Package: btw (via r-universe)

July 3, 2026

Title A Toolkit for Connecting R and Large Language Models

Version 1.3.0

Description A complete toolkit for connecting 'R' environments with Large Language Models (LLMs). Provides utilities for describing 'R' objects, package documentation, and workspace state in plain text formats optimized for LLM consumption. Supports multiple workflows: interactive copy-paste to external chat interfaces, programmatic tool registration with 'ellmer' chat clients, batteries-included chat applications via 'shinychat', and exposure to external coding agents through the Model Context Protocol. Project configuration files enable stable, repeatable conversations with project-specific context and preferred LLM settings.

License MIT + file LICENSE

URL https://github.com/posit-dev/btw, https://posit-dev.github.io/btw/

BugReports https://github.com/posit-dev/btw/issues

Depends R (>= 4.2.0)

Imports brio, cli, clipr, dplyr, ellmer (>= 0.3.0), frontmatter, fs, jsonlite, lifecycle, mcptools, pkgsearch, rlang (>= 1.1.0), rmarkdown, rstudioapi, S7, sessioninfo, skimr, utils, withr, xml2

Suggests bslib (>= 0.11.0), callr, chromote, covr, DBI, devtools, diffviewer, duckdb, evaluate, fansi, gert, gh, htmltools, pandoc, pkgload, processx, ragg, Rapp (>= 0.3.0), renv, roxygen2, shiny, shinychat (>= 0.4.0), testthat (>= 3.0.0), tibble, usethis

Config/Needs/website brand.yml, tidyverse/tidytemplate

Config/roxygen2/version 8.0.0

Config/testthat/edition 3

Config/testthat/parallel true

Config/testthat/start-first web, news, covr, search

Encoding UTF-8

Collate 'aaa-tools.R' 'addins.R' 'btw-package.R' 'btw.R' 'btw_client.R' 'btw_client_app.R' 'btw_task.R' 'btw_this.R' 'cli.R' 'clipboard.R' 'deprecated.R' 'edit_btw_md.R' 'import-standalone-obj-type.R' 'import-standalone-purrr.R' 'import-standalone-types-check.R' 'mcp.R' 'task_create_btw_md.R' 'task_create_readme.R' 'task_create_skill.R' 'tool-result.R' 'tool-agent-subagent.R' 'tool-agent-custom.R' 'tool-cran.R' 'tool-docs-news.R' 'tool-docs.R' 'tool-env-df.R' 'tool-env.R' 'tool-files-edit.R' 'tool-files-list.R' 'tool-files-patch.R' 'tool-files-read.R' 'tool-files-replace.R' 'tool-files-search.R' 'tool-files-write.R' 'tool-git.R' 'tool-github.R' 'tool-ide.R' 'tool-pkg-covr.R' 'tool-pkg-devtools.R' 'tool-run.R' 'tool-session-package-installed.R' 'tool-sessioninfo.R' 'tool-skills.R' 'tool-web.R' 'tools.R' 'utils-ellmer.R' 'utils-gitignore.R' 'utils-ide.R' 'utils-md.R' 'utils-r.R' 'utils.R' 'zzz.R'

NeedsCompilation no

Author Garrick Aden-Buie [aut, cre] (ORCID: <https://orcid.org/0000-0002-7111-0077>), Simon Couch [aut] (ORCID: <https://orcid.org/0000-0001-5676-5107>), Joe Cheng [aut], Posit Software, PBC [cph, fnd], Google [cph] (Material Design Icons), Microsoft [cph] (@vscode/codicons), Jamie Perkins [cph] (countUp.js author)

Maintainer Garrick Aden-Buie <garrick@adenbuie.com>

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

Date/Publication 2026-07-02 20:10:02 UTC

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

RemoteRef HEAD

RemoteSha 10a9424dd9458d7c23f56f3b5dc41adb2c9e72b6

Contents

`btw` *Plain-text descriptions of R objects*

Description

This function allows you to quickly describe your computational environment to a model by concatenating plain-text descriptions of "R stuff", from data frames to packages to function documentation.

There are two key ways to use `btw()`:

1. Use it interactively at the console to gather information about your environment into prompt text that you can paste into the chat interface of an LLM, like ChatGPT or Claude. By default, `btw()` copies the prompt to the clipboard for you.

   ```
   btw(vignette("colwise", "dplyr"), dplyr::across, dplyr::starwars)
   #> btw copied to the clipboard!
   ```

2. Pair `btw()` with ellmer::Chat during a chat session to create a prompt that includes additional context drawn from your environment and help pages.

   ```
   library(ellmer)

   chat <- chat_anthropic() # requires an Anthropic API key
   chat <- chat_ollama(model = "llama3.1:8b") # requires ollama and a local model

   chat$chat(btw(
     vignette("colwise", "dplyr"),
     dplyr::across,
     dplyr::starwars,
     "Create a few interesting examples that use `dplyr::across()`",
     "with the `starwars` data set."
   ))
   ```

Additional examples:

1. Use `btw()` without arguments to describe all objects in your workspace:

   ```
   btw()
   #> btw copied to the clipboard!
   ```

2. Describe a function (it's documentation) and a data frame:

   ```
   btw(dplyr::mutate, mtcars)
   #> btw copied to the clipboard!
   ```

3. Use `btw()` to give additional context to an ellmer::Chat session:

   ```
   library(ellmer)

   chat <- chat_ollama(model = "llama3.1:8b")
   chat$chat(
     btw(mtcars, "Are there cars with 8 cylinders in this dataset?")
   )
   ```

Usage

```
btw(..., clipboard = TRUE)
```

Arguments

`...`	Objects to describe from your R environment. You can pass objects themselves, like data frames or functions, or the function also accepts output from `btw_tool_*()` functions like `btw_tool_docs_package_help_topics()`, `btw_tool_docs_help_page()` etc. If omitted, this function will just describe the elements in your global R environment.
`clipboard`	Whether to write the results to the clipboard. A single logical value; will default to `TRUE` when run interactively.

Value

Returns an ellmer::ContentText object with the collected prompt. If `clipboard = TRUE`, the prompt text is copied to the clipboard when the returned object is printed for the first time (e.g. calling `btw()` without assignment).

Examples

```
# See documentation for detailed examples
btw(mtcars)
```

`btw_agent_tool` *Create a custom agent tool from a markdown file*

Description

Creates an `ellmer::tool()` from a markdown file that defines a custom agent. The tool can be registered with a chat client to delegate tasks to a specialized assistant with its own system prompt and tool configuration.

Agent File Format:

Agent files use YAML frontmatter to configure the agent, with the markdown body becoming the agent's system prompt. The file should be named `agent-{name}.md`.

Required Fields:

- `name`: A valid R identifier (letters, numbers, underscores) that becomes part of the tool name: `btw_tool_agent_{name}`. The final name cannot conflict with any existing `btw_tools()` names.

Optional Fields:

- `description`: Tool description shown to the LLM. Defaults to a generic delegation message.
- `title`: User-facing title for the tool. Defaults to title-cased name.

- `icon`: Icon specification for the agent (see **Icon Specification** below). Defaults to the standard agent icon.
- `client`: Model specification like `"anthropic/claude-sonnet-4-20250514"`. Falls back to `btw.subagent.client` or `btw.client` options.
- `tools`: List of tool names or groups available to this agent. Defaults to all non-agent tools.

Icon Specification:
The `icon` field supports three formats:

1. **Plain icon name**: Uses `shiny::icon()` (Font Awesome icons). Example: `icon: robot` or `icon: code`
2. **Raw SVG**: Starts with `<svg` and is used literally. Example: `icon: '<svg viewBox="0 0 24 24">...</svg>'`
3. **Package-prefixed icon**: Uses `pkg::icon-name` format to specify icons from other icon packages. Supported packages:

Package	Syntax	Function Called
fontawesome	`fontawesome::home`	`fontawesome::fa()`
bsicons	`bsicons::house`	`bsicons::bs_icon()`
phosphoricons	`phosphoricons::house`	`phosphoricons::ph()`
rheroicons	`rheroicons::home`	`rheroicons::rheroicon()`
tabler	`tabler::home`	`tabler::icon()`
shiny	`shiny::home`	`shiny::icon()`

The specified package must be installed. If the package is missing or the icon name is invalid, a warning is issued and the default agent icon is used.

Example Agent File:

```
---
name: code_reviewer
description: Reviews code for best practices and potential issues.
title: Code Reviewer
icon: magnifying-glass
tools:
  - files
  - docs
---

You are a code reviewer. Analyze code for:
- Best practices and style
- Potential bugs or issues
- Performance considerations

Provide specific, actionable feedback.
```

Automatic Discovery:
Agent files are automatically discovered by `btw_tools()` when placed in the following locations (in order of priority):

- **Project level (btw)**: `.btw/agent-*.md` in your project directory

- **User level (btw)**: `~/.btw/agent-*.md` or `~/.config/btw/agent-*.md`
- **Project level (Claude Code)**: `.claude/agents/*.md` in your project directory
- **User level (Claude Code)**: `~/.claude/agents/*.md`

btw-style agents take precedence over Claude Code agents with the same name. When duplicate agent names are found, a warning is issued.

Claude Code Compatibility:

btw supports loading agent files from Claude Code's `.claude/agents/` directory for compatibility. However, there are some small differences when Claude Code agents are used in btw:

- **Name normalization**: Agent names with hyphens (e.g., `code-reviewer`) are automatically converted to underscores (`code_reviewer`) for R compatibility.
- **Ignored fields**: The following Claude Code fields are ignored (with a warning): `model`, `tools`, `permissionMode`, `skills`. Use btw's `client` field instead of `model`, and btw agents use default tools.
- `client` **argument**: Use the `client` argument to manually override the model for any agent file.

Usage

```
btw_agent_tool(path, client = NULL)
```

Arguments

`path`	Path to an agent markdown file.
`client`	Optional. A client specification to override the agent's configured client. Can be a string like `"anthropic/claude-sonnet-4-20250514"`, an ellmer::Chat object, or a list with `provider` and `model` keys. If `NULL` (default), uses the `client` field from the agent file or falls back to btw's default client resolution.

