Package 'gooseR'

December 22, 2025

Title R Integration for 'Goose' AI

Version 0.1.1

Description Seamless integration between R and 'Goose' AI capabilities including memory management, visualization enhancements, and workflow automation. Save R objects to 'Goose' memory, apply Block branding to visualizations, and manage data science project workflows. For more information about 'Goose' AI, see <https://github.com/block/goose>.

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Suggests rmarkdown, testthat (>= 3.0.0), dplyr, markdown, stringr, htmltools, DT, RSQLite, clipr, janitor

Config/testthat/edition 3

Imports cli, fs, here, jsonlite, magrittr, purrr, ggplot2, yaml, grDevices, systemfonts, processx, DBI, digest, future, future.apply, promises, glue, R6, rappdirs, later, shiny, miniUI, rstudioapi, knitr

VignetteBuilder knitr

NeedsCompilation no

Author Brandon Theriault [aut, cre]

Maintainer Brandon Theriault <btheriault@block.xyz>

Repository CRAN

Date/Publication 2025-12-22 10:40:09 UTC

Contents

brand_css 4
brand_palette 5
brand_rmd_template 5
format_ai_response 6
format_conversation 7

gooseR-branding . . . 7
goose_addins . . . 7
goose_addin_chat . . . 7
goose_addin_quick . . . 8
goose_addin_review . . . 8
goose_addin_snippet . . . 8
goose_addin_template . . . 8
goose_ask . . . 9
goose_ask_raw . . . 10
goose_async . . . 11
goose_async_retry . . . 11
goose_async_timeout . . . 12
goose_backup . . . 12
goose_batch . . . 13
goose_batch_file . . . 14
goose_cache . . . 14
goose_cached . . . 15
goose_cache_clear . . . 15
goose_cache_export . . . 16
goose_cache_get . . . 16
goose_cache_import . . . 17
goose_cache_init . . . 17
goose_cache_set . . . 18
goose_cache_stats . . . 18
goose_cache_ui . . . 19
goose_cache_warmup . . . 19
goose_check_installation . . . 20
goose_clean_text . . . 20
goose_clear_all . . . 21
goose_clear_category . . . 21
goose_clear_tags . . . 22
goose_configure . . . 23
goose_continuation_prompt . . . 24
goose_conversation_ui . . . 25
goose_create_brand . . . 25
goose_create_brand_ai . . . 26
goose_create_quarto . . . 27
goose_create_report . . . 27
goose_create_template . . . 28
goose_delete . . . 29
goose_divider . . . 29
goose_document . . . 30
goose_exists . . . 31
goose_explain_error . . . 31
goose_format_options . . . 32
goose_generate_tests . . . 33
goose_get_config . . . 33
goose_give_sample . . . 34

goose_honk . . . 35
goose_insert_chunk . . . 36
goose_list . . . 36
goose_load . . . 37
goose_loop_me . . . 37
goose_make_a_plan . . . 38
goose_map . . . 39
goose_mapreduce . . . 40
goose_optimize_palette . . . 40
goose_optimize_plot . . . 41
goose_pipeline . . . 42
goose_quarto . . . 42
goose_quarto_chunk . . . 43
goose_query . . . 44
goose_recipe . . . 44
goose_reduce . . . 45
goose_rename . . . 45
goose_rename_columns . . . 46
goose_review_code . . . 47
goose_rmd_ai . . . 48
goose_save . . . 49
goose_session . . . 50
goose_session_clear . . . 50
goose_session_end . . . 51
goose_session_list . . . 51
goose_session_save . . . 52
goose_session_start . . . 52
goose_stream . . . 53
goose_streaming . . . 54
goose_stream_async . . . 54
goose_stream_multi . . . 55
goose_stream_session . . . 55
goose_suggest_colors . . . 56
goose_template . . . 57
goose_template_apply . . . 58
goose_template_builtin . . . 58
goose_template_from_query . . . 59
goose_template_list . . . 59
goose_template_load . . . 60
goose_template_save . . . 60
goose_template_validate . . . 61
goose_test_cli . . . 61
goose_ui_components . . . 62
goose_version . . . 62
goose_view_column_map . . . 63
goose_worker_pool . . . 63
load_brand . . . 64
preview_brand . . . 64

print.goose_cache_stats 65
print.goose_code_review 65
print.goose_error_explanation 66
print.goose_palette 66
print.goose_response 67
print.goose_session 67
print.goose_template 68
register_goose_engine 68
StreamHandler 68
StreamSession 70
theme_brand 71
validate_brand 72
with_goose_session 72
WorkerPool 73

Index **75**

`brand_css` *Generate CSS from brand configuration*

Description

Generate CSS from brand configuration

Usage

```
brand_css(brand = "block", output_file = NULL, minify = FALSE)
```

Arguments

`brand`	Name of the brand
`output_file`	Path to save CSS file (NULL to return as string)
`minify`	Minify the CSS output (default: FALSE)

Value

CSS content as string (invisibly if saved to file)

Examples

```
## Not run:
# Save to file in temp directory
temp_file <- file.path(tempdir(), "block.css")
brand_css("block", temp_file)

# Get as string
css <- brand_css("block")

## End(Not run)
```

brand_palette *Get brand color palette*

Description

Get brand color palette

Usage

```
brand_palette(brand = "block", palette = "categorical", n = NULL)
```

Arguments

brand	Name of the brand
palette	Type of palette ("categorical", "sequential", "diverging")
n	Number of colors to return (NULL for all)

Value

Character vector of hex colors

Examples

```
## Not run:
colors <- brand_palette("block", "categorical")

## End(Not run)
```

brand_rmd_template *Generate RMarkdown template with brand styling*

Description

Generate RMarkdown template with brand styling

Usage

```
brand_rmd_template(
  brand = "block",
  title = "Report",
  output_format = "html_document",
  output_file = NULL
)
```

Arguments

brand	Name of the brand
title	Document title
output_format	RMarkdown output format (default: "html_document")
output_file	Path to save template (NULL to return as string)

Value

RMarkdown template content

Examples

```
## Not run:
# Create branded RMarkdown template in temp directory
temp_file <- file.path(tempdir(), "report.Rmd")
brand_rmd_template("block", "My Report", output_file = temp_file)

## End(Not run)
```

---

format_ai_response *Format AI response text for better display in R console*

Description

Format AI response text for better display in R console

Usage

```
format_ai_response(
  text,
  width = 80,
  indent = 0,
  preserve_code = TRUE,
  color = TRUE
)
```

Arguments

text	Character string with AI response
width	Integer, line width for wrapping (default 80)
indent	Integer, spaces to indent wrapped lines (default 0)
preserve_code	Logical, whether to preserve code blocks (default TRUE)
color	Logical, whether to use color output (default TRUE)

Value

Formatted text (invisibly), printed to console

format_conversation *Format and display a gooseR conversation*

Description

Format and display a gooseR conversation

Usage

```
format_conversation(messages, show_dividers = TRUE)
```

Arguments

messages	List of messages (question/response pairs)
show_dividers	Logical, show dividers between messages

gooseR-branding *GooseR Universal Branding System*

Description

Functions for creating and applying consistent branding across R outputs

goose_addins *RStudio/Positron Addins for GooseR*

Description

Interactive UI components for IDE integration

goose_addin_chat *GooseR Chat Interface*

Description

Interactive chat interface for AI conversations

Usage

```
goose_addin_chat()
```

goose_addin_quick *Quick Ask*

Description

Quick question to Goose (non-interactive)

Usage

```
goose_addin_quick()
```

goose_addin_review *Code Review Interface*

Description

Review selected code with AI

Usage

```
goose_addin_review()
```

goose_addin_snippet *Code Snippet Generator*

Description

Generate and insert code snippets

Usage

```
goose_addin_snippet()
```

goose_addin_template *Template Builder UI*

Description

Visual interface for creating prompt templates

Usage

```
goose_addin_template()
```

goose_ask *Enhanced Ask Goose with Formatting*

Description

Send a query to Goose AI and get a beautifully formatted response. This is an enhanced version of goose_ask that formats the output by default.

Usage

```
goose_ask(
  prompt,
  format = getOption("goose.auto_format", TRUE),
  output_format = c("text", "json"),
  quiet = TRUE,
  timeout = 30,
  session_id = NULL,
  width = getOption("goose.format_width", 80),
  color = getOption("goose.format_color", TRUE),
  ...
)
```

Arguments

prompt	Character string with the question or prompt
format	Logical, whether to format the response (default TRUE)
output_format	Character, either "text" or "json"
quiet	Logical, suppress status messages
timeout	Numeric, timeout in seconds (default 30)
session_id	Optional session ID for context preservation
width	Integer, line width for wrapping (default 80)
color	Logical, whether to use color output (default TRUE)
...	Additional formatting arguments

Value

If format=TRUE and output_format="text", displays formatted response and returns raw text invisibly. If format=FALSE or output_format="json", returns the raw response.

Examples

```
## Not run:
# Get a beautifully formatted response (default)
goose_ask("What is the tidyverse?")
```

```
# Get raw unformatted response
raw <- goose_ask("What is R?", format = FALSE)

# Get JSON response (never formatted)
data <- goose_ask("List 5 R packages", output_format = "json")

# Customize formatting
goose_ask("Explain ggplot2", width = 100, color = FALSE)

## End(Not run)
```

`goose_ask_raw` *Original goose_ask (unformatted)*

Description

This is the original goose_ask function without formatting. Use this if you need the raw behavior.

Usage

```
goose_ask_raw(
  prompt,
  output_format = c("text", "json"),
  quiet = TRUE,
  timeout = 30,
  session_id = NULL
)
```

Arguments

`prompt`	Character string with the question or prompt
`output_format`	Character, either "text" or "json"
`quiet`	Logical, suppress status messages
`timeout`	Numeric, timeout in seconds (default 30)
`session_id`	Optional session ID for context preservation

Value

Character string with response (text format) or list (json format)

goose_async *Async and Parallel Execution Module for GooseR*

Description

Provides asynchronous and parallel query execution

Run a query in the background and return a promise

Usage

```
goose_async(query, session_id = NULL)
```

Arguments

query	The query to execute
session_id	Optional session ID

Value

A promise that resolves to the response

Examples

```
## Not run:
library(promises)

# Single async query
goose_async("Explain async programming") %...>%
  { cat("Response:", .) }

# Chain multiple async operations
goose_async("Write a function") %...>%
  { goose_async(paste("Optimize this:", .)) } %...>%
  { cat("Optimized:", .) }

## End(Not run)
```

goose_async_retry *Async Query with Retry*

Description

Execute query with automatic retry on failure

Usage

```
goose_async_retry(query, max_retries = 3, retry_delay = 2)
```

Arguments

query	The query to execute
max_retries	Maximum number of retries
retry_delay	Delay between retries in seconds

Value

Response or error

---

goose_async_timeout *Async Query with Timeout*

Description

Execute query with timeout protection

Usage

```
goose_async_timeout(query, timeout = 30)
```

Arguments

query	The query to execute
timeout	Timeout in seconds

Value

Response or timeout error

---

goose_backup *Create a backup of all gooseR memory*

Description

Create a backup of all gooseR memory

Usage

```
goose_backup(backup_dir = "goose_backup", timestamp = TRUE)
```

Arguments

backup_dir	Character, directory to save backup
timestamp	Logical, whether to add timestamp to backup files

Value

Invisible integer count of backed up items

Examples

```
## Not run:
# Create backup with timestamp
goose_backup()

# Create backup in specific directory
goose_backup("my_backups")

## End(Not run)
```

---

goose_batch *Execute Multiple Queries in Parallel*

Description

Run multiple queries simultaneously

Usage

```
goose_batch(queries, max_workers = 4, progress = TRUE, cache = TRUE)
```

Arguments

queries	Vector or list of queries
max_workers	Maximum number of parallel workers
progress	Show progress bar
cache	Use caching for responses

Value

List of responses

Examples

```
## Not run:
queries <- c(
  "Explain R functions",
  "Write a data analysis script",
  "Create a visualization"
)

results <- goose_batch(queries, max_workers = 3)

## End(Not run)
```

goose_batch_file *Batch Process File*

Description

Process a file of queries in parallel

Usage

```
goose_batch_file(file, output_file = NULL, max_workers = 4)
```

Arguments

file	Path to file with queries (one per line)
output_file	Optional output file for results
max_workers	Maximum parallel workers

Value

List of results

goose_cache *Response Caching Module for GooseR*

Description

Provides intelligent caching of AI responses for performance

See Also

Requires Suggests: RSQLite. Functions in this module check for RSQLite availability at runtime.

goose_cached *Cached Query Execution*

Description

Execute query with automatic caching

Usage

```
goose_cached(query, use_cache = TRUE, max_age = 86400, force_refresh = FALSE)
```

Arguments

query	The query to execute
use_cache	Whether to use cache
max_age	Maximum cache age in seconds
force_refresh	Force new execution

Value

Query response

goose_cache_clear *Clear Cache*

Description

Clear cache entries based on criteria

Usage

```
goose_cache_clear(older_than = NULL, pattern = NULL, all = FALSE, conn = NULL)
```

Arguments

older_than	Clear entries older than this (seconds)
pattern	Clear entries matching this pattern
all	Clear all entries
conn	Database connection

Value

Number of entries cleared

goose_cache_export *Export Cache*

Description

Export cache to file for backup or sharing

Usage

```
goose_cache_export(file, format = "json", conn = NULL)
```

Arguments

file	Path to export file
format	Export format (json, csv, rds)
conn	Database connection

Value

Number of entries exported

goose_cache_get *Get Cached Response*

Description

Retrieve a cached response for a query

Usage

```
goose_cache_get(query, model = NULL, max_age = NULL, conn = NULL)
```

Arguments

query	The query text
model	Optional model identifier
max_age	Maximum age in seconds (NULL for no limit)
conn	Database connection

Value

Response text or NULL if not found

goose_cache_import *Import Cache*

Description

Import cache from file

Usage

```
goose_cache_import(file, format = "json", conn = NULL)
```

Arguments

file	Path to import file
format	Import format (json, csv, rds)
conn	Database connection

Value

Number of entries imported

goose_cache_init *Initialize Cache Database*

Description

Create or connect to the cache database

Usage

```
goose_cache_init(cache_dir = NULL)
```

Arguments

cache_dir	Directory for cache database (default: tempdir() for CRAN compliance)

Value

DBI connection object

Examples

```
## Not run:
# Initialize cache in temp directory (CRAN compliant)
conn <- goose_cache_init()

# Or specify custom directory
conn <- goose_cache_init(cache_dir = tempdir())

## End(Not run)
```

`goose_cache_set` *Cache an AI Response*

Description

Store a query-response pair in the cache

Usage

```
goose_cache_set(query, response, model = NULL, metadata = NULL, conn = NULL)
```

Arguments

`query`	The query text
`response`	The response text
`model`	Optional model identifier
`metadata`	Optional metadata as list
`conn`	Database connection (auto-created if NULL)

Value

Logical indicating success

`goose_cache_stats` *Get Cache Statistics*

Description

Get statistics about cache usage

Usage

```
goose_cache_stats(conn = NULL)
```

Arguments

conn	Database connection

Value

List of cache statistics

---

goose_cache_ui *Cache Browser UI*

Description

Visual interface for browsing and managing cache

Usage

```
goose_cache_ui()
```

---

goose_cache_warmup *Cache Warmup*

Description

Pre-populate cache with common queries

Usage

```
goose_cache_warmup(queries, parallel = FALSE)
```

Arguments

queries	Vector of queries to cache
parallel	Execute in parallel

Value

Number of queries cached

goose_check_installation *Check Goose CLI Installation*

Description

Check Goose CLI Installation

Usage

```
goose_check_installation()
```

Value

Logical, TRUE if Goose CLI is installed

goose_clean_text *Get copy-friendly version of formatted text*

Description

Removes formatting artifacts and returns clean text for copying

Usage

```
goose_clean_text(text, preserve_markdown = TRUE)
```

Arguments

text	Character string with formatted text
preserve_markdown	Logical, keep markdown formatting (default TRUE)

Value

Clean text suitable for copying

Examples

```
## Not run:
response <- goose_ask("What is R?")
# Copy-friendly version:
clean <- goose_clean_text(response)
cat(clean)

## End(Not run)
```

goose_clear_all *Clear all gooseR memory*

Description

Clear all gooseR memory

Usage

```
goose_clear_all(
  confirm = TRUE,
  backup_first = FALSE,
  backup_dir = "goose_backup"
)
```

Arguments

confirm	Logical, whether to ask for confirmation (default TRUE)
backup_first	Logical, whether to create backup before clearing
backup_dir	Character, directory for backup if backup_first is TRUE

Value

Invisible integer count of deleted items

Examples

```
## Not run:
# Clear all memory with confirmation
goose_clear_all()

# Clear with backup
goose_clear_all(backup_first = TRUE)

## End(Not run)
```

goose_clear_category *Clear all items in a category*

Description

Clear all items in a category

Usage

```
goose_clear_category(category, confirm = TRUE, verbose = TRUE)
```

Arguments

category	Character string specifying the category to clear
confirm	Logical, whether to ask for confirmation (default TRUE)
verbose	Logical, whether to print progress messages

Value

Invisible integer count of deleted items

Examples

```
## Not run:
# Clear all items in "temp" category
goose_clear_category("temp")

# Clear without confirmation
goose_clear_category("temp", confirm = FALSE)

## End(Not run)
```

---

goose_clear_tags *Clear all items with specified tags*

Description

Clear all items with specified tags

Usage

```
goose_clear_tags(tags, confirm = TRUE, verbose = TRUE)
```

Arguments

tags	Character vector of tags
confirm	Logical, whether to ask for confirmation (default TRUE)
verbose	Logical, whether to print progress messages

Value

Invisible integer count of deleted items

Examples

```
## Not run:
# Clear all items tagged as "test"
goose_clear_tags("test")

# Clear multiple tags
goose_clear_tags(c("test", "temp", "draft"))

## End(Not run)
```

`goose_configure` *Configure Goose CLI Settings*

Description

Set up Goose CLI configuration. Note: If Goose CLI is already configured (e.g., for Block employees), this function is not needed. The package will use the existing CLI configuration automatically.

Usage

```
goose_configure(
  provider = NULL,
  model = NULL,
  api_key = NULL,
  save_to_renviron = FALSE,
  check_cli_first = TRUE
)
```

Arguments

`provider`	Character string specifying the AI provider (e.g., "openai", "anthropic")
`model`	Character string specifying the model (e.g., "gpt-4", "claude-3")
`api_key`	Character string with the API key (stored securely)
`save_to_renviron`	Logical, whether to save to .Renviron file
`check_cli_first`	Logical, check if CLI already works before configuring (default TRUE)

Value

Invisible TRUE if successful

Examples

```
## Not run:
# For Block employees with configured CLI, just check:
goose_test_cli()

# For external users who need API keys:
goose_configure(provider = "openai", model = "gpt-4", api_key = "your-key")

## End(Not run)
```

---

goose_continuation_prompt

Generate Continuation Prompt for Next Session

Description

Reviews current work and generates a comprehensive continuation prompt with progress tracking, file mapping, and next steps for seamless handoff to the next working session.

Usage

```
goose_continuation_prompt(
  path = ".",
  include_files = TRUE,
  include_todos = TRUE,
  save_to = NULL
)
```

Arguments

path	Path to review (default: current working directory)
include_files	Whether to include file listing (default: TRUE)
include_todos	Whether to scan for TODO comments (default: TRUE)
save_to	Path to save the continuation prompt (auto-generated if NULL)

Value

Path to the saved continuation prompt

Examples

```
## Not run:
# Generate continuation prompt for current project
goose_continuation_prompt()

# Generate without file listing
goose_continuation_prompt(include_files = FALSE)
```

```
# Save to specific location
goose_continuation_prompt(save_to = "project_docs/continuation.md")

## End(Not run)
```

`goose_conversation_ui` *Conversation Manager UI*

Description

Manage and replay AI conversation sessions

Usage

```
goose_conversation_ui()
```

`goose_create_brand` *Create a new brand configuration interactively*

Description

Create a new brand configuration interactively

Usage

```
goose_create_brand(brand_name = NULL, template = "default", interactive = TRUE)
```

Arguments

`brand_name`	Name for the new brand
`template`	Template to use (default: "default")
`interactive`	Use interactive prompts (default: TRUE)

Value

Path to created brand configuration

Examples

```
## Not run:
goose_create_brand("my_company")

## End(Not run)
```

`goose_create_brand_ai` *Create Brand with AI Assistance*

Description

Enhanced brand creation with AI-powered suggestions for colors and typography.

Usage

```
goose_create_brand_ai(
  brand_name,
  industry = NULL,
  style = NULL,
  use_ai = TRUE,
  output_dir = NULL
)
```

Arguments

`brand_name`	Name of the brand
`industry`	Optional industry context for better suggestions
`style`	Optional style preference (e.g., "modern", "classic", "playful")
`use_ai`	Whether to use AI for suggestions (requires goose_ask)
`output_dir`	Directory to save brand configuration

Value

Path to created brand configuration

Examples

```
## Not run:
# Create brand with AI assistance
goose_create_brand_ai("TechStartup",
                      industry = "fintech",
                      style = "modern")

## End(Not run)
```

goose_create_quarto *Create Quarto AI Document*

Description

Generate a complete Quarto document with AI assistance

Usage

```
goose_create_quarto(
  title,
  author = Sys.info()["user"],
  outline,
  format = "html",
  output_file = NULL
)
```

Arguments

title	Document title
author	Document author
outline	Topic outline or description
format	Output format (html, pdf, docx)
output_file	Output file path

Value

Path to created document

goose_create_report *Create RMarkdown Report with AI*

Description

Generate a complete RMarkdown report

Usage

```
goose_create_report(
  title,
  data = NULL,
  analysis_type = "exploratory",
  output_file = NULL
)
```

Arguments

`title`	Report title
`data`	Data frame to analyze
`analysis_type`	Type of analysis
`output_file`	Output file path

Value

Path to created report

`goose_create_template` *Create Parameterized Report Template*

Description

Generate a parameterized Quarto/RMarkdown template

Usage

```
goose_create_template(
  type = "analysis",
  parameters = list(),
  output_file = NULL
)
```

Arguments

`type`	Report type (analysis, dashboard, presentation)
`parameters`	List of parameters
`output_file`	Output file path

Value

Path to template file

goose_delete *Delete Object from Goose Memory*

Description

Remove a saved R object from Goose's memory system.

Usage

```
goose_delete(name, category = "r_objects", global = TRUE, confirm = TRUE)
```

Arguments

name	Character string. Name of the object to delete
category	Character string. Category where the object was saved (default: "r_objects")
global	Logical. If TRUE (default), deletes from global memory. If FALSE, deletes from project-local memory.
confirm	Logical. If TRUE (default), asks for confirmation before deleting.

Value

Invisibly returns TRUE if successful

Examples

```
## Not run:
# Delete an object
goose_delete("old_model", category = "models")

# Delete without confirmation
goose_delete("temp_data", confirm = FALSE)

## End(Not run)
```

goose_divider *Create a markdown-style divider*

Description

Create a markdown-style divider

Usage

```
goose_divider(char = "-", width = 60)
```

Arguments

char	Character to use for divider
width	Width of divider

---

goose_document *Generate R Documentation with AI*

Description

Generate roxygen2 documentation for R functions.

Usage

```
goose_document(func, style = "roxygen2", examples = TRUE)
```

Arguments

func	Function object or function name
style	Character, documentation style ("roxygen2", "detailed", "minimal")
examples	Logical, whether to generate examples

Value

Character string with documentation

Examples

```
## Not run:
my_func <- function(x, y, method = "pearson") {
  cor(x, y, method = method)
}

docs <- goose_document(my_func)
cat(docs)

## End(Not run)
```

goose_exists *Check if an item exists in gooseR memory*

Description

Check if an item exists in gooseR memory

Usage

```
goose_exists(name, category = NULL)
```

Arguments

name	Character string, name of the item
category	Character string, category of the item (optional)

Value

Logical, TRUE if item exists

Examples

```
## Not run:
if (goose_exists("my_data", "analysis")) {
  data <- goose_load("my_data", "analysis")
}

## End(Not run)
```

goose_explain_error *Explain R Error with AI*

Description

Get AI-powered explanation and solution for R errors.

Usage

```
goose_explain_error(error = NULL, code = NULL, context = NULL)
```

Arguments

error	The error object or error message
code	Optional code that caused the error
context	Optional context about what you were trying to do

Value

List with explanation and suggested solutions

Examples

```
## Not run:
# Explain last error
tryCatch({
  data.frame(x = 1:3, y = 1:4)
}, error = function(e) {
  explanation <- goose_explain_error(e)
  print(explanation)
})

## End(Not run)
```

---

goose_format_options *Set global formatting options for gooseR*

Description

Set global formatting options for gooseR

Usage

```
goose_format_options(
  auto_format = TRUE,
  width = 80,
  color = TRUE,
  code_highlight = TRUE
)
```

Arguments

auto_format	Logical, automatically format all AI responses
width	Line width for text wrapping
color	Use colored output
code_highlight	Highlight code blocks

Examples

```
## Not run:
# Enable auto-formatting for all AI responses
goose_format_options(auto_format = TRUE, width = 100)

# Disable colors for plain text output
goose_format_options(color = FALSE)

## End(Not run)
```

goose_generate_tests *Generate Unit Tests with AI*

Description

Generate testthat unit tests for R functions.

Usage

```
goose_generate_tests(func, test_cases = 5, edge_cases = TRUE)
```

Arguments

func	Function object to test
test_cases	Number of test cases to generate
edge_cases	Logical, include edge case tests

Value

Character string with testthat code

goose_get_config *Get Current Goose Configuration*

Description

Get Current Goose Configuration

Usage

```
goose_get_config()
```

Value

List with provider, model, and api_key status

goose_give_sample *Share R Object Structure with Goose*

Description

Captures the structure and sample values of an R object and saves it in a format that Goose can understand and reference when providing code. This enables Goose to write accurate code that references actual column names and understands data types.

Usage

```
goose_give_sample(object, name = NULL, rows = 5, save_to_memory = TRUE)
```

Arguments

object	An R object (data.frame, matrix, tibble, list, etc.)
name	Optional custom name for the object. Defaults to the object's name.
rows	Number of sample rows to include (default: 5)
save_to_memory	Whether to save to Goose memory (default: TRUE)

Value

Invisible NULL. Prints object summary and saves to memory.

Examples

```
## Not run:
# Share a data frame with Goose
goose_give_sample(mtcars)

# Share with custom name
goose_give_sample(iris, "flower_data")

# Share just structure without saving
goose_give_sample(my_model, save_to_memory = FALSE)

## End(Not run)
```

goose_honk *Review and Challenge Current Work*

Description

Analyzes your actual code and data to provide tailored suggestions and challenges. Reads R scripts, RMarkdown files, and available data objects to give specific, contextual feedback rather than generic advice.

Usage

```
goose_honk(path = ".", focus = NULL, severity = "moderate")
```

Arguments

path	Path to script, RMarkdown file, or directory (default: current working directory)
focus	Area to focus review on: "statistics", "visualization", "performance", "methodology", NULL for comprehensive
severity	Level of critique: • "gentle" = Encouraging feedback with light suggestions • "moderate" = Balanced critique with actionable improvements (default) • "harsh" = Direct, no-nonsense feedback for maximum improvement • "brutal" = Unfiltered critique (use with caution!)

Value

List containing review comments, specific code issues, and tailored suggestions

Examples

```
## Not run:
# Review current directory with moderate critique
goose_honk()

# Review specific script with gentle feedback
goose_honk("analysis.R", severity = "gentle")

# Focus on statistics with harsh critique
goose_honk(focus = "statistics", severity = "harsh")

# Get brutal honesty about your visualization code
goose_honk(focus = "visualization", severity = "brutal")

## End(Not run)
```

goose_insert_chunk *Insert AI Chunk in Current Document*

Description

Interactive function to insert AI chunk at cursor

Usage

```
goose_insert_chunk()
```

goose_list *List Objects in Goose Memory*

Description

List all R objects saved in Goose memory, optionally filtered by category or tags.

Usage

```
goose_list(category = NULL, tags = NULL, global = TRUE)
```

Arguments

category	Character string. Filter by category (default: NULL shows all)
tags	Character vector. Filter by tags (default: NULL shows all)
global	Logical. If TRUE (default), lists global memory. If FALSE, lists project-local memory.

Value

A data.frame with information about saved objects

Examples

```
## Not run:
# List all saved objects
goose_list()

# List objects in a specific category
goose_list(category = "models")

# List objects with specific tags
goose_list(tags = "production")

## End(Not run)
```

goose_load *Load R Object from Goose Memory*

Description

Retrieve a previously saved R object from Goose's memory system.

Usage

```
goose_load(name, category = "r_objects", global = TRUE)
```

Arguments

name	Character string. Name of the object to load
category	Character string. Category where the object was saved (default: "r_objects")
global	Logical. If TRUE (default), loads from global memory. If FALSE, loads from project-local memory.

Value

The R object that was saved

Examples

```
## Not run:
# Load a previously saved model
model <- goose_load("mtcars_model", category = "models")

# Load a dataset
data <- goose_load("iris_data", category = "datasets")

## End(Not run)
```

goose_loop_me *Convert Code to Loop Structure*

Description

Takes a script or code block and converts it to an efficient loop structure based on the specified iteration requirements.

Usage

```
goose_loop_me(code, loop_over, iterator_name = "i", parallel = FALSE)
```

Arguments

code	Character string or file path containing the code to loop
loop_over	What to loop over (e.g., "files", "columns", "rows", "list elements")
iterator_name	Name for the loop variable (default: "i")
parallel	Whether to use parallel processing (default: FALSE)

Value

Modified code with loop structure

Examples

```
## Not run:
# Convert file processing to loop
code <- "data <- read.csv('file.csv')\nsummary(data)"
goose_loop_me(code, loop_over = "files")

# Create parallel loop
goose_loop_me("process_data(df)", loop_over = "datasets", parallel = TRUE)

## End(Not run)
```

---

goose_make_a_plan *Generate Analysis Plan from Shared Objects*

Description

Reviews all objects shared in the current session and generates a comprehensive, phased analysis plan with specific recommendations.

Usage

```
goose_make_a_plan(focus = NULL, output_format = "console")
```

Arguments

focus	Optional focus area: "exploratory", "predictive", "descriptive", "diagnostic"
output_format	Format for the plan: "console", "markdown", "html"

Value

Analysis plan as text or formatted output

Examples

```
## Not run:
# Generate comprehensive plan
goose_make_a_plan()

# Focus on predictive modeling
goose_make_a_plan(focus = "predictive")

# Save plan as markdown
goose_make_a_plan(output_format = "markdown")

## End(Not run)
```

---

goose_map *Map Async Function Over Data*

Description

Apply an AI query to each element of a dataset

Usage

```
goose_map(data, query_template, max_workers = 4)
```

Arguments

data	Vector or list of data elements
query_template	Query template with {x} placeholder
max_workers	Maximum parallel workers

Value

List of AI responses

Examples

```
## Not run:
# Analyze multiple code snippets
code_snippets <- c("function(x) x^2", "for(i in 1:10) print(i)")
reviews <- goose_map(code_snippets, "Review this R code: {x}")

## End(Not run)
```

goose_mapreduce *Parallel Map-Reduce with AI*

Description

Map-reduce pattern for AI processing

Usage

```
goose_mapreduce(data, map_query, reduce_query, max_workers = 4)
```

Arguments

data	Input data
map_query	Query template for mapping
reduce_query	Query for reduction
max_workers	Maximum parallel workers

Value

Reduced result

goose_optimize_palette *Optimize Brand Palette with AI*

Description

Get AI suggestions to improve an existing brand palette.

Usage

```
goose_optimize_palette(
  brand,
  goals = c("accessibility", "harmony", "contrast"),
  constraints = NULL
)
```

Arguments

brand	Name of the brand to optimize
goals	Character vector of optimization goals
constraints	Optional constraints (e.g., "keep primary color")

Value

List with original and optimized palettes

goose_optimize_plot *Optimize ggplot2 Code with AI*

Description

Get AI suggestions to improve a ggplot2 visualization.

Usage

```
goose_optimize_plot(
  plot_code,
  goals = c("aesthetics", "clarity", "accessibility"),
  data_sample = NULL
)
```

Arguments

plot_code	Character string or expression with ggplot2 code
goals	Character vector of optimization goals
data_sample	Optional sample of the data being plotted

Value

List with optimized code and suggestions

Examples

```
## Not run:
plot_code <- "
ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point()
"

optimized <- goose_optimize_plot(plot_code,
                                 goals = c("aesthetics", "clarity"))

## End(Not run)
```

goose_pipeline *Create Async Query Pipeline*

Description

Chain multiple async operations together

Usage

```
goose_pipeline(...)
```

Arguments

...	Query functions or strings

Value

A promise chain

Examples

```
## Not run:
goose_pipeline(
  "Write a data analysis function",
  ~ paste("Add error handling to:", .),
  ~ paste("Add documentation to:", .),
  ~ paste("Create tests for:", .)
) %...>% { cat("Final result:", .) }

## End(Not run)
```

goose_quarto *Quarto and RMarkdown Integration for GooseR*

Description

AI-powered chunks and document generation

goose_quarto_chunk *Create AI-Powered Quarto Chunk*

Description

Generate a Quarto chunk that executes AI queries

Usage

```
goose_quarto_chunk(
  prompt,
  label = NULL,
  echo = FALSE,
  eval = TRUE,
  cache = TRUE
)
```

Arguments

prompt	The AI prompt to execute
label	Optional chunk label
echo	Whether to show the prompt in output
eval	Whether to evaluate the chunk
cache	Whether to cache the result

Value

Quarto chunk text

Examples

```
## Not run:
# In a Quarto document
goose_quarto_chunk("Explain linear regression", label = "explain-lm")

## End(Not run)
```

goose_query *Query Goose with Session Context*

Description

Send a query to Goose while maintaining session context.

Usage

```
goose_query(prompt, session = NULL, ...)
```

Arguments

prompt	Character string with the query
session	Goose session object or session ID
...	Additional arguments passed to goose_ask

Value

Query response

goose_recipe *Execute Goose Recipe*

Description

Run a Goose recipe with parameters.

Usage

```
goose_recipe(recipe, params = list(), explain = FALSE, render = FALSE)
```

Arguments

recipe	Character, recipe name or path to recipe file
params	Named list of parameters to pass to recipe
explain	Logical, show recipe explanation instead of running
render	Logical, render recipe instead of running

Value

Recipe output or explanation

goose_reduce *Reduce Results with AI*

Description

Combine multiple results using AI

Usage

```
goose_reduce(results, reduce_prompt = "Summarize these results:")
```

Arguments

results	List of results to combine
reduce_prompt	Prompt for reduction

Value

Combined result

goose_rename *Rename an item in gooseR memory*

Description

Rename an item in gooseR memory

Usage

```
goose_rename(old_name, new_name, category = "general")
```

Arguments

old_name	Character string, current name
new_name	Character string, new name
category	Character string, category of the item

Value

Logical, TRUE if successful

Examples

```
## Not run:
goose_rename("old_analysis", "final_analysis", category = "results")

## End(Not run)
```

goose_rename_columns *Intelligently Rename Survey Columns*

Description

Transforms long survey question text into short, meaningful variable names. Combines janitor::clean_names() functionality with intelligent abbreviation to create readable, consistent column names. Saves a mapping file for reference.

Usage

```
goose_rename_columns(
  data,
  max_length = 20,
  style = c("snake_case", "camelCase", "SCREAMING_SNAKE"),
  abbreviate = TRUE,
  save_map = TRUE,
  map_file = NULL,
  preview_only = FALSE,
  custom_dict = NULL
)
```

Arguments

data	A data frame with columns to rename (typically survey data)
max_length	Maximum length for new column names (default: 20)
style	Naming style: "snake_case" (default), "camelCase", "SCREAMING_SNAKE"
abbreviate	Logical, whether to use intelligent abbreviation (default: TRUE)
save_map	Logical, whether to save the name mapping (default: TRUE)
map_file	Path to save the mapping CSV file (auto-generated if NULL)
preview_only	Logical, if TRUE only shows proposed changes without applying (default: FALSE)
custom_dict	Named vector of custom abbreviations (e.g., c("satisfaction" = "sat"))

Details

The function performs intelligent renaming by:

1. Removing special characters and standardizing format (via clean_names)
2. Detecting common survey patterns (scales, demographics, etc.)
3. Creating meaningful abbreviations for long questions
4. Ensuring uniqueness of all column names
5. Saving a reference map for documentation

Common patterns detected:

- Likert scales ("How satisfied...", "To what extent...")
- Demographics ("What is your age", "Gender", etc.)
- Yes/No questions
- Multiple choice questions
- Open-ended responses

Value

If preview_only = FALSE: Data frame with renamed columns If preview_only = TRUE: Data frame showing the mapping

Examples

```
## Not run:
# Basic usage - rename survey columns
survey_clean <- goose_rename_columns(survey_data)

# Preview changes first
goose_rename_columns(survey_data, preview_only = TRUE)

# Use custom abbreviations
survey_clean <- goose_rename_columns(survey_data,
  custom_dict = c(
    "satisfaction" = "sat",
    "recommendation" = "rec",
    "likelihood" = "likely"
  )
)

# Use camelCase instead of snake_case
survey_clean <- goose_rename_columns(survey_data, style = "camelCase")

## End(Not run)
```

---

goose_review_code *Review R Code with AI*

Description

Get AI-powered code review and suggestions for improvement.

Usage

```
goose_review_code(
  code,
  focus = c("performance", "style", "bugs", "documentation"),
  context = NULL,
  detailed = TRUE
)
```

Arguments

code	Character string or function containing R code to review
focus	Character vector of focus areas (e.g., "performance", "style", "bugs")
context	Optional context about the code's purpose
detailed	Logical, whether to request detailed analysis

Value

List with review results including suggestions, issues, and improvements

Examples

```
## Not run:
# Review a function
my_func <- function(x) {
  for(i in 1:length(x)) {
    x[i] <- x[i] * 2
  }
  return(x)
}
review <- goose_review_code(my_func, focus = c("performance", "style"))

# Review code string
code <- "df$new_col = df$col1 + df$col2"
review <- goose_review_code(code, context = "Adding columns in data frame")

## End(Not run)
```

---

goose_rmd_ai *Create RMarkdown AI Section*

Description

Generate an RMarkdown section with AI content

Usage

```
goose_rmd_ai(title, prompt, level = 2, include_code = TRUE)
```

Arguments

title	Section title
prompt	AI prompt for content
level	Heading level (1-6)
include_code	Include code examples

Value

RMarkdown text

goose_save *Save R Object to Goose Memory*

Description

Save an R object to Goose's memory system with optional tags and category. Objects are serialized to RDS format and metadata is stored in Goose's text format.

Usage

```
goose_save(
  object,
  name,
  category = "r_objects",
  tags = NULL,
  description = NULL,
  global = TRUE,
  overwrite = FALSE
)
```

Arguments

object	The R object to save (can be any R object: data.frame, model, list, etc.)
name	Character string. Name for the saved object (will be used as filename)
category	Character string. Category for organizing memories (default: "r_objects")
tags	Character vector. Optional tags for searching/filtering
description	Character string. Optional description of the object
global	Logical. If TRUE (default), saves to global Goose memory. If FALSE, saves to project-local memory.
overwrite	Logical. If TRUE, overwrites existing object. If FALSE (default), errors if object exists.

Value

Invisibly returns the path where the object was saved

Examples

```
## Not run:
# Save a model
model <- lm(mpg ~ wt, data = mtcars)
goose_save(model, "mtcars_model",
           category = "models",
           tags = c("regression", "mtcars"),
           description = "Linear model predicting mpg from weight")
```

```
# Save a data frame
goose_save(iris, "iris_data",
           category = "datasets",
           tags = "example")

## End(Not run)
```

`goose_session` *Create or Resume Goose Session*

Description

Manage Goose sessions for maintaining context across queries.

Usage

```
goose_session(
  action = c("create", "resume", "list", "remove"),
  name = NULL,
  session_id = NULL
)
```

Arguments

`action`	Character, one of "create", "resume", "list", "remove"
`name`	Optional session name
`session_id`	Optional session ID

Value

Session information or query result

`goose_session_clear` *Clear all items from current session*

Description

Clear all items from current session

Usage

```
goose_session_clear(confirm = TRUE)
```

Arguments

`confirm`	Logical, whether to confirm deletion

Value

Invisible integer count of deleted items

`goose_session_end` *End the current gooseR session*

Description

End the current gooseR session

Usage

```
goose_session_end(cleanup = FALSE)
```

Arguments

`cleanup`	Logical, whether to delete session items

Value

Invisible NULL

`goose_session_list` *List items saved in current session*

Description

List items saved in current session

Usage

```
goose_session_list()
```

Value

Data frame of session items

goose_session_save *Save an object with session tracking*

Description

A wrapper around goose_save that automatically adds the current session tag

Usage

```
goose_session_save(..., tags = NULL)
```

Arguments

...	Arguments passed to goose_save
tags	Additional tags (session tag will be added automatically)

Value

Same as goose_save

goose_session_start *Start a gooseR session for tracking saved items*

Description

Note: To automatically tag items with the session ID, you need to manually add the session tag when saving, or use the wrapper functions.

Usage

```
goose_session_start(session_id = NULL)
```

Arguments

session_id	Character, optional session identifier

Value

Character, the session ID

Examples

```
## Not run:
# Start a session
session_id <- goose_session_start()

# Save items (manually tag with session)
goose_save(mtcars, "cars_data", category = "analysis",
           tags = c("myanalysis", getOption("goose.session_id")))

# See what was saved in this session
goose_session_list()

# Clean up session
goose_session_clear()
goose_session_end()

## End(Not run)
```

`goose_stream` *Stream Response from Goose*

Description

Execute a query with streaming response handling

Usage

```
goose_stream(
  query,
  callback = NULL,
  error_callback = NULL,
  complete_callback = NULL,
  session_id = NULL
)
```

Arguments

`query`	The query to send to Goose
`callback`	Function to call with each chunk (default: cat)
`error_callback`	Function to call on error
`complete_callback`	Function to call on completion
`session_id`	Optional session ID for context

Value

Invisible NULL (results handled via callbacks)

Examples

```
## Not run:
# Simple streaming with default output
goose_stream("Explain R functions")

# Custom callback for processing chunks
goose_stream("Write a function", callback = function(chunk) {
  message("Received: ", nchar(chunk), " characters")
  cat(chunk)
})

## End(Not run)
```

---

goose_streaming *Streaming Response Module for GooseR*

Description

Provides real-time streaming responses from Goose AI

---

goose_stream_async *Async Stream with Promise*

Description

Stream response that returns a promise for async handling

Usage

```
goose_stream_async(query, show_progress = TRUE)
```

Arguments

query	The query to execute
show_progress	Show progress during streaming

Value

A promise that resolves to the complete response

Examples

```
## Not run:
library(promises)
goose_stream_async("Explain promises") %...>%
  { cat("Complete response:", .) }

## End(Not run)
```

goose_stream_multi *Stream Multiple Queries*

Description

Stream multiple queries sequentially with progress

Usage

```
goose_stream_multi(queries, callback = NULL)
```

Arguments

queries	Vector of queries to execute
callback	Optional callback for each response

Value

List of responses

goose_stream_session *Create Streaming Session*

Description

Create a persistent streaming session for multiple queries

Usage

```
goose_stream_session(session_name = NULL)
```

Arguments

session_name	Name for the session

Value

StreamSession object

`goose_suggest_colors` *Suggest Color Palette with AI*

Description

Get AI-powered color palette suggestions for data visualization.

Usage

```
goose_suggest_colors(purpose, n_colors = 5, brand = NULL, constraints = NULL)
```

Arguments

`purpose`	Character, the purpose of the palette (e.g., "heatmap", "categorical", "diverging")
`n_colors`	Integer, number of colors needed
`brand`	Optional brand name for brand-appropriate colors
`constraints`	Optional constraints (e.g., "colorblind-safe", "print-friendly")

Value

List with hex colors and rationale

Examples

```
## Not run:
# Get heatmap colors
colors <- goose_suggest_colors("heatmap", n_colors = 9)

# Get brand colors
colors <- goose_suggest_colors("categorical", n_colors = 5, brand = "Block")

# Get accessible colors
colors <- goose_suggest_colors("bar chart", n_colors = 4,
                               constraints = "colorblind-safe")

## End(Not run)
```

goose_template *Create a Prompt Template*

Description

Define a reusable prompt template with variables

Usage

```
goose_template(
  name,
  template,
  description = NULL,
  variables = NULL,
  examples = NULL
)
```

Arguments

name	Template name
template	Template text with {variable} placeholders
description	Optional description
variables	List of variable definitions
examples	Optional usage examples

Value

Template object

Examples

```
## Not run:
# Create a code review template
review_template <- goose_template(
  name = "code_review",
  template = "Review this {language} code:\n\n{code}\n\nFocus on: {focus}",
  variables = list(
    language = "Programming language",
    code = "Code to review",
    focus = "Specific areas to focus on"
  )
)

## End(Not run)
```

goose_template_apply *Apply Template with Variables*

Description

Fill in template variables and execute query

Usage

```
goose_template_apply(template, ..., execute = TRUE)
```

Arguments

template	Template object or name
...	Variable values
execute	Whether to execute the query

Value

Filled template or query response

Examples

```
## Not run:
# Use the template
result <- goose_template_apply(
  review_template,
  language = "R",
  code = "function(x) x^2",
  focus = "efficiency and style"
)

## End(Not run)
```

goose_template_builtin *Get Built-in Template*

Description

Access built-in template library

Usage

```
goose_template_builtin(name = NULL)
```

Arguments

name	Optional template name

Value

Template or list of templates

---

goose_template_from_query *Create Template from History*

Description

Create a template from a previous query

Usage

```
goose_template_from_query(query, name, variables = NULL)
```

Arguments

query	Previous query text
name	Template name
variables	Variables to extract

Value

Template object

---

goose_template_list *List Available Templates*

Description

Show all available templates

Usage

```
goose_template_list(include_builtin = TRUE)
```

Arguments

include_builtin	Include built-in templates

Value

Data frame of templates

goose_template_load *Load Template from Library*

Description

Load a saved template

Usage

```
goose_template_load(name)
```

Arguments

name	Template name

Value

Template object

goose_template_save *Save Template to Library*

Description

Save a template for future use

Usage

```
goose_template_save(template, overwrite = FALSE, template_dir = NULL)
```

Arguments

template	Template object
overwrite	Overwrite if exists
template_dir	Directory to save templates (default: tempdir() for CRAN compliance)

Value

Logical indicating success

Examples

```
## Not run:
# Create and save template in temp directory
template <- goose_template("test", "Hello {name}")
goose_template_save(template, template_dir = tempdir())

## End(Not run)
```

goose_template_validate *Validate Template*

Description

Check if template is valid

Usage

```
goose_template_validate(template)
```

Arguments

template	Template object or text

Value

List with validation results

goose_test_cli *Test if Goose CLI is Working*

Description

Tests if Goose CLI is properly configured and can execute queries. This is especially useful for Block employees who have CLI configured but don't need to provide API keys in R.

Usage

```
goose_test_cli(verbose = TRUE)
```

Arguments

verbose	Logical, whether to print status messages

Value

Logical, TRUE if CLI works, FALSE otherwise

Examples

```
## Not run:
# Check if CLI works
if (goose_test_cli()) {
  # Ready to use goose_ask() etc.
  response <- goose_ask("Hello!")
} else {
  # May need configuration
  goose_configure(provider = "openai", model = "gpt-4", api_key = "key")
}

## End(Not run)
```

---

goose_ui_components *Visual UI Components for GooseR*

Description

Shiny-based visual interfaces for cache and conversation management

---

goose_version *Get Goose CLI Version*

Description

Get Goose CLI Version

Usage

```
goose_version()
```

Value

Character string with version or NULL if not installed

goose_view_column_map *View Column Name Mapping*

Description

Display or retrieve the column name mapping from a renamed data frame

Usage

```
goose_view_column_map(data, return_df = FALSE)
```

Arguments

data	A data frame that was processed with goose_rename_columns()
return_df	Logical, if TRUE returns the mapping as a data frame (default: FALSE)

Value

If return_df = TRUE: Returns the mapping data frame If return_df = FALSE: Invisibly returns NULL and displays the mapping

Examples

```
## Not run:
# View the mapping
goose_view_column_map(survey_clean)

# Get mapping as data frame
map_df <- goose_view_column_map(survey_clean, return_df = TRUE)

## End(Not run)
```

goose_worker_pool *Create Async Worker Pool*

Description

Create a pool of workers for processing queries

Usage

```
goose_worker_pool(n_workers = 4)
```

Arguments

n_workers	Number of workers

Value

WorkerPool object

`load_brand` *Load brand configuration*

Description

Load brand configuration

Usage

```
load_brand(brand = "block")
```

Arguments

`brand`	Name of the brand to load

Value

List containing brand configuration

Examples

```
## Not run:
config <- load_brand("block")

## End(Not run)
```

`preview_brand` *Preview Brand Configuration*

Description

Generate a preview of brand colors and typography.

Usage

```
preview_brand(brand, output_file = NULL)
```

Arguments

`brand`	Name of the brand to preview
`output_file`	Optional file to save preview

Value

ggplot object with brand preview

print.goose_cache_stats *Print Cache Statistics*

Description

Print Cache Statistics

Usage

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

Arguments

x	Cache statistics object
...	Additional arguments

print.goose_code_review *Print method for goose_code_review*

Description

Print method for goose_code_review

Usage

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

Arguments

x	goose_code_review object
...	additional arguments (unused)

print.goose_error_explanation *Print method for goose_error_explanation*

Description

Print method for goose_error_explanation

Usage

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

Arguments

x	goose_error_explanation object
...	additional arguments (unused)

print.goose_palette *Print method for goose_palette*

Description

Print method for goose_palette

Usage

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

Arguments

x	goose_palette object
...	additional arguments (unused)

print.goose_response *Print formatted AI response*

Description

Print formatted AI response

Usage

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

Arguments

x	AI response object or text
...	Additional arguments passed to format_ai_response

print.goose_session *Print method for goose_session*

Description

Print method for goose_session

Usage

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

Arguments

x	goose_session object
...	additional arguments (unused)

print.goose_template *Print Template*

Description

Print Template

Usage

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

Arguments

x	Template object
...	Additional arguments (unused)

register_goose_engine *Register Goose Chunk Engine*

Description

Register custom knitr engine for Goose chunks

Usage

```
register_goose_engine()
```

StreamHandler *Stream Handler R6 Class*

Description

Manages streaming responses from Goose

Public fields

`process` The processx process object

`buffer` Accumulated response buffer

`callback` Chunk callback function

`error_callback` Error callback function

`complete_callback` Completion callback function Initialize stream handler

Methods

Public methods:

- StreamHandler$new()
- StreamHandler$start()
- StreamHandler$monitor()
- StreamHandler$get_response()
- StreamHandler$stop()
- StreamHandler$clone()

Method `new()`**:**

Usage:

```
StreamHandler$new(callback, error_callback, complete_callback)
```

Arguments:

`callback` Function to call with chunks

`error_callback` Function for errors

`complete_callback` Function for completion Start streaming process

Method `start()`**:**

Usage:

```
StreamHandler$start(query, session_id = NULL)
```

Arguments:

`query` The query to execute

`session_id` Optional session ID Monitor streaming process Get accumulated buffer

Method `monitor()`**:**

Usage:

```
StreamHandler$monitor()
```

Method `get_response()`**:**

Usage:

```
StreamHandler$get_response()
```

Returns: Complete response text Stop streaming

Method `stop()`**:**

Usage:

```
StreamHandler$stop()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`StreamSession` *Stream Session R6 Class*

Description

Manages a persistent streaming session

Public fields

`session_id` Unique session identifier

`history` Query/response history

`active` Whether session is active Initialize session

Methods

Public methods:

- `StreamSession$new()`
- `StreamSession$query()`
- `StreamSession$get_history()`
- `StreamSession$close()`
- `StreamSession$clone()`

Method `new()`:

Usage:

```
StreamSession$new(session_id)
```

Arguments:

`session_id` Session identifier Send query in session

Method `query()`:

Usage:

```
StreamSession$query(query, show_output = TRUE)
```

Arguments:

`query` Query to execute

`show_output` Show streaming output

Returns: Response text Get session history

Method `get_history()`:

Usage:

```
StreamSession$get_history()
```

Returns: Data frame of queries and responses Close session

Method `close()`:

Usage:

```
StreamSession$close()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

`theme_brand` *Generate ggplot2 theme from brand configuration*

Description

Generate ggplot2 theme from brand configuration

Usage

```
theme_brand(
  brand = "block",
  variant = "light",
  base_theme = ggplot2::theme_minimal(),
  base_size = NULL
)
```

Arguments

`brand`	Name of the brand to use
`variant`	Theme variant ("light", "dark", or NULL for default)
`base_theme`	Base ggplot2 theme to build upon (default: theme_minimal())
`base_size`	Base font size (overrides brand config if specified)

Value

A ggplot2 theme object

Examples

```
## Not run:
library(ggplot2)
# Light theme
ggplot(mtcars, aes(mpg, wt)) +
  geom_point() +
  theme_brand("block", variant = "light")

# Dark theme
```

```
ggplot(mtcars, aes(mpg, wt)) +
  geom_point() +
  theme_brand("block", variant = "dark")

## End(Not run)
```

`validate_brand` *Validate brand configuration*

Description

Validate brand configuration

Usage

```
validate_brand(config)
```

Arguments

`config`	Brand configuration list

Value

Logical indicating if configuration is valid

`with_goose_session` *Execute code with automatic gooseR session management*

Description

Execute code with automatic gooseR session management

Usage

```
with_goose_session(expr, cleanup = TRUE, session_id = NULL)
```

Arguments

`expr`	Expression to evaluate
`cleanup`	Logical, whether to clean up after execution
`session_id`	Optional session identifier

Value

Result of expression

Examples

```
## Not run:
# Run analysis with automatic cleanup
result <- with_goose_session({
  goose_save(mtcars, "cars", category = "temp")
  goose_save(iris, "flowers", category = "temp")
  # Do analysis...
  "Analysis complete"
}, cleanup = TRUE)

## End(Not run)
```

`WorkerPool` *Worker Pool R6 Class*

Description

Manages a pool of async workers

Public fields

`n_workers` Number of workers

`queue` Query queue

`results` Results list

`active` Whether pool is active Initialize worker pool

Methods

Public methods:

- `WorkerPool$new()`
- `WorkerPool$add()`
- `WorkerPool$process()`
- `WorkerPool$get_results()`
- `WorkerPool$clear_queue()`
- `WorkerPool$shutdown()`
- `WorkerPool$clone()`

Method `new()`:

Usage:

```
WorkerPool$new(n_workers = 4)
```

Arguments:

`n_workers` Number of workers Add query to queue

Method `add()`:

Usage:

```
WorkerPool$add(query, id = NULL)
```

Arguments:

`query` Query to add

`id` Optional query ID Process all queued queries

Method `process()`**:**

Usage:

```
WorkerPool$process(progress = TRUE)
```

Arguments:

`progress` Show progress

Returns: List of results Get results

Method `get_results()`**:**

Usage:

```
WorkerPool$get_results()
```

Returns: Results list Clear queue Shutdown pool

Method `clear_queue()`**:**

Usage:

```
WorkerPool$clear_queue()
```

Method `shutdown()`**:**

Usage:

```
WorkerPool$shutdown()
```

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

Usage:

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

Arguments:

`deep` Whether to make a deep clone.

Index

brand_css, 4
brand_palette, 5
brand_rmd_template, 5

format_ai_response, 6
format_conversation, 7

goose_addin_chat, 7
goose_addin_quick, 8
goose_addin_review, 8
goose_addin_snippet, 8
goose_addin_template, 8
goose_addins, 7
goose_ask, 9
goose_ask_raw, 10
goose_async, 11
goose_async_retry, 11
goose_async_timeout, 12
goose_backup, 12
goose_batch, 13
goose_batch_file, 14
goose_cache, 14
goose_cache_clear, 15
goose_cache_export, 16
goose_cache_get, 16
goose_cache_import, 17
goose_cache_init, 17
goose_cache_set, 18
goose_cache_stats, 18
goose_cache_ui, 19
goose_cache_warmup, 19
goose_cached, 15
goose_check_installation, 20
goose_clean_text, 20
goose_clear_all, 21
goose_clear_category, 21
goose_clear_tags, 22
goose_configure, 23
goose_continuation_prompt, 24
goose_conversation_ui, 25
goose_create_brand, 25
goose_create_brand_ai, 26
goose_create_quarto, 27
goose_create_report, 27
goose_create_template, 28
goose_delete, 29
goose_divider, 29
goose_document, 30
goose_exists, 31
goose_explain_error, 31
goose_format_options, 32
goose_generate_tests, 33
goose_get_config, 33
goose_give_sample, 34
goose_honk, 35
goose_insert_chunk, 36
goose_list, 36
goose_load, 37
goose_loop_me, 37
goose_make_a_plan, 38
goose_map, 39
goose_mapreduce, 40
goose_optimize_palette, 40
goose_optimize_plot, 41
goose_pipeline, 42
goose_quarto, 42
goose_quarto_chunk, 43
goose_query, 44
goose_recipe, 44
goose_reduce, 45
goose_rename, 45
goose_rename_columns, 46
goose_review_code, 47
goose_rmd_ai, 48
goose_save, 49
goose_session, 50
goose_session_clear, 50
goose_session_end, 51
goose_session_list, 51

goose_session_save, 52
goose_session_start, 52
goose_stream, 53
goose_stream_async, 54
goose_stream_multi, 55
goose_stream_session, 55
goose_streaming, 54
goose_suggest_colors, 56
goose_template, 57
goose_template_apply, 58
goose_template_builtin, 58
goose_template_from_query, 59
goose_template_list, 59
goose_template_load, 60
goose_template_save, 60
goose_template_validate, 61
goose_test_cli, 61
goose_ui_components, 62
goose_version, 62
goose_view_column_map, 63
goose_worker_pool, 63
gooseR-branding, 7

load_brand, 64

preview_brand, 64
print.goose_cache_stats, 65
print.goose_code_review, 65
print.goose_error_explanation, 66
print.goose_palette, 66
print.goose_response, 67
print.goose_session, 67
print.goose_template, 68

register_goose_engine, 68

StreamHandler, 68
StreamSession, 70

theme_brand, 71

validate_brand, 72

with_goose_session, 72
WorkerPool, 73