name.

Usage

```
boilerplate_methods(unified_db, name = NULL)
```

Arguments

unified_db	List. The unified boilerplate database
name	Character. Optional specific method to retrieve using dot notation

Value

List or character. The requested methods database or specific method

Examples

```
# Create a temporary directory and initialise database
temp_dir <- tempdir()
data_path <- file.path(temp_dir, "boilerplate_methods_example", "data")

# Initialise with default methods content
```

```
boilerplate_init(
  categories = "methods",
  data_path = data_path,
  create_dirs = TRUE,
  create_empty = FALSE,
  confirm = FALSE,
  quiet = TRUE
)

# Import all databases
unified_db <- boilerplate_import(data_path = data_path, quiet = TRUE)

# Get all methods
methods_db <- boilerplate_methods(unified_db)
names(methods_db)

# Get a specific method using dot notation (if it exists)
if ("statistical" %in% names(methods_db) &&
    "longitudinal" %in% names(methods_db$statistical) &&
    "lmtp" %in% names(methods_db$statistical$longitudinal)) {
  lmtp_method <- boilerplate_methods(unified_db, "statistical.longitudinal.lmtp")
}

# Clean up
unlink(file.path(temp_dir, "boilerplate_methods_example"), recursive = TRUE)
```

`boilerplate_migrate_to_json` *Migration Utilities for RDS to JSON Conversion*

Description

Functions to help migrate from individual RDS files to unified JSON structure Migrate boilerplate Database from RDS to JSON

Usage

```
boilerplate_migrate_to_json(
  source_path,
  output_path,
  format = c("unified", "separate"),
  validate = TRUE,
  backup = TRUE,
  quiet = FALSE
)
```

Arguments

source_path	Path to directory containing RDS files or single RDS file
output_path	Path where JSON files should be saved
format	Output format: "unified" (single JSON) or "separate" (one per category)
validate	Logical. Validate against JSON schema after conversion? Default is TRUE.
backup	Logical. Create backup of original files? Default is TRUE.
quiet	Logical. Suppress progress messages? Default is FALSE.

Details

Comprehensive migration tool that converts existing RDS-based boilerplate databases to JSON format. Supports both unified (single file) and separate (multiple files) output formats with optional schema validation.

The migration process:

1. Scans source directory for RDS files
2. Creates timestamped backup if requested
3. Converts RDS to JSON format
4. Validates against schema if requested
5. Reports results and any issues

The "unified" format creates a single JSON file containing all categories, which is recommended for version control and collaborative workflows.

Value

List with migration results containing:

- `migrated`: Character vector of successfully migrated files
- `errors`: List of any errors encountered during migration
- `validation`: Validation results if validate=TRUE

See Also

boilerplate_rds_to_json, validate_json_database

Examples

```
# Create temporary directories for example
source_dir <- tempfile()
output_dir <- tempfile()
dir.create(source_dir)
dir.create(output_dir)

# Create sample RDS file
sample_db <- list(
  name = "Example Measure",
```

```
  description = "An example for migration",
  items = c("item1", "item2")
)
saveRDS(sample_db, file.path(source_dir, "measures_db.rds"))

# Migrate from RDS to JSON
results <- boilerplate_migrate_to_json(
  source_path = source_dir,
  output_path = output_dir,
  format = "unified",
  validate = FALSE
)

# Check results
if (length(results$errors) == 0) {
  message("Migration successful!")
} else {
  print(results$errors)
}

# Clean up
unlink(source_dir, recursive = TRUE)
unlink(output_dir, recursive = TRUE)
```

`boilerplate_path_ops` *Path-Based Database Operations*

Description

These functions manipulate databases using dot-separated paths, preserving nesting.

Usage

```
boilerplate_add_entry(db, path, value, auto_sort = TRUE)

boilerplate_update_entry(db, path, value, auto_sort = TRUE)

boilerplate_remove_entry(db, path, auto_sort = TRUE)

boilerplate_get_entry(db, path)

boilerplate_sort_db(db)

boilerplate_path_exists(db, path)

boilerplate_list_paths(db, prefix = "")
```

Arguments

db	List. The database to examine.
path	Character. Dot-separated path.
value	Any. The value to set.
auto_sort	Logical. Whether to auto-sort at each level. Default TRUE.
prefix	Character. Optional prefix for path construction.

Value

Modified database (list).

Modified database (list).

Modified database (list).

The value at the specified path.

The sorted database.

Logical. TRUE if the path exists, FALSE otherwise.

Character vector of all available paths.

---

boilerplate_rds_to_json

Convert RDS Database to JSON

Description

Utility function to convert existing RDS databases to JSON format. Can convert a single file or batch convert all RDS files in a directory.

Usage

```
boilerplate_rds_to_json(
  input_path,
  output_path = NULL,
  pretty = TRUE,
  quiet = FALSE
)
```

Arguments

input_path	Path to RDS file or directory containing RDS files
output_path	Path to save JSON files. If NULL, saves in same location as input with .json extension.
pretty	Pretty print JSON for readability? Default is TRUE.
quiet	Suppress messages? Default is FALSE.

Details

This function is useful for migrating existing RDS-based databases to the more portable JSON format. JSON files are human-readable, work well with version control, and can be used across different programming languages.

Value

Invisible TRUE on success

See Also

boilerplate_migrate_to_json for full migration workflow

Examples

```
# Create temporary directory for example
temp_dir <- tempfile()
dir.create(temp_dir)

# Create sample RDS file
sample_db <- list(methods = list(sampling = "Random sampling"))
saveRDS(sample_db, file.path(temp_dir, "methods_db.rds"))

# Convert single file
boilerplate_rds_to_json(file.path(temp_dir, "methods_db.rds"))

# Convert all RDS files in directory
boilerplate_rds_to_json(temp_dir)

# Convert to different location
output_dir <- tempfile()
dir.create(output_dir)
boilerplate_rds_to_json(temp_dir, output_path = output_dir)

# Clean up
unlink(temp_dir, recursive = TRUE)
unlink(output_dir, recursive = TRUE)
```

---

boilerplate_restore_backup

Restore Database from Backup

Description

Convenience function to restore a database from backup files. Can either import the backup for inspection or restore it as the current database.

Usage

```
boilerplate_restore_backup(
  category = NULL,
  backup_version = NULL,
  data_path = NULL,
  restore = FALSE,
  confirm = TRUE,
  quiet = FALSE
)
```

Arguments

category	Character. Category to restore, or NULL for unified database.
backup_version	Character. Specific backup timestamp (format: "YYYYMMDD_HHMMSS"), or NULL to use the latest backup.
data_path	Character. Path to directory containing backup files. If NULL (default), uses tools::R_user_dir("boilerplate", "data").
restore	Logical. If TRUE, saves the backup as the current standard file (overwrites existing). If FALSE (default), just returns the backup content.
confirm	Logical. If TRUE (default), asks for confirmation before overwriting. Ignored if restore = FALSE.
quiet	Logical. If TRUE, suppresses messages. Default is FALSE.

Value

The restored database (invisibly if restore = TRUE).

See Also

boilerplate_list_files to see available backups, boilerplate_import for general import functionality.

Examples

```
# Create temporary directory for example
temp_dir <- tempfile()
dir.create(temp_dir)

# Initialise and create some content
boilerplate_init(data_path = temp_dir, categories = "methods",
                 create_dirs = TRUE, confirm = FALSE, quiet = TRUE)
db <- boilerplate_import(data_path = temp_dir, quiet = TRUE)

# Create a backup by saving with timestamp
boilerplate_save(db, data_path = temp_dir, timestamp = TRUE,
                 confirm = FALSE, quiet = TRUE)

# List available files to see backup
files <- boilerplate_list_files(data_path = temp_dir)
```

```
# Create a proper backup by using create_backup parameter
boilerplate_save(db, data_path = temp_dir, create_backup = TRUE,
                 confirm = FALSE, quiet = TRUE)

# Now list files again to see the backup
files <- boilerplate_list_files(data_path = temp_dir)

# View latest backup without restoring
if (nrow(files$backups) > 0) {
  backup_db <- boilerplate_restore_backup(data_path = temp_dir)
}

# Clean up
unlink(temp_dir, recursive = TRUE)
```

`boilerplate_results` *Access Results from Unified Database*

Description

This function extracts and returns the results portion of a unified database, optionally retrieving a specific result section by name using dot notation.

Usage

```
boilerplate_results(unified_db, name = NULL)
```

Arguments

`unified_db`	List. The unified boilerplate database
`name`	Character. Optional specific result section to retrieve (supports dot notation)

Value

List or character. The requested results database or specific section

Examples

```
# Create a temporary directory and initialise database
temp_dir <- tempdir()
data_path <- file.path(temp_dir, "boilerplate_results_example", "data")

# Initialise with default results
boilerplate_init(
  categories = "results",
  data_path = data_path,
```

```
  create_dirs = TRUE,
  create_empty = FALSE,
  confirm = FALSE,
  quiet = TRUE
)

# Import all databases
unified_db <- boilerplate_import(data_path = data_path, quiet = TRUE)

# Get all results sections
results_db <- boilerplate_results(unified_db)
names(results_db)

# Clean up
unlink(file.path(temp_dir, "boilerplate_results_example"), recursive = TRUE)
```

`boilerplate_save` *Save boilerplate Database*

Description

This function saves a boilerplate database to disk in RDS, JSON, or both formats.

Usage

```
boilerplate_save(
  db,
  category = NULL,
  data_path = NULL,
  format = "json",
  confirm = TRUE,
  create_dirs = FALSE,
  quiet = FALSE,
  pretty = TRUE,
  timestamp = FALSE,
  entry_level_confirm = TRUE,
  create_backup = NULL,
  select_elements = NULL,
  output_file = NULL,
  project = "default"
)
```

Arguments

db	List. The database to save. Can be a single category database or unified database.
category	Character. The category name if saving a single category. If NULL and db contains multiple categories, saves as unified database.

data_path	Character. Base path for data directory. If NULL (default), uses tools::R_user_dir("boilerplate", "data").
format	Character. Format to save: "json" (default), "rds", or "both".
confirm	Logical. If TRUE, asks for confirmation. Default is TRUE.
create_dirs	Logical. If TRUE, creates directories if they don't exist. Default is FALSE.
quiet	Logical. If TRUE, suppresses all CLI alerts. Default is FALSE.
pretty	Logical. If TRUE (default), pretty-print JSON for readability.
timestamp	Logical. If TRUE, add timestamp to filename. Default is FALSE.
entry_level_confirm	Logical. Not used, kept for backward compatibility.
create_backup	Logical. If TRUE, creates a backup before saving. Default is TRUE unless running examples or in a temporary directory.
select_elements	Character vector. Not used, kept for backward compatibility.
output_file	Character. Not used, kept for backward compatibility.
project	Character. Project name for organizing databases. Default is "default". Projects are stored in separate subdirectories to allow multiple independent boilerplate collections.

Value

Invisible TRUE if successful, with saved file paths as an attribute.

Examples

```
# Create a temporary directory
temp_dir <- tempdir()
data_path <- file.path(temp_dir, "boilerplate_save_example", "data")

# Create a test database
test_db <- list(
  methods = list(
    sample = list(default = "Sample text")
  ),
  measures = list(
    test_measure = list(name = "Test", description = "A test measure")
  )
)

# Save as unified database
boilerplate_save(
  db = test_db,
  data_path = data_path,
  create_dirs = TRUE,
  confirm = FALSE,
  quiet = TRUE
)
```

```
# Check that file was created
file.exists(file.path(data_path, "boilerplate_unified.rds"))

# Save a single category
boilerplate_save(
  db = test_db$methods,
  category = "methods",
  data_path = data_path,
  confirm = FALSE,
  quiet = TRUE
)

# Clean up
unlink(file.path(temp_dir, "boilerplate_save_example"), recursive = TRUE)
```

`boilerplate_standardise_measures`
Standardise Measure Entries in Database

Description

This function standardises measure entries by extracting scale information, identifying reversed items, cleaning descriptions, and ensuring consistent structure.

Usage

```
boilerplate_standardise_measures(
  db,
  measure_names = NULL,
  extract_scale = TRUE,
  identify_reversed = TRUE,
  clean_descriptions = TRUE,
  ensure_structure = TRUE,
  standardise_references = TRUE,
  json_compatible = FALSE,
  verbose = FALSE,
  quiet = FALSE
)
```

Arguments

`db`	List. Either a single measure or a measures database to standardise.
`measure_names`	Character vector. Specific measures to standardise. If NULL, processes all.
`extract_scale`	Logical. Extract scale information from descriptions. Default is TRUE.
`identify_reversed`	Logical. Identify reversed items. Default is TRUE.

`clean_descriptions`
: Logical. Clean up description text. Default is TRUE.

`ensure_structure`
: Logical. Ensure all measures have standard fields. Default is TRUE.

`standardise_references`
: Logical. Standardise reference format. Default is TRUE.

`json_compatible`
: Logical. Ensure JSON-compatible structure by converting vectors to lists and removing NULL values. Default is FALSE.

`verbose`
: Logical. Show detailed progress information. Default is FALSE.

`quiet`
: Logical. Suppress all messages. Default is FALSE.

Value

List. The standardised measure(s) or database.

Examples

```
# Create a temporary directory and initialise database
temp_dir <- tempdir()
data_path <- file.path(temp_dir, "boilerplate_standardise_example", "data")

# Initialise database
boilerplate_init(
  categories = "measures",
  data_path = data_path,
  create_dirs = TRUE,
  create_empty = FALSE,
  confirm = FALSE,
  quiet = TRUE
)

# Import database
unified_db <- boilerplate_import(data_path = data_path, quiet = TRUE)

# Standardise all measures in database
unified_db$measures <- boilerplate_standardise_measures(
  unified_db$measures,
  quiet = TRUE
)

# Standardise with JSON compatibility
unified_db$measures <- boilerplate_standardise_measures(
  unified_db$measures,
  json_compatible = TRUE,
  quiet = TRUE
)

# Check that standardisation worked
names(unified_db$measures$anxiety)
```

```
# Standardise specific measures only
unified_db$measures <- boilerplate_standardise_measures(
  unified_db$measures,
  measure_names = c("anxiety", "depression"),
  quiet = TRUE
)

# Clean up
unlink(file.path(temp_dir, "boilerplate_standardise_example"), recursive = TRUE)
```

`boilerplate_template` *Access Template Content*

Description

This function extracts and returns the template portion of a unified database, optionally retrieving a specific template by name.

Usage

```
boilerplate_template(unified_db, name = NULL)
```

Arguments

`unified_db`	List. The unified boilerplate database containing template content
`name`	Character. Optional specific template to retrieve by name

Details

Access Templates from Unified Database

Value

List or character. The requested template database or specific template. If name is NULL, returns the entire template database. If name is specified, returns the template with that name.

Examples

```
# Create a temporary directory and initialise database
temp_dir <- tempdir()
data_path <- file.path(temp_dir, "boilerplate_template_example", "data")

# Initialise with default template content
boilerplate_init(
  categories = "template",
  data_path = data_path,
  create_dirs = TRUE,
  create_empty = FALSE,
```

```
  confirm = FALSE,
  quiet = TRUE
)

# Import all databases
unified_db <- boilerplate_import(data_path = data_path, quiet = TRUE)

# Get all templates
template_db <- boilerplate_template(unified_db)
names(template_db)

# Get a specific template (if it exists)
if ("template" %in% names(unified_db) && length(unified_db$template) > 0) {
  template_names <- names(unified_db$template)
  if (length(template_names) > 0) {
    first_template <- boilerplate_template(unified_db, template_names[1])
  }
}

# Clean up
unlink(file.path(temp_dir, "boilerplate_template_example"), recursive = TRUE)
```

`boilerplate_update_bibliography` *Update Bibliography from Remote Source*

Description

Downloads and caches a bibliography file from a remote URL specified in the database.

Usage

```
boilerplate_update_bibliography(
  db,
  cache_dir = NULL,
  force = FALSE,
  quiet = FALSE
)
```

Arguments

db	Database object containing bibliography information
cache_dir	Directory to cache the bibliography file. Default uses tools::R_user_dir("boilerplate", "cache")
force	Logical. Force re-download even if cached file exists
quiet	Logical. Suppress messages

Value

Path to the local bibliography file, or NULL if no bibliography specified

Examples

```
# Create temporary directory for example
temp_dir <- tempfile()
dir.create(temp_dir)

# Initialise and import
boilerplate_init(data_path = temp_dir, create_dirs = TRUE, confirm = FALSE, quiet = TRUE)
db <- boilerplate_import(data_path = temp_dir, quiet = TRUE)

# Update bibliography
bib_file <- boilerplate_update_bibliography(db)

# Clean up
unlink(temp_dir, recursive = TRUE)
```

---

boilerplate_validate_references

Validate References in boilerplate Database

Description

Checks that all citations in the boilerplate text exist in the bibliography file.

Usage

```
boilerplate_validate_references(
  db,
  bib_file = NULL,
  categories = c("methods", "results", "discussion", "appendix"),
  quiet = FALSE
)
```

Arguments

db	Database object to validate
bib_file	Path to bibliography file. If NULL, will try to download from database
categories	Character vector of categories to check. Default is all text categories.
quiet	Logical. Suppress detailed messages

Value

List with validation results including used keys, available keys, and missing keys

Examples

```
# Create temporary directory for example
temp_dir <- tempfile()
dir.create(temp_dir)

# Initialise and import
boilerplate_init(data_path = temp_dir, create_dirs = TRUE, confirm = FALSE, quiet = TRUE)
db <- boilerplate_import(data_path = temp_dir, quiet = TRUE)

# Validate references
validation <- boilerplate_validate_references(db)
if (length(validation$missing) > 0) {
  warning("Missing references: ", paste(validation$missing, collapse = ", "))
}

# Clean up
unlink(temp_dir, recursive = TRUE)
```

---

compare_rds_json *Compare RDS and JSON Databases*

Description

Compares RDS and JSON database files to verify migration accuracy and identify any differences in structure or content.

Usage

```
compare_rds_json(rds_path, json_path, ignore_meta = TRUE)
```

Arguments

rds_path	Path to RDS file
json_path	Path to JSON file
ignore_meta	Logical. Ignore metadata fields (those starting with underscore)? Default is TRUE.

Details

This function performs a deep comparison of database structures, checking:

- Data types match between formats
- All keys/fields are present in both versions
- Values are equivalent (accounting for format differences)

Useful for verifying that migration from RDS to JSON preserves all data and structure correctly.

Value

List of differences, each containing:

- path: Location of difference in database structure
- type: Type of difference (type_mismatch, missing_in_json, missing_in_rds, value_mismatch)
- Additional fields depending on difference type

See Also

boilerplate_migrate_to_json, boilerplate_rds_to_json

Examples

```
# Create temporary directories for example
temp_dir <- tempfile()
dir.create(temp_dir)

# Create sample RDS database
sample_db <- list(
  methods = list(
    sampling = "Random sampling",
    analysis = "Regression analysis"
  )
)
rds_path <- file.path(temp_dir, "methods_db.rds")
saveRDS(sample_db, rds_path)

# Convert to JSON
json_path <- file.path(temp_dir, "methods_db.json")
boilerplate_rds_to_json(rds_path, quiet = TRUE)

# Compare original and migrated databases
differences <- compare_rds_json(rds_path, json_path)

if (length(differences) == 0) {
  message("Migration successful - databases are equivalent!")
} else {
  # Review differences
  for (diff in differences) {
    cat(sprintf("Difference at %s: %s\n", diff$path, diff$type))
  }
}

# Clean up
unlink(temp_dir, recursive = TRUE)
```

print.boilerplate_files *Print Method for boilerplate_files*

Description

Print Method for boilerplate_files

Usage

```
## S3 method for class 'boilerplate_files'
print(x, ...)
```

Arguments

x	A boilerplate_files object from boilerplate_list_files
...	Additional arguments (ignored)

Value

Invisible NULL. Information is printed to the console.

validate_json_database *Validate JSON Database Against Schema*

Description

Validates a JSON database file against the appropriate JSON schema to ensure data integrity and structure compliance.

Usage

```
validate_json_database(json_file, type = "unified")
```

Arguments

json_file	Path to JSON file to validate
type	Database type ("methods", "measures", "results", "discussion", "appendix", "template", "unified"). Default is "unified".

Details

Uses JSON Schema Draft 7 for validation via the jsonvalidate package. Schemas define required fields, data types, and structure for each database type. This ensures consistency across different database files and helps catch errors early.

If jsonvalidate is not installed, validation is skipped with a message.

Value

Character vector of validation errors (empty if valid)

See Also

boilerplate_migrate_to_json

Examples

```
# Create temporary directory for example
temp_dir <- tempfile()
dir.create(temp_dir)

# Create sample JSON database
sample_db <- list(
  measures = list(
    anxiety = list(name = "Anxiety Scale", items = 10)
  )
)
json_path <- file.path(temp_dir, "measures_db.json")
if (requireNamespace("jsonlite", quietly = TRUE)) {
  jsonlite::write_json(sample_db, json_path)

  # Validate a measures database
  errors <- validate_json_database(json_path, "measures")
  if (length(errors) == 0) {
    message("Database is valid!")
  } else {
    cat("Validation errors:\n", paste(errors, collapse = "\n"))
  }
}

# Clean up
unlink(temp_dir, recursive = TRUE)
```

Index