Value

An `ellmer::ToolDef` object that can be registered with a chat client, or `NULL` if the file is invalid (with a warning).

See Also

`btw_tools()` for automatic agent discovery, `btw_client()` for creating chat clients with tools.

Examples

```
# Create a btw-style agent file
withr::with_tempdir({
  dir.create(".btw")
  writeLines(
    c(
      "---",
      "name: code_reviewer",
      "description: Reviews code for best practices.",
      "---",
```

```
      "",
      "You are a code reviewer. Analyze code for best practices."
    ),
    ".btw/agent-code_reviewer.md"
  )

  tool <- btw_agent_tool(".btw/agent-code_reviewer.md")
  # Use `chat$register_tool(tool)` to register with an ellmer chat client

  tool
})

# Create a Claude Code-style agent file (name with hyphens)
withr::with_tempdir({
  dir.create(".claude/agents", recursive = TRUE)
  writeLines(
    c(
      "---",
      "name: test-helper",
      "description: Helps write tests.",
      "model: sonnet",
      "---",
      "",
      "You help write tests for R code."
    ),
    ".claude/agents/test-helper.md"
  )

  tool <- btw_agent_tool(".claude/agents/test-helper.md")
  # Use `chat$register_tool(tool)` to register with an ellmer chat client

  tool
})
```

`btw_client` *Create a btw-enhanced ellmer chat client*

Description

Creates an ellmer::Chat client, enhanced with the tools from `btw_tools()`. Use `btw_client()` to create the chat client for general or interactive use at the console, or `btw_app()` to create a chat client and launch a Shiny app for chatting with a btw-enhanced LLM in your local workspace.

Project Context:

You can keep track of project-specific rules, guidance and context by adding a `btw.md` file, `AGENTS.md`, or `CLAUDE.md` in your project directory. See `use_btw_md()` for help creating a `btw.md` file in your project, or use `path_btw` to tell `btw_client()` to use a specific context file. Note that `CLAUDE.md` files will have their YAML frontmatter stripped but not used for configuration.

`btw_client()` will also include context from an `llms.txt` file in the system prompt, if one is found in your project directory or as specified by the `path_llms_txt` argument.

Client Settings with User-Level Fallback:

Client settings in `client` and `tools` from a project-level `btw.md` or `AGENTS.md` file take precedence. If a project file doesn't specify a setting, btw will fall back to settings in a user-level `btw.md` file (typically in `~/btw.md` or `~/.config/btw/btw.md`). Project-level btw tool options under the `options` key are merged with user-level options, with project-level options taking precedence.

Project-specific instructions from both files are combined with a divider, allowing you to maintain global guidelines in your user file and project-specific rules in your project file.

Client Options:

The following R options are consulted when creating a new btw chat client and take precedence over settings in a `btw.md` file:

- `btw.client`: The ellmer::Chat client or a `provider/model` string (see `ellmer::chat()`) to use as the basis for new `btw_client()` or `btw_app()` chats.
- `btw.tools`: The btw tools to include by default when starting a new btw chat, see `btw_tools()` for details.‘

Multiple Providers and Models:

You can configure multiple client options in your `btw.md` file. When `btw_client()` is called interactively from the console, you'll be presented with a menu to choose which client to use. In non-interactive contexts, the first client is used automatically.

Array format (unnamed list):

```
client:
  - anthropic/claude-sonnet-4
  - openai/gpt-4.1
  - aws_bedrock/us.anthropic.claude-sonnet-4-20250514-v1:0
```

Alias format (named list):

```
client:
  haiku: aws_bedrock/us.anthropic.claude-haiku-4-5-20251001-v1:0
  sonnet:
    provider: aws_bedrock
    model: us.anthropic.claude-sonnet-4-5-20250929-v1:0
```

With aliases, you can select a client by name in the interactive menu or pass the alias directly: `btw_client(client = "sonnet")`.

Usage

```
btw_client(
  ...,
  client = NULL,
  tools = NULL,
  path_btw = NULL,
  path_llms_txt = NULL
)
```

```
btw_app(
  ...,
  client = NULL,
  tools = NULL,
  path_btw = NULL,
  messages = list(),
  model_choices = c("auto", "btw_md", "provider", "none")
)
```

Arguments

`...`	In `btw_app()`, additional arguments are passed to `shiny::shinyApp()`. In `btw_client()`, additional arguments are ignored.
`client`	An ellmer::Chat client, or a `provider/model` string to be passed to `ellmer::chat()` to create a chat client, or an alias to a client setting in your `btw.md` file (see "Multiple Providers" section). Defaults to `ellmer::chat_anthropic()`. You can use the `btw.client` option to set a default client for new `btw_client()` calls, or use a `btw.md` project file for default chat client settings, like provider and model. We check the `client` argument, then the `btw.client` R option, and finally the `btw.md` project file (falling back to user-level `btw.md` if needed), using only the client definition from the first of these that is available.
`tools`	A list of tools to include in the chat, defaults to `btw_tools()`. Join `btw_tools()` with additional tools defined by `ellmer::tool()` to include additional tools in the chat client. Alternatively, you can use a character values to refer to specific btw tools by name or by group. For example, use `tools = "docs"` to include only the documentation related tools, or `tools = c("env", "docs")` to include the environment and documentation tools, and so on. You can also refer to btw tools by name, e.g. `tools = "btw_tool_docs_help_page"` or alternatively in the shorter form `tools = "docs_help_page"`. Finally, set `tools = FALSE` to skip registering **btw** tools with the chat client.
`path_btw`	A path to a `btw.md`, `AGENTS.md`, or `CLAUDE.md` project context file. If `NULL`, btw will find a project-specific `btw.md`, `AGENTS.md`, or `CLAUDE.md` file in the parents of the current working directory, with fallback to user-level `btw.md` if no project file is found. Set `path_btw = FALSE` to create a chat client without using a `btw.md` file.
`path_llms_txt`	A path to an `llms.txt` file containing context about the current project. By default, btw will look for an `llms.txt` file in the your current working directory or its parents. Set `path_llms_txt = FALSE` to skip looking for an `llms.txt` file.
`messages`	A list of initial messages to show in the chat, passed to `shinychat::chat_mod_ui()`.
`model_choices`	Can be one of `"btw_md"` (model choices from your `path_btw` configuration), `"provider"` (models from the provider API), `"auto"` (uses `path_btw` if `client` comes from `path_btw`, otherwise falling back to provider), or `"none"` (don't show model choices).

Value

Returns an ellmer::Chat object with additional tools registered from `btw_tools()`. `btw_app()` returns the chat object invisibly, and the chat object with the messages added during the chat session.

Functions

- `btw_client()`: Create a btw-enhanced ellmer::Chat client
- `btw_app()`: Create a btw-enhanced client and launch a Shiny app to chat

Examples

```
withr::local_options(list(
  btw.client = ellmer::chat_ollama(model="llama3.1:8b")
))

chat <- btw_client()
chat$chat(
  "How can I replace `stop()` calls with functions from the cli package?"
)
```

`btw_skill_install_github`

Install a skill from GitHub

Description

Download and install a skill from a GitHub repository. The repository should contain one or more skill directories, each with a `SKILL.md` file.

Usage

```
btw_skill_install_github(
  repo,
  skill = NULL,
  scope = "project",
  overwrite = NULL
)
```

Arguments

`repo`	GitHub repository in `"owner/repo"` format. Optionally include a Git reference (branch, tag, or SHA) as `"owner/repo@ref"`, following the convention used by `pak::pak()` and `remotes::install_github()`. Defaults to `"HEAD"` when no ref is specified.
`skill`	Optional skill name. If `NULL` and the repository contains multiple skills, an interactive picker is shown (or an error in non-interactive sessions).

scope — Where to install the skill. One of:

- "project" (default): Installs to a project-level skills directory, chosen from .btw/skills/ or .agents/skills/ in that order. If one already exists, it is used; otherwise .btw/skills/ is created.
- "user": Installs to the first of ~/.btw/skills, ~/.config/btw/skills, or tools::R_user_dir("btw")/skills that already exists, defaulting to ~/.btw/skills if none do.
- A directory path: Installs to a custom directory, e.g. scope = ".openhands/skills". Use I("project") or I("user") if you need a literal directory with those names.

overwrite — Whether to overwrite an existing skill with the same name. If NULL (default), prompts interactively when a conflict exists; in non-interactive sessions defaults to FALSE, which errors. Set to TRUE to always overwrite, or FALSE to always error on conflict.

Value

The path to the installed skill directory, invisibly.

See Also

Other skills: btw_skill_install_package(), btw_skill_install_project(), btw_tool_skill()

---

btw_skill_install_package

Install a skill from an R package

Description

Install a skill bundled in an R package. Packages can bundle skills in their inst/skills/ directory, where each subdirectory containing a SKILL.md file is a skill.

Note that if a package is attached with library(), its skills are **automatically available** without installation — btw discovers skills from all attached packages at runtime. Use this function when you want to permanently copy a skill to your project or user directory so it remains available regardless of which packages are loaded.

Usage

```
btw_skill_install_package(
  package,
  skill = NULL,
  scope = "project",
  overwrite = NULL
)
```

Arguments

package	Name of an installed R package that bundles skills.
skill	Optional skill name. If NULL and the package contains multiple skills, an interactive picker is shown (or an error in non-interactive sessions).
scope	Where to install the skill. One of: • "project" (default): Installs to a project-level skills directory, chosen from .btw/skills/ or .agents/skills/ in that order. If one already exists, it is used; otherwise .btw/skills/ is created. • "user": Installs to the first of ~/.btw/skills, ~/.config/btw/skills, or tools::R_user_dir("btw")/skills that already exists, defaulting to ~/.btw/skills if none do. • A directory path: Installs to a custom directory, e.g. scope = ".openhands/skills". Use I("project") or I("user") if you need a literal directory with those names.
overwrite	Whether to overwrite an existing skill with the same name. If NULL (default), prompts interactively when a conflict exists; in non-interactive sessions defaults to FALSE, which errors. Set to TRUE to always overwrite, or FALSE to always error on conflict.

Value

The path to the installed skill directory, invisibly.

See Also

Other skills: btw_skill_install_github(), btw_skill_install_project(), btw_tool_skill()

---

btw_skill_install_project

Install skills from all project dependencies

Description

Discovers R packages that are dependencies of the current project and installs skills from any that bundle them in inst/skills/. If a DESCRIPTION file exists in the working directory, packages are read from its Imports and Suggests fields. Otherwise, renv::dependencies() is used as a fallback (requires the renv package).

Packages without skills are silently skipped. If no dependencies bundle skills, a message is printed and nothing is installed.

Usage

```
btw_skill_install_project(path = ".", scope = "project", overwrite = NULL)
```

Arguments

path	Path to the project directory. Defaults to the current working directory.
scope	Where to install the skills. See btw_skill_install_package() for details.
overwrite	Whether to overwrite existing skills. See btw_skill_install_package() for details.

Value

The paths to all installed skill directories, invisibly.

See Also

Other skills: btw_skill_install_github(), btw_skill_install_package(), btw_tool_skill()

---

btw_task *Run a pre-formatted btw task*

Description

Runs a btw task defined in a file with YAML frontmatter configuration and a markdown body containing the task prompt. The task file format is similar to btw.md files, with client and tool configuration in the frontmatter and the task instructions in the body.

Task File Format:

Task files use the same format as btw.md files:

```
---
client:
  provider: anthropic
  model: claude-sonnet-4
tools: [docs, files]
---

Your task prompt here with {{ variable }} interpolation...
```

Template Variables:

The task prompt body supports template variable interpolation using {{ variable }} syntax via ellmer::interpolate(). Pass named arguments to provide values for template variables:

```
btw_task("my-task.md", package_name = "dplyr", version = "1.1.0")
```

Additional Context:

Unnamed arguments are treated as additional context and converted to text using btw(). This context is appended to the system prompt:

```
btw_task("analyze.md", dataset_name = "mtcars", mtcars, my_function)
#                      ^-- template var        ^-- additional context
```

Usage

```
btw_task(
  path,
  ...,
  client = NULL,
  mode = c("app", "console", "client", "tool")
)
```

Arguments

path	Path to the task file containing YAML configuration and prompt.
...	Named arguments become template variables for interpolation in the task prompt. Unnamed arguments are treated as additional context objects and converted to text via btw().
client	An ellmer::Chat client to override the task file's client configuration. If NULL, uses the client specified in the task file's YAML frontmatter, falling back to the default client resolution of btw_client().
mode	The execution mode for the task: • "app": Launch interactive Shiny app (default) • "console": Interactive console chat with ellmer::live_console() • "client": Return configured ellmer::Chat client without running • "tool": Return an ellmer::tool() object for programmatic use

Value

Depending on mode:

- "app": Returns the chat client invisibly after launching the app
- "console": Returns the chat client after console interaction
- "client": Returns the configured chat client
- "tool": Returns an ellmer::tool() object

See Also

Other task and agent functions: btw_task_create_btw_md(), btw_task_create_readme(), btw_task_create_skill()

Examples

```
# Create a simple task file
tmp_task_file <- tempfile(fileext = ".md")

cat(file = tmp_task_file, '---
client: anthropic/claude-sonnet-4-6
tools: [docs, files]
---

Analyze the {{ package_name }} package and create a summary.
```

```
')

# Task with template interpolation
btw_task(tmp_task_file, package_name = "dplyr", mode = "tool")

# Include additional context
btw_task(
  tmp_task_file,
  package_name = "ggplot2",
  mtcars,  # Additional context
  mode = "tool"
)
```

`btw_task_create_btw_md`

Task: Initialize Project Context File

Description

Create a comprehensive context `btw.md` or `AGENTS.md` file for your project. If launched in app or console mode, this task will start an interactive chat session to guide you through the process of creating a context file.

This task focuses on documenting project context for developers and agents. See `btw_client()` for additional details about the format and usage of the `btw.md` context file, including choosing the default LLM provider and model or the default set of tools to use with `btw_client()`.

Usage

```
btw_task_create_btw_md(
  ...,
  path = "btw.md",
  client = NULL,
  mode = c("app", "console", "client", "tool")
)
```

Arguments

`...`	Additional context to provide to the AI. This can be any text or R objects that can be converted to text using `btw()`.
`path`	The path to the context file to create. Defaults to `btw.md`.
`client`	An ellmer::Chat client, or a `provider/model` string to be passed to `ellmer::chat()` to create a chat client, or an alias to a client setting in your `btw.md` file (see "Multiple Providers" section). Defaults to `ellmer::chat_anthropic()`. You can use the `btw.client` option to set a default client for new `btw_client()` calls, or use a `btw.md` project file for default chat client settings, like provider and model. We check the `client` argument, then the `btw.client` R option,

and finally the `btw.md` project file (falling back to user-level `btw.md` if needed), using only the client definition from the first of these that is available.

`mode` The mode to run the task in, which affects what is returned from this function. `"app"` and `"console"` modes launch interactive sessions, while `"client"` and `"tool"` modes return objects for programmatic use.

Value

When `mode` is `"app"` or `"console"`, this function launches an interactive session in the browser or the R console, respectively. The ellmer chat object with the conversation history is returned invisibly when the session ends.

When `mode` is `"client"`, this function returns the configured ellmer chat client object. When `mode` is `"tool"`, this function returns an ellmer tool object that can be used in other chat instances.

See Also

Other task and agent functions: `btw_task()`, `btw_task_create_readme()`, `btw_task_create_skill()`

Examples

```
withr::with_envvar(list(ANTHROPIC_API_KEY = "example"), {
  btw_task_create_btw_md(mode = "tool", client = "anthropic")
})
```

`btw_task_create_readme`

Task: Create a Polished README

Description

Create a compelling, user-focused README file for your project. If launched in app or console mode, this task will start an interactive chat session to guide you through the process of creating a polished README that clearly communicates value and helps potential users make informed decisions.

This task focuses on creating READMEs for END USERS, not developers, with emphasis on clarity, accessibility, and authentic communication of value. The process involves exploring your project files, understanding your target audience and goals, proposing a structure, and then iteratively drafting each section with your input.

Usage

```
btw_task_create_readme(
  ...,
  client = NULL,
  mode = c("app", "console", "client", "tool")
)
```

Arguments

`...`	Additional context to provide to the AI. This can be any text or R objects that can be converted to text using `btw()`.
`client`	An ellmer::Chat client, or a `provider/model` string to be passed to `ellmer::chat()` to create a chat client, or an alias to a client setting in your `btw.md` file (see "Multiple Providers" section). Defaults to `ellmer::chat_anthropic()`. You can use the `btw.client` option to set a default client for new `btw_client()` calls, or use a `btw.md` project file for default chat client settings, like provider and model. We check the `client` argument, then the `btw.client` R option, and finally the `btw.md` project file (falling back to user-level `btw.md` if needed), using only the client definition from the first of these that is available.
`mode`	The mode to run the task in, which affects what is returned from this function. `"app"` and `"console"` modes launch interactive sessions, while `"client"` and `"tool"` modes return objects for programmatic use.

Value

When `mode` is `"app"` or `"console"`, this function launches an interactive session in the browser or the R console, respectively. The ellmer chat object with the conversation history is returned invisibly when the session ends.

When `mode` is `"client"`, this function returns the configured ellmer chat client object. When `mode` is `"tool"`, this function returns an ellmer tool object that can be used in other chat instances.

See Also

Other task and agent functions: `btw_task()`, `btw_task_create_btw_md()`, `btw_task_create_skill()`

Examples

```
withr::with_envvar(list(ANTHROPIC_API_KEY = "example"), {
  btw_task_create_readme(mode = "tool", client = "anthropic")
})
```

---

btw_task_create_skill *Task: Create a Skill*

Description

Create a new skill for your project using interactive guidance. If launched in app or console mode, this task will start an interactive chat session to guide you through the process of creating a skill that extends Claude's capabilities with specialized knowledge, workflows, or tool integrations.

Usage

```
btw_task_create_skill(
  ...,
  name = NULL,
  client = NULL,
  mode = c("app", "console", "client", "tool"),
  tools = "docs"
)
```

Arguments

`...`	Additional context to provide to the AI. This can be any text or R objects that can be converted to text using `btw()`.
`name`	Optional skill name. If provided, the AI will skip the naming step and use this name directly.
`client`	An ellmer::Chat client, or a `provider/model` string to be passed to `ellmer::chat()` to create a chat client, or an alias to a client setting in your `btw.md` file (see "Multiple Providers" section). Defaults to `ellmer::chat_anthropic()`. You can use the `btw.client` option to set a default client for new `btw_client()` calls, or use a `btw.md` project file for default chat client settings, like provider and model. We check the `client` argument, then the `btw.client` R option, and finally the `btw.md` project file (falling back to user-level `btw.md` if needed), using only the client definition from the first of these that is available.
`mode`	The mode to run the task in, which affects what is returned from this function. `"app"` and `"console"` modes launch interactive sessions, while `"client"` and `"tool"` modes return objects for programmatic use.
`tools`	Optional list or character vector of tools to allow the task to use when creating the skill. By default documentation tools are included to allow the task to help create package-based skills. You can include additional tools as needed. Because the task requires file tools to create skills with resources, tools for listing, reading and writing files are always included.

Value

When `mode` is `"app"` or `"console"`, this function launches an interactive session in the browser or the R console, respectively. The ellmer chat object with the conversation history is returned invisibly when the session ends.

When `mode` is `"client"`, this function returns the configured ellmer chat client object. When `mode` is `"tool"`, this function returns an ellmer tool object that can be used in other chat instances.

See Also

Other task and agent functions: `btw_task()`, `btw_task_create_btw_md()`, `btw_task_create_readme()`

Examples

```
withr::with_envvar(list(ANTHROPIC_API_KEY = "example"), {
  btw_task_create_skill(mode = "tool", client = "anthropic")
})
```

`btw_this` *Describe something for use by an LLM*

Description

A generic function used to describe an object for use by LLM.

Usage

```
btw_this(x, ...)
```

Arguments

`x`	The thing to describe.
`...`	Additional arguments passed down to underlying methods. Unused arguments are silently ignored.

Value

A character vector of lines describing the object.

See Also

Other btw formatting methods: `btw_this.character()`, `btw_this.data.frame()`, `btw_this.environment()`

Examples

```
btw_this(mtcars) # describe the mtcars dataset
btw_this(dplyr::mutate) # include function source
```

`btw_this.character` *Describe objects*

Description

Character strings in `btw_this()` are used as shortcuts to many underlying methods. `btw_this()` detects specific formats in the input string to determine which method to call, or by default it will try to evaluate the character string as R code and return the appropriate object description.

`btw_this()` knows about the following special character string formats:

- `"./path"`
  Any string starting with `./` is treated as a relative path. If the path is a file, we call `btw_tool_files_read()` and if the path is a directory we call `btw_tool_files_list()` on the path.
  - `btw_this("./data")` lists the files in `data/`.
  - `btw_this("./R/load_data.R")` reads the source of the `R/load_data.R` file.
- `"{pkgName}"` or `"@pkg pkgName"`
  A package name wrapped in braces, or using the `@pkg` command. Returns the list of help topics (`btw_tool_docs_package_help_topics()`) and, if it exists, the introductory vignette for the package (`btw_tool_docs_vignette()`).
  - `btw_this("{dplyr}")` or `btw_this("@pkg dplyr")` includes dplyr's introductory vignette.
  - `btw_this("{btw}")` returns only the package help index (because `btw` doesn't have an intro vignette, yet).
- `"?help_topic"` or `"@help topic"`
  When the string starts with `?` or `@help`, btw searches R's help topics using `btw_tool_docs_help_page()`. Supports multiple formats:
  - `btw_this("?dplyr::across")` or `btw_this("@help dplyr::across")`
  - `btw_this("@help dplyr across")` - space-separated format
  - `btw_this("@help across")` - searches all packages
- `"@news {{package_name}} {{search_term}}"`
  Include the release notes (NEWS) from the latest package release, e.g. `"@news dplyr"`, or that match a search term, e.g. `"@news dplyr join_by"`.
- `"@url {{url}}"`
  Include the contents of a web page at the specified URL as markdown, e.g. `"@url https://cran.r-project.org/doc/`
  Requires the **chromote** package to be installed.
- `"@git status"`, `"@git diff"`, `"@git log"`
  Git commands for viewing repository status, diffs, and commit history. Requires **gert** package and a git repository.
  - `btw_this("@git status")` - show working directory status
  - `btw_this("@git status staged")` - show only staged files
  - `btw_this("@git diff")` - show unstaged changes
  - `btw_this("@git diff HEAD")` - show staged changes

  - `btw_this("@git log")` - show recent commits (default 10)
  - `btw_this("@git log main 20")` - show 20 commits from main branch

- `"@issue #number"` or `"@pr #number"`
  Fetch a GitHub issue or pull request. Automatically detects the current repository, or you can specify `owner/repo#number` or `owner/repo number`. Requires **gh** package and GitHub authentication.
  - `btw_this("@issue #65")` - issue from current repo
  - `btw_this("@pr posit-dev/btw#64")` - PR from specific repo
  - `btw_this("@issue tidyverse/dplyr 1234")` - space-separated format
- `"@current_file"` or `"@current_selection"`
  When used in RStudio or Positron, or anywhere else that the **rstudioapi** is supported, `btw("@current_file")` includes the contents of the file currently open in the editor using `rstudioapi::getSourceEditorContext()`.
- `"@clipboard"`
  Includes the contents currently stored in your clipboard.
- `"@platform_info"`
  Includes information about the current platform, such as the R version, operating system, IDE or UI being used, as well as language, locale, timezone and current date.
- `"@attached_packages"`, `"@loaded_packages"`, `"@installed_packages"`
  Includes information about the attached, loaded, or installed packages in your R session, using `sessioninfo::package_info()`.
- `"@last_error"`
  Includes the message from the last error that occurred in your session. To reliably capture the last error, you need to enable `rlang::global_entrace()` in your session.
- `"@last_value"`
  Includes the `.Last.value`, i.e. the result of the last expression evaluated in your R console.

Usage

```
## S3 method for class 'character'
btw_this(x, ..., caller_env = parent.frame())
```

Arguments

`x`	A character string
`...`	Ignored.
`caller_env`	The caller environment.

Value

A character vector of lines describing the object.

See Also

Other btw formatting methods: `btw_this()`, `btw_this.data.frame()`, `btw_this.environment()`

Examples

```
mtcars[1:3, 1:4]
cat(btw_this("@last_value"))
```

`btw_this.data.frame` *Describe a data frame in plain text*

Description

Describe a data frame in plain text

Usage

```
## S3 method for class 'data.frame'
btw_this(
  x,
  ...,
  format = c("skim", "glimpse", "print", "json"),
  max_rows = 5,
  max_cols = 100,
  package = NULL
)

## S3 method for class 'tbl'
btw_this(
  x,
  ...,
  format = c("skim", "glimpse", "print", "json"),
  max_rows = 5,
  max_cols = 100,
  package = NULL
)
```

Arguments

`x`	A data frame or tibble.
`...`	Additional arguments are silently ignored.
`format`	One of `"skim"`, `"glimpse"`, `"print"`, or `"json"`.

- `"skim"` is the most information-dense format for describing the data. It uses and returns the same information as `skimr::skim()` but formatting as a JSON object that describes the dataset.
- To glimpse the data column-by-column, use `"glimpse"`. This is particularly helpful for getting a sense of data frame column names, types, and distributions, when pairings of entries in individual rows aren't particularly important.

- To just print out the data frame, use `print()`.
- To get a json representation of the data, use `"json"`. This is particularly helpful when the pairings among entries in specific rows are important to demonstrate.

`max_rows` The maximum number of rows to show in the data frame. Only applies when `format = "json"`.

`max_cols` The maximum number of columns to show in the data frame. Only applies when `format = "json"`.

`package` The name of the package that provides the data set. If not provided, `data_frame` must be loaded in the current environment, or may also be inferred from the name of the data frame, e.g. `"dplyr::storms"`.

Value

A character vector containing a representation of the data frame. Will error if the named data frame is not found in the environment.

Functions

- `btw_this(data.frame)`: Summarize a data frame.
- `btw_this(tbl)`: Summarize a `tbl`.

See Also

`btw_tool_env_describe_data_frame()`

Other btw formatting methods: `btw_this()`, `btw_this.character()`, `btw_this.environment()`

Examples

```
btw_this(mtcars)

btw_this(mtcars, format = "print")

btw_this(mtcars, format = "json")
```

---

btw_this.environment *Describe the contents of an environment*

Description

Describe the contents of an environment

Usage

```
## S3 method for class 'environment'
btw_this(x, ..., items = NULL)
```

Arguments

x	An environment.
...	Additional arguments are silently ignored.
items	Optional. A character vector of objects in the environment to describe.

Value

A string describing the environment contents with #> prefixing each object's printed representation.

See Also

btw_tool_env_describe_environment()

Other btw formatting methods: btw_this(), btw_this.character(), btw_this.data.frame()

Examples

```
env <- new.env()
env$cyl_6 <- mtcars[mtcars$cyl == 6, ]
env$gear_5 <- mtcars[mtcars$gear == 5, ]
btw_this(env)
```

---

btw_tool_agent_subagent

Tool: Subagent

Description

btw_tool_agent_subagent() is a btw tool that enables hierarchical agent workflows. When used by an LLM assistant (like btw_app(), btw_client(), or third-party tools like Claude Code), this tool allows the orchestrating agent to delegate complex tasks to specialized subagents, each with their own isolated conversation thread and tool access.

This function is primarily intended to be called by LLM assistants via tool use, not directly by end users.

How Subagents Work:

When an LLM calls this tool:

1. A new chat session is created (or an existing one is resumed)
2. The subagent receives the prompt and begins working with only the tools specified in the tools parameter
3. The subagent works independently, making tool calls until it completes the task
4. The function returns the subagent's final message text and a session_id
5. The orchestrating agent can resume the session later by providing the session_id

Each subagent maintains its own conversation context, separate from the orchestrating agent's context. Subagent sessions persist for the duration of the R session.

Tool Access:

The orchestrating agent must specify which tools the subagent can use via the `tools` parameter. The subagent is restricted to only these tools - it cannot access tools from the parent session. Tools can be specified by:

- **Specific tool names**: `c("btw_tool_files_read_text_file", "btw_tool_files_write_text_file")`
- **Tool groups**: `"files"` includes all file-related tools
- **NULL** (default): Uses the default tool set from options or `btw_tools()`

Configuration Options:

Subagent behavior can be configured via R options:

- `btw.subagent.client`: The ellmer::Chat client or `provider/model` string to use for subagents. If not set, falls back to `btw.client`, then to the default Anthropic client.
- `btw.subagent.tools_default`: Default tools to provide to subagents when the orchestrating agent doesn't specify tools via the `tools` parameter. If not set, falls back to `btw.tools`, then all btw tools from `btw_tools()`. This is a convenience option for setting reasonable defaults.
- `btw.subagent.tools_allowed`: An allowlist of tools that subagents are allowed to use at all. When set, any tools requested (either explicitly via the `tools` parameter or from defaults) will be filtered against this list. If disallowed tools are requested, an error is thrown. This provides a security boundary to restrict subagent capabilities. If not set, all `btw_tools()` are allowed.

These options follow the precedence: function argument > `btw.subagent.*` option > `btw.*` option > default value. The `tools_allowed` option acts as a filter on top of the resolved tools, regardless of their source.

Usage

```
btw_tool_agent_subagent(
  prompt,
  tools = NULL,
  session_id = NULL,
  `_intent` = ""
)
```

Arguments

`prompt`	Character string with the task description for the subagent. The subagent will work on this task using only the tools specified in `tools`. The subagent does not have access to the orchestrating agent's conversation history.
`tools`	Optional character vector of tool names or tool groups that the subagent is allowed to use. Can be specific tool names (e.g., `"btw_tool_files_read_text_file"`), tool group names (e.g., `"files"`), or `NULL` to use the default tools from `btw.subagent.tools_default`, `btw.tools`, or `btw_tools()`.
`session_id`	Optional character string with a session ID from a previous call. When provided, resumes the existing subagent conversation instead of starting a new one. Session IDs are returned in the result and have the format "adjective_noun" (e.g., "swift_falcon").

_intent Optional string describing the intent of the tool call. Added automatically by the ellmer framework when tools are called by LLMs.

Value

A `BtwSubagentResult` object (inherits from `BtwToolResult`) with:

- `value`: The final message text from the subagent
- `session_id`: The session identifier for resuming this conversation

See Also

`btw_tools()` for available tools and tool groups

Examples

```
# This tool is typically called by LLMs via tool use, not directly.
# The examples below show how to configure subagent behavior.

# Configure the client and default tools for subagents
withr::with_options(
  list(
    btw.subagent.client = "anthropic/claude-sonnet-4-20250514",
    btw.subagent.tools_default = "files"
  ),
  {
    getOption("btw.subagent.client")
  }
)

# Restrict subagents to only certain tools
withr::with_options(
  list(
    btw.subagent.tools_allowed = c("files", "docs"),
    btw.subagent.tools_default = "files"
  ),
  {
    getOption("btw.subagent.tools_allowed")
  }
)
```

---

btw_tool_cran_package *Tool: Describe a CRAN package*

Description

Describes a CRAN package using `pkgsearch::cran_package()`.

Usage

```
btw_tool_cran_package(package_name, `_intent` = "")
```

Arguments

package_name	The name of a package on CRAN.
_intent	An optional string describing the intent of the tool use. When the tool is used by an LLM, the model will use this argument to explain why it called the tool.

Value

An info sheet about the package.

See Also

Other cran tools: `btw_tool_cran_search()`

Examples

```
cli::cat_line(
  btw_this(pkgsearch::cran_package("anyflights"))
)
```

---

btw_tool_cran_search *Tool: Search for an R package on CRAN*

Description

Uses `pkgsearch::pkg_search()` to search for R packages on CRAN.

Usage

```
btw_tool_cran_search(
  query,
  format = c("short", "long"),
  n_results = NULL,
  `_intent` = ""
)
```

Arguments

query	Search query string. If this argument is missing or NULL, then the results of the last query are printed, in *short* and *long* formats, in turns for successive pkg_search() calls. If this argument is missing, then all other arguments are ignored.
format	Default formatting of the results. *short* only outputs the name and title of the packages, *long* also prints the author, last version, full description and URLs. Note that this only affects the default printing, and you can still inspect the full results, even if you specify *short* here.
n_results	Number of search results to include. Defaults to 10 for 'short' format and 5 for 'long' format.
_intent	An optional string describing the intent of the tool use. When the tool is used by an LLM, the model will use this argument to explain why it called the tool.

Value

A listing of packages matching the search term.

See Also

btw_tools()

Other cran tools: btw_tool_cran_package()

Examples

```
# Copy pkgsearch results to the clipboard for use in any LLM app
btw(
  pkgsearch::pkg_search("network visualization", size = 1),
  clipboard = FALSE
)
btw(
  pkgsearch::pkg_search("network visualization", format = "long", size = 1),
  clipboard = FALSE
)
```

---

btw_tool_docs_package_news

Tool: Package Release Notes

Description

Include release notes for a package, either the release notes for the most recent package release or release notes matching a search term.

Usage

```
btw_tool_docs_package_news(package_name, search_term = "", `_intent` = "")
```

Arguments

package_name	The name of the package as a string, e.g. "shiny".
search_term	A regular expression to search for in the NEWS entries. If empty, the release notes of the current installed version is included.
_intent	An optional string describing the intent of the tool use. When the tool is used by an LLM, the model will use this argument to explain why it called the tool.

Value

Returns the release notes for the currently installed version of the package, or the release notes matching the search term.

See Also

btw_tools()

Other docs tools: btw_tool_package_docs

Examples

```
# Copy release notes to the clipboard for use in any AI app
btw("@news dplyr", clipboard = FALSE)

btw("@news dplyr join_by", clipboard = FALSE)

if (interactive()) { # can be slow
  if (R.version$major == 4 && R.version$minor > "2.0") {
    # Search through R's release notes.
    # This should find a NEWS entry from R 4.2
    btw("@news R dynamic rd content", clipboard = FALSE)
  }
}

# Tool use by LLMs via ellmer or MCP ----
btw_tool_docs_package_news("dplyr")

btw_tool_docs_package_news("dplyr", "join_by")
```

`btw_tool_env_describe_data_frame` *Tool: Describe data frame*

Description

Tool: Describe data frame

Usage

```
btw_tool_env_describe_data_frame(
  data_frame,
  format = c("skim", "glimpse", "print", "json"),
  max_rows = 5,
  max_cols = 100,
  package = NULL,
  `_intent` = ""
)
```

Arguments

`data_frame`	The data frame to describe
`format`	One of `"skim"`, `"glimpse"`, `"print"`, or `"json"`. • `"skim"` is the most information-dense format for describing the data. It uses and returns the same information as `skimr::skim()` but formatting as a JSON object that describes the dataset. • To glimpse the data column-by-column, use `"glimpse"`. This is particularly helpful for getting a sense of data frame column names, types, and distributions, when pairings of entries in individual rows aren't particularly important. • To just print out the data frame, use `print()`. • To get a json representation of the data, use `"json"`. This is particularly helpful when the pairings among entries in specific rows are important to demonstrate.
`max_rows`	The maximum number of rows to show in the data frame. Only applies when `format = "json"`.
`max_cols`	The maximum number of columns to show in the data frame. Only applies when `format = "json"`.
`package`	The name of the package that provides the data set. If not provided, `data_frame` must be loaded in the current environment, or may also be inferred from the name of the data frame, e.g. `"dplyr::storms"`.
`_intent`	An optional string describing the intent of the tool use. When the tool is used by an LLM, the model will use this argument to explain why it called the tool.

Value

A character vector containing a representation of the data frame. Will error if the named data frame is not found in the environment.

See Also

`btw_this.data.frame()`, `btw_tools()`

Other env tools: `btw_tool_env_describe_environment()`

Examples

```
btw_tool_env_describe_data_frame(mtcars)
```

`btw_tool_env_describe_environment`

Tool: Describe an environment

Description

This tool can be used by the LLM to describe the contents of an R session, i.e. the data frames and other objects loaded into the global environment. This tool will only see variables that you've named and created in the global environment, it cannot reach into package namespaces, see which packages you have loaded, or access files on your computer.

Usage

```
btw_tool_env_describe_environment(items = NULL, `_intent` = "")
```

Arguments

`items`	Optional. A character vector of objects in the environment to describe.
`_intent`	An optional string describing the intent of the tool use. When the tool is used by an LLM, the model will use this argument to explain why it called the tool.

Value

A string describing the environment contents with #> prefixing each object's printed representation.

See Also

`btw_this.environment()`, `btw_tools()`

Other env tools: `btw_tool_env_describe_data_frame()`

Examples

```
my_cars <- mtcars[mtcars$mpg > 25, ]
btw_tool_env_describe_environment("my_cars")
```

`btw_tool_files_edit` *Tool: Edit a text file*

Description

Description:

This tool makes targeted, validated edits to a file using hashline references from `btw_tool_files_read()`. This approach is based on the technique described in "The Harness Problem" by Can Duruk, with additional customizations to improve the performance of the technique, in agentic coding session.

The core idea is that each line is annotated with a short content hash ("hashline") serving as a stable reference. When submitting an edit, the model includes hashline references for the target lines. The tool validates these hashes against the current file before applying changes, ensuring edits are applied to the intended lines. If a hash mismatch occurs, the edit is rejected and the model must re-read the file for fresh references.

How hashlines work:

`btw_tool_files_read()` annotates each line of the file with a short content hash in the format `line_number:hash|content`, e.g. `2:d6a|library(btw)`. The 3-character hash is derived from `rlang::hash()` over the line content (trimmed of leading/trailing whitespace and truncated to 80 characters).

When the model submits an edit, it references specific lines by their hashline, e.g. `"2:d6a"`, which ensures that the edit is applied precisely to the intended line. If the file has changed since the last read, the hashline references will not match the current file state, and the edit will be rejected with a hash mismatch error, prompting the model to re-read the file.

Edit actions:

`btw_tool_files_edit()` supports three types of edit actions, each using hashline references for validation:

- `"replace"`: Replace a single line. The model provides a `line` with a single hashline reference, e.g. `"2:d6a"`, and `content` to replace the line. The model can also remove the line by providing empty `content`.
- `"insert_after"`: Insert new lines after a reference line. The model provides a `line` with a hashline reference and `content` with the lines to insert after it. The model can also insert at the start of the file using `line = "0:000"`.
- `"replace_range"`: Replace a range of consecutive lines. The model provides a `line` with two hashline references indicating the start and end of the range, e.g. `"10:a3f,15:b2c"`, and `content` with the replacement lines.

Multiple edits in a single call are allowed and are applied together: all succeed or all fail. Edits must not have overlapping line ranges.

Response format:

After a successful edit, the response includes updated hashline references for the edited regions (plus 1 line of surrounding context). When edits change the total line count, the response includes shift hints that tell the model how to adjust any cached line numbers without re-reading the entire file (e.g. "update line numbers by +2").

Benefits and limitations:

The hashline approach provides strong validation for targeted edits, and for a single round of edits, it's generally more token efficient than the find-and-replace approach of `btw_tool_files_replace()` (which is also the approach used by many file-editing tools, e.g. `Edit` in Claude Code). With the hashline approach, the model can make very specific edits without needing to include large amounts of unchanged context around the edit, which is often required for find-and-replace to avoid unintended matches.

However, when repeatedly editing the same file, the hashline approach can cause the model to repeatedly re-read the file to get fresh hashline references. I've attempted to mitigate this by including updated references and shift hints (e.g. "update line numbers by +2") in the edit response, which allows the model to adjust the hashline references of previously read lines without re-reading the entire file. Models can also choose to re-read only specific sections of the file using the `line_start` and `line_end` parameters of `btw_tool_files_read()` to minimize token usage while refreshing references for the relevant lines.

However, not all models will make use of these features, and in practice, over longer editing sessions, the hashline approach may lead to more token usage due to the model repeatedly re-reading the file to get fresh references.

Usage

```
btw_tool_files_edit(path, edits, `_intent` = "")
```

Arguments

`path`	Path to the file to edit. The `path` must be in the current working directory.
`edits`	A list of edit operations. Each edit is a named list with `action`, `line`, and `content` fields. See **Edit actions** for details.
`_intent`	An optional string describing the intent of the tool use. When the tool is used by an LLM, the model will use this argument to explain why it called the tool.

Value

Returns a message confirming the edits were applied, including updated hashline references for the edited regions and shift hints when line numbers have changed.

See Also

`btw_tool_files_read()` for reading files with hashline annotations, `btw_tool_files_replace()` for find-and-replace edits that don't require hashline references.

Other files tools: `btw_tool_files_list()`, `btw_tool_files_patch()`, `btw_tool_files_read()`, `btw_tool_files_replace()`, `btw_tool_files_search()`, `btw_tool_files_write()`

`btw_tool_files_list` *Tool: List files*

Description

Tool: List files

Usage

```
btw_tool_files_list(
  path = NULL,
  type = c("any", "file", "directory"),
  regexp = "",
  `_intent` = ""
)
```

Arguments

`path`	Path to a directory or file for which to get information. The `path` must be in the current working directory. If `path` is a directory, we use `fs::dir_info()` to list information about files and directories in `path` (use `type` to pick only one or the other). If `path` is a file, we show information about that file.
`type`	File type(s) to return, one of `"any"` or `"file"` or `"directory"`.
`regexp`	A regular expression (e.g. `[.]csv$`) passed on to `grep()` to filter paths.
`_intent`	An optional string describing the intent of the tool use. When the tool is used by an LLM, the model will use this argument to explain why it called the tool.

Value

Returns a character table of file information.

See Also

Other files tools: `btw_tool_files_edit()`, `btw_tool_files_patch()`, `btw_tool_files_read()`, `btw_tool_files_replace()`, `btw_tool_files_search()`, `btw_tool_files_write()`

Examples

```
withr::with_tempdir({
  write.csv(mtcars, "mtcars.csv")

  btw_tool_files_list(type = "file")
})
```

`btw_tool_files_patch` *Tool: Apply a patch to files*

Description

Applies a structured diff-like patch envelope to one or more files. Unlike `btw_tool_files_edit()` (which requires hashline references from a prior read) or `btw_tool_files_replace()` (which requires exact strings), the patch tool uses context-matching hunks so models can produce coordinated edits across multiple files in a single tool call. A single patch envelope can add, update, delete, and move files atomically: either all operations succeed or none are applied.

Patch syntax:

A patch is a text envelope that begins with `*** Begin Patch` and ends with `*** End Patch`. Inside the envelope, each operation starts with a header:

```
** Begin Patch
** Add File: docs/example.md
Hello
World
** Update File: src/main.py
@@
context line
old line
new line
** Delete File: old.txt
** Update File: src/old.ts
** Move to: src/new.ts
@@
context line
export const oldName = 1
export const newName = 1
** End Patch
```

Headers:

- `*** Add File: <path>` – create a new file (must not exist).
- `*** Update File: <path>` – modify an existing file.
- `*** Delete File: <path>` – remove an existing file.
- `*** Move to: <path>` – sub-header inside an `Update File` block; renames the file to `<path>` after applying any hunks. Destination must not exist.

Hunk lines (inside `Update File`):

- `@` – hunk boundary; any trailing text on this line is informational and ignored.
- ` <text>` (space prefix) – context line that must match the file exactly.
- `-<text>` – line to delete; must match the file exactly at this position.
- `+<text>` – line to insert.

Every hunk must include at least one context or delete line to anchor the edit; pure-insert hunks are rejected. Use `*** Add File` for new files.

Add File body:

All body lines under `*** Add File` must start with `+`; the file content is the text after each `+`.

Usage

```
btw_tool_files_patch(patch, `_intent` = "")
```

Arguments

patch	The full patch text in the wire format described below. Must begin with `*** Begin Patch` and end with `*** End Patch`.
_intent	An optional string describing the intent of the tool use. When the tool is used by an LLM, the model will use this argument to explain why it called the tool.

Value

Returns a summary of the operations applied.

See Also

btw_tool_files_edit() for hashline-based targeted edits, btw_tool_files_replace() for exact find-and-replace edits.

Other files tools: btw_tool_files_edit(), btw_tool_files_list(), btw_tool_files_read(), btw_tool_files_replace(), btw_tool_files_search(), btw_tool_files_write()

---

btw_tool_files_read *Tool: Read a file*

Description

Description:

Reads the contents of a text file, optionally restricted to a line range. Each line is annotated with a hashline prefix (`line_number:hash|content`) that enables validated editing via btw_tool_files_edit().

Hashline annotations:

The hashline format prefixes each line with `line_number:hash|`, e.g. `2:f1a| return("world")`. The 3-character hash is a truncated rlang::hash() of the line content after trimming whitespace and truncating to 80 characters. These hashes are used by btw_tool_files_edit() to validate that the file hasn't changed between reading and editing. See btw_tool_files_edit() for details on the hashline approach and its benefits and limitations.

Hashline annotations are only included in the model-facing tool output. The display shown to users in btw_app() or shinychat::chat_ui() is always a clean code block.

Usage

```
btw_tool_files_read(path, line_start = 1, line_end = 1000, `_intent` = "")
```

Arguments

`path`	Path to a file for which to get information. The `path` must be in the current working directory.
`line_start`	Starting line to read, defaults to 1 (starting from the first line).
`line_end`	Ending line to read, defaults to 1000. Change only this value if you want to read more or fewer lines. Use in combination with `line_start` to read a specific line range of the file.
`_intent`	An optional string describing the intent of the tool use. When the tool is used by an LLM, the model will use this argument to explain why it called the tool.

Value

Returns lines with hashline annotations (see **Hashline annotations**).

See Also

`btw_tool_files_edit()` for making validated edits using hashline references, `btw_tool_files_replace()` for exact string find-and-replace, `btw_tool_files_write()` for writing entire files.

Other files tools: `btw_tool_files_edit()`, `btw_tool_files_list()`, `btw_tool_files_patch()`, `btw_tool_files_replace()`, `btw_tool_files_search()`, `btw_tool_files_write()`

Examples

```
withr::with_tempdir({
  write.csv(mtcars, "mtcars.csv")

  btw_tool_files_read("mtcars.csv", line_end = 5)
})
```

`btw_tool_files_replace`

Tool: Replace exact strings in a text file

Description

Description:

Finds and replaces exact string occurrences in a file. Because this tool operates on exact string matches, it's suited for simple renames, value updates, or repetitive text changes where the target string is unambiguous.

Replace vs Edit:

btw's two file-editing tools serve different use cases:

- `btw_tool_files_replace()` is best for exact text substitutions: renaming a variable, updating a URL, or changing a value. It does not require a prior `btw_tool_files_read()` call. By default, it requires the match to be unique to prevent unintended changes.

- btw_tool_files_edit() is best for structural, line-based edits: inserting new lines, deleting lines, replacing a range of lines, or making several edits at once. It requires hashline references from a prior btw_tool_files_read() call and validates them against the current file state.

Usage

```
btw_tool_files_replace(
  path,
  old_string,
  new_string,
  replace_all = FALSE,
  `_intent` = ""
)
```

Arguments

path	Path to the file to edit. The path must be in the current working directory.
old_string	The exact string to find in the file. Must match character-for-character, including whitespace and indentation. Must be unique unless replace_all is TRUE.
new_string	The replacement string. Must differ from old_string. Use an empty string ("") to delete the matched text.
replace_all	If TRUE, replace all occurrences of old_string. Defaults to FALSE, which requires exactly one occurrence.
_intent	An optional string describing the intent of the tool use. When the tool is used by an LLM, the model will use this argument to explain why it called the tool.

Value

Returns a message confirming the replacement was applied, including the number of occurrences replaced.

See Also

btw_tool_files_edit() for line-based structural edits using hashline references, btw_tool_files_read() for reading files.

Other files tools: btw_tool_files_edit(), btw_tool_files_list(), btw_tool_files_patch(), btw_tool_files_read(), btw_tool_files_search(), btw_tool_files_write()

---

btw_tool_files_search *Tool: Code Search in Project*

Description

Search through code files in the project directory for specific terms.

Usage

```
btw_tool_files_search(
  term,
  limit = 100,
  case_sensitive = TRUE,
  use_regex = FALSE,
  show_lines = FALSE,
  `_intent` = ""
)
```

Arguments

term	The term to search for in the code files.
limit	Maximum number of matching lines to return (between 1 and 1000, default 100).
case_sensitive	Whether the search should be case-sensitive (default is FALSE).
use_regex	Whether to interpret the search term as a regular expression (default is FALSE).
show_lines	Whether to show the matching lines in the results. Defaults to FALSE, which means only the file names and count of matching lines are returned.
_intent	An optional string describing the intent of the tool use. When the tool is used by an LLM, the model will use this argument to explain why it called the tool.

Details

Options:

You can configure which file extensions are included and which paths are excluded from code search by using two options:

- `btw.files_search.extensions`: A character vector of file extensions to search in (default includes R, Python, JavaScript, TypeScript, Markdown, SCSS, and CSS files).
- `btw.files_search.exclusions`: A character vector of gitignore-style patterns to exclude paths and directories from the search. The default value includes a set of common version control, IDE, and cache folders.

Alternatively, you can also set these options in your `btw.md` file under the `options` section, like this:

```
---
client:
  provider: anthropic
tools: [files_search]
options:
  files_search:
    extensions: ["R", "Rmd", "py", "qmd"]
    exclusions: ["DEFAULT", ".quarto/"]
---
```

Include "DEFAULT" in the exclusions option to use btw's default exclusions, which cover common directories like .git/, .vscode/.

If the **gert** package is installed and the project is a Git repository, the tool will also respect the .gitignore file and exclude any ignored paths, regardless of the btw.files_search.exclusions option.

Value

Returns a tool result with a data frame of search results, with columns for filename, size, last_modified, content and line.

See Also

Other files tools: btw_tool_files_edit(), btw_tool_files_list(), btw_tool_files_patch(), btw_tool_files_read(), btw_tool_files_replace(), btw_tool_files_write()

Examples

```
withr::with_tempdir({
  writeLines(state.name[1:25], "state_names_1.md")
  writeLines(state.name[26:50], "state_names_2.md")

  tools <- btw_tools("files_search")
  tools$btw_tool_files_search(
    term = "kentucky",
    case_sensitive = FALSE,
    show_lines = TRUE
  )
})
```

---

btw_tool_files_write *Tool: Write a text file*

Description

Tool: Write a text file

Usage

```
btw_tool_files_write(path, content, `_intent` = "")
```

Arguments

path	Path to the file to write. The path must be in the current working directory.
content	The text content to write to the file. This should be the complete content as the file will be overwritten.
_intent	An optional string describing the intent of the tool use. When the tool is used by an LLM, the model will use this argument to explain why it called the tool.

Value

Returns a message confirming the file was written.

See Also

Other files tools: btw_tool_files_edit(), btw_tool_files_list(), btw_tool_files_patch(), btw_tool_files_read(), btw_tool_files_replace(), btw_tool_files_search()

Examples

```
withr::with_tempdir({
  btw_tool_files_write("example.txt", "Hello\nWorld!")
  readLines("example.txt")
})
```

---

btw_tool_git_branch_checkout

Tool: Git Branch Checkout

Description

Allows an LLM to switch to a different git branch using gert::git_branch_checkout(), equivalent to `git checkout <branch>` in the terminal.

Usage

```
btw_tool_git_branch_checkout(branch, force = FALSE, `_intent` = "")
```

Arguments

branch	Name of branch to check out.
force	Whether to force checkout even with uncommitted changes. Defaults to FALSE.
_intent	An optional string describing the intent of the tool use. When the tool is used by an LLM, the model will use this argument to explain why it called the tool.

Value

Returns a confirmation message.

See Also

Other git tools: btw_tool_git_branch_create(), btw_tool_git_branch_list(), btw_tool_git_commit(), btw_tool_git_diff(), btw_tool_git_log(), btw_tool_git_status()

Examples

```
withr::with_tempdir({
  gert::git_init()
  gert::git_config_set("user.name", "R Example")
  gert::git_config_set("user.email", "ex@example.com")

  fs::file_touch("hello.md")

  gert::git_add("hello.md")
  gert::git_commit("Initial commit")

  gert::git_branch_create("feature-1")

  # LLM checks out an existing branch
  res <- btw_tool_git_branch_checkout(branch = "feature-1")

  # What the LLM sees
  cat(res@value)
})
```

---

btw_tool_git_branch_create

Tool: Git Branch Create

Description

Allows an LLM to create a new git branch using `gert::git_branch_create()`, equivalent to `git branch <branch>` in the terminal.

Usage

```
btw_tool_git_branch_create(
  branch,
  ref = "HEAD",
  checkout = TRUE,
  `_intent` = ""
)
```

Arguments

branch	Name of the new branch to create.
ref	Optional reference point for the new branch. Defaults to "HEAD".
checkout	Whether to check out the new branch after creation. Defaults to TRUE.
_intent	An optional string describing the intent of the tool use. When the tool is used by an LLM, the model will use this argument to explain why it called the tool.

Value

Returns a confirmation message.

See Also

Other git tools: btw_tool_git_branch_checkout(), btw_tool_git_branch_list(), btw_tool_git_commit(), btw_tool_git_diff(), btw_tool_git_log(), btw_tool_git_status()

Examples

```
withr::with_tempdir({
  gert::git_init()
  gert::git_config_set("user.name", "R Example")
  gert::git_config_set("user.email", "ex@example.com")

  fs::file_touch("hello.md")
  gert::git_add("hello.md")
  gert::git_commit("Initial commit")

  # LLM creates a new branch
  res <- btw_tool_git_branch_create(branch = "feature/new-analysis")

  # What the LLM sees
  cat(res@value)
})
```

---

btw_tool_git_branch_list

Tool: Git Branch List

Description

This tool allows an LLM to list git branches in the repository using gert::git_branch_list(), equivalent to `git branch` in the terminal.

Usage

```
btw_tool_git_branch_list(include = c("local", "remote", "all"), `_intent` = "")
```

Arguments

include	Once of "local" (default), "remote", or "all" to filter branches to local branches only, remote branches only, or all branches.
_intent	An optional string describing the intent of the tool use. When the tool is used by an LLM, the model will use this argument to explain why it called the tool.

Value

Returns a character table of branches.

See Also

Other git tools: btw_tool_git_branch_checkout(), btw_tool_git_branch_create(), btw_tool_git_commit(), btw_tool_git_diff(), btw_tool_git_log(), btw_tool_git_status()

Examples

```
withr::with_tempdir({
  gert::git_init()
  gert::git_config_set("user.name", "R Example")
  gert::git_config_set("user.email", "ex@example.com")

  fs::file_touch("hello.md")
  gert::git_add("hello.md")
  gert::git_commit("Initial commit")

  gert::git_branch_create("feature-1")
  gert::git_branch_create("feature-2")

  # What the LLM sees
  cat(btw_tool_git_branch_list()@value)
})
```

---

btw_tool_git_commit *Tool: Git Commit*

Description

This tool allows an LLM stage files and create a git commit. This tool uses a combination of gert::git_add() to stage files and gert::git_commit() to commit them, which is equivalent to `git add` and `git commit` in the terminal, respectively.

Usage

```
btw_tool_git_commit(message, files = NULL, `_intent` = "")
```

Arguments

message	A commit message describing the changes.
files	Optional character vector of file paths to stage and commit. Use "." to stage all changed files. If NULL, commits currently staged files.
_intent	An optional string describing the intent of the tool use. When the tool is used by an LLM, the model will use this argument to explain why it called the tool.

Value

Returns the commit SHA.

See Also

Other git tools: btw_tool_git_branch_checkout(), btw_tool_git_branch_create(), btw_tool_git_branch_list(), btw_tool_git_diff(), btw_tool_git_log(), btw_tool_git_status()

Examples

```
withr::with_tempdir({
  gert::git_init()
  gert::git_config_set("user.name", "R Example")
  gert::git_config_set("user.email", "ex@example.com")

  writeLines("hello, world", "hello.md")

  res <- btw_tool_git_commit("Initial commit", files = "hello.md")

  # What the LLM sees
  cat(res@value)
})
```

---

btw_tool_git_diff *Tool: Git Diff*

Description

This tool allows an LLM to run gert::git_diff_patch(), equivalent to `git diff` in the terminal, and to see the detailed changes made in a commit.

Usage

```
btw_tool_git_diff(ref = NULL, `_intent` = "")
```

Arguments

ref	a reference such as "HEAD", or a commit id, or NULL to the diff the working directory against the repository index.
_intent	An optional string describing the intent of the tool use. When the tool is used by an LLM, the model will use this argument to explain why it called the tool.

Value

Returns a diff patch as a formatted string.

See Also

Other git tools: btw_tool_git_branch_checkout(), btw_tool_git_branch_create(), btw_tool_git_branch_list(), btw_tool_git_commit(), btw_tool_git_log(), btw_tool_git_status()

Examples

```
withr::with_tempdir({
  gert::git_init()
  gert::git_config_set("user.name", "R Example")
  gert::git_config_set("user.email", "ex@example.com")

  writeLines("hello, world", "hello.md")
  gert::git_add("hello.md")
  gert::git_commit("Initial commit")

  writeLines("hello, universe", "hello.md")

  # What the LLM sees
  cat(btw_tool_git_diff()@value)
})
```

---

btw_tool_git_log *Tool: Git Log*

Description

This tool allows an LLM to run gert::git_log(), equivalent to `git log` in the terminal, and to see the commit history of a repository.

Usage

```
btw_tool_git_log(ref = "HEAD", max = 10, after = NULL, `_intent` = "")
```

Arguments

ref	Revision string with a branch/tag/commit value. Defaults to "HEAD".
max	Maximum number of commits to retrieve. Defaults to 10.
after	Optional date or timestamp: only include commits after this date.
_intent	An optional string describing the intent of the tool use. When the tool is used by an LLM, the model will use this argument to explain why it called the tool.

Value

Returns a character table of commit history.

See Also

Other git tools: btw_tool_git_branch_checkout(), btw_tool_git_branch_create(), btw_tool_git_branch_list(), btw_tool_git_commit(), btw_tool_git_diff(), btw_tool_git_status()

Examples

```
withr::with_tempdir({
  gert::git_init()
  gert::git_config_set("user.name", "R Example")
  gert::git_config_set("user.email", "ex@example.com")

  writeLines("hello, world", "hello.md")
  gert::git_add("hello.md")
  gert::git_commit("Initial commit")

  writeLines("hello, universe", "hello.md")
  gert::git_add("hello.md")
  gert::git_commit("Update hello.md")

  # What the LLM sees
  cat(btw_tool_git_log()@value)
})
```

---

btw_tool_git_status *Tool: Git Status*

Description

This tool allows the LLM to run gert::git_status(), equivalent to `git status` in the terminal, and to see the current status of the working directory.

Usage

```
btw_tool_git_status(
  include = c("both", "staged", "unstaged"),
  pathspec = NULL,
  `_intent` = ""
)
```

Arguments

include	One of "both", "staged", or "unstaged". Use "both" to show both staged and unstaged files (default).
pathspec	Optional character vector with paths to match.
_intent	An optional string describing the intent of the tool use. When the tool is used by an LLM, the model will use this argument to explain why it called the tool.

Value

Returns a character table of file statuses.

See Also

Other git tools: btw_tool_git_branch_checkout(), btw_tool_git_branch_create(), btw_tool_git_branch_list(), btw_tool_git_commit(), btw_tool_git_diff(), btw_tool_git_log()

Examples

```
withr::with_tempdir({
  gert::git_init()
  gert::git_config_set("user.name", "R Example")
  gert::git_config_set("user.email", "ex@example.com")

  writeLines("hello, world", "hello.md")

  # What the LLM sees
  cat(btw_tool_git_status()@value)
})
```

---

btw_tool_github *Tool: GitHub*

Description

Execute R code that calls the GitHub API using gh::gh().

This tool is designed such that models can write very limited R code to call gh::gh() and protections are inserted to prevent the model from calling unsafe or destructive actions via the API. The **Endpoint Validation** section below describes how API endpoints are validated to ensure safety.

While this tool *can* execute R code, the code is evaluated in an environment where only a limited set of functions and variables are available. In particular, only the `gh()` and `gh_whoami()` functions from the gh package are available, along with `owner` and `repo` variables that are pre-defined to point to the current repository (if detected). This allows models to focus on writing GitHub API calls without needing to load packages or manage authentication.

Endpoint Validation:

This tool uses endpoint validation to ensure only safe GitHub API operations are performed. By default, most read operations and low-risk write operations (like creating issues or PRs) are allowed, while dangerous operations (like merging PRs or deleting repositories) are blocked.

To customize which endpoints are allowed or blocked, use the `btw.github.allow` and `btw.github.block` options:

```
# Allow a specific endpoint
options(btw.github.allow = c(
  getOption("btw.github.allow"),
```

```
  "GET /repos/*/*/topics"
))

# Block a specific endpoint
options(btw.github.block = c(
  getOption("btw.github.block"),
  "GET /repos/*/*/branches"
))
```

You can also set these options in your btw.md file under the `options` field:

```
tools: github
options:
  github:
    allow:
      - "PATCH /repos/*/*/pulls/*" # Allow converting PRs to/from draft
      - "POST /repos/*/*/git/refs" # Allow creating branches
    block:
      - "DELETE /repos/**" # Block any delete action under /repos
```

The precedence order for rules is:

1. User block rules (checked first, highest priority)
2. User allow rules
3. Built-in block rules
4. Built-in allow rules
5. Default: reject (if no rules match)

Additional Examples:

```
# Get an issue
btw_tool_github(
  code = 'gh("/repos/{owner}/{repo}/issues/123", owner = owner, repo = repo)'
)

# Create an issue
btw_tool_github(code = r"(
  gh(
    "POST /repos/{owner}/{repo}/issues",
    title = \"Bug report\",
    body = \"Description of bug\",
    owner = owner,
    repo = repo
  )
)")

# Target a different repository
btw_tool_github(code = 'gh("/repos/tidyverse/dplyr/issues/123")')
```

Usage

```
btw_tool_github(code, fields = "default", `_intent` = "")
```

Arguments

code	R code that calls gh() or gh_whoami(). The code will be evaluated in an environment where owner and repo variables are predefined (defaulting to the current repository if detected). The gh() function is available without needing to load the gh package.
fields	Optional character vector of GitHub API response fields to retain. If provided, only these fields will be included in the result. Defaults to a curated set of commonly used fields.
_intent	An optional string describing the intent of the tool use. When the tool is used by an LLM, the model will use this argument to explain why it called the tool.

Value

A btw_tool_result containing the result of the GitHub API call.

Examples

```
# This tool requires the gh package and authentication to GitHub.
# See additional examples in the documentation above.
```

---

btw_tool_ide_read_current_editor

Tool: Read current file

Description

Reads the current file using the **rstudioapi**, which works in RStudio, Positron and VS Code (with the vscode-r extension).

Usage

```
btw_tool_ide_read_current_editor(
  selection = TRUE,
  consent = FALSE,
  `_intent` = ""
)
```

Arguments

selection	Should only the selected text be included? If no text is selected, the full file contents are returned.
consent	Boolean indicating whether the user has consented to reading the current file. The tool definition includes language to induce LLMs to confirm with the user before calling the tool. Not all models will follow these instructions. Users can also include the string @current_file to induce the tool.
_intent	An optional string describing the intent of the tool use. When the tool is used by an LLM, the model will use this argument to explain why it called the tool.

Value

Returns the contents of the current editor.

Examples

```
btw_tool_ide_read_current_editor(consent = TRUE)
```

`btw_tool_package_docs` *Tool: Describe R package documentation*

Description

These functions describe package documentation in plain text.

Additional Examples:

Show a list of available vignettes in the `dplyr` package:

```
btw_tool_docs_available_vignettes("dplyr")
```

Get the introductory vignette for the `dplyr` package:

```
btw_tool_docs_vignette("dplyr")
```

Get a specific vignette, such as the programming vignette for the `dplyr` package:

```
btw_tool_docs_vignette("dplyr", "programming")
```

Usage

```
btw_tool_docs_package_help_topics(package_name, `_intent` = "")

btw_tool_docs_help_page(topic, package_name = "", `_intent` = "")

btw_tool_docs_available_vignettes(package_name, `_intent` = "")

btw_tool_docs_vignette(package_name, vignette = package_name, `_intent` = "")
```

Arguments

`package_name`	The name of the package as a string, e.g. `"shiny"`.
`_intent`	An optional string describing the intent of the tool use. When the tool is used by an LLM, the model will use this argument to explain why it called the tool.
`topic`	The `topic_id` or `alias` of the help page, e.g. `"withProgress"` or `"incProgress"`. Find `topic_ids` or `aliases` using `get_package_help()`.
`vignette`	The name (or index) of the vignette to retrieve. Defaults to the "intro" vignette to the package (by the same rules as pkgdown.)

Value

- `btw_tool_docs_package_help_topics()` returns the `topic_id`, `title`, and `aliases` fields for every topic in a package's documentation as a json-formatted string.
- `btw_tool_docs_help_page()` returns the help-page for a package topic as a string.

See Also

`btw_tools()`

Other docs tools: `btw_tool_docs_package_news()`

Examples

```
btw_tool_docs_package_help_topics("btw")

btw_tool_docs_help_page("btw", "btw")
```

---

btw_tool_pkg_check *Tool: Run R CMD check on a package*

Description

Run R CMD check on a package using `devtools::check()`. This performs comprehensive checks on the package structure, code, and documentation.

Usage

```
btw_tool_pkg_check(pkg = ".", `_intent` = "")
```

Arguments

`pkg`	Path to package directory. Defaults to '.'. Must be within current working directory.
`_intent`	An optional string describing the intent of the tool use. When the tool is used by an LLM, the model will use this argument to explain why it called the tool.

Details

The check runs with `remote = TRUE`, `cran = TRUE`, `manual = FALSE`, and `error_on = "never"` to provide comprehensive feedback without failing.

Value

The output from `devtools::check()`.

See Also

btw_tools()

Other pkg tools: btw_tool_pkg_coverage(), btw_tool_pkg_document(), btw_tool_pkg_load_all(), btw_tool_pkg_test()

`btw_tool_pkg_coverage` *Tool: Compute package test coverage*

Description

Compute test coverage for an R package using covr::package_coverage(). Returns either a file-level summary for the entire package or line-level details for a specific file.

Usage

```
btw_tool_pkg_coverage(pkg = ".", filename = NULL, `_intent` = "")
```

Arguments

pkg	Path to package directory. Defaults to ".". Must be within current working directory.
filename	Optional filename to filter coverage results. If NULL (default), returns file-level summary for entire package. If provided, returns line-level results for the specified file.
_intent	An optional string describing the intent of the tool use. When the tool is used by an LLM, the model will use this argument to explain why it called the tool.

Value

A data frame with different structures depending on filename:

- When filename = NULL: Returns file-level summary with columns filename and coverage (percentage).
- When filename is specified: Returns line-level details with columns filename, functions, line_start, line_end, is_covered, and med_hits.

See Also

btw_tools()

Other pkg tools: btw_tool_pkg_check(), btw_tool_pkg_document(), btw_tool_pkg_load_all(), btw_tool_pkg_test()

btw_tool_pkg_document *Tool: Generate package documentation*

Description

Generate package documentation using devtools::document(). This runs **roxygen2** on the package to create/update man pages and NAMESPACE.

Usage

```
btw_tool_pkg_document(pkg = ".", `_intent` = "")
```

Arguments

pkg	Path to package directory. Defaults to ".". Must be within current working directory.
_intent	An optional string describing the intent of the tool use. When the tool is used by an LLM, the model will use this argument to explain why it called the tool.

Value

The output from devtools::document().

See Also

btw_tools()

Other pkg tools: btw_tool_pkg_check(), btw_tool_pkg_coverage(), btw_tool_pkg_load_all(), btw_tool_pkg_test()

btw_tool_pkg_load_all *Tool: Load package code*

Description

Load package code using pkgload::load_all() in a separate R process via callr::r(). This verifies that the package code loads without syntax errors and triggers recompilation of any compiled code (C, C++, etc.).

Usage

```
btw_tool_pkg_load_all(pkg = ".", `_intent` = "")
```

Arguments

pkg	Path to package directory. Defaults to '.'. Must be within current working directory.
_intent	An optional string describing the intent of the tool use. When the tool is used by an LLM, the model will use this argument to explain why it called the tool.

Details

Important: This tool runs `load_all()` in an isolated R process and does NOT load the package code into your current R session. If you need to load the package code in your current session for interactive use, use the run R code tool to call `pkgload::load_all()` directly.

Value

The output from `pkgload::load_all()`.

See Also

`btw_tools()`

Other pkg tools: `btw_tool_pkg_check()`, `btw_tool_pkg_coverage()`, `btw_tool_pkg_document()`, `btw_tool_pkg_test()`

---

btw_tool_pkg_test *Tool: Run package tests*

Description

Run package tests using `devtools::test()`. Optionally filter tests by name pattern.

Usage

```
btw_tool_pkg_test(pkg = ".", filter = NULL, `_intent` = "")
```

Arguments

pkg	Path to package directory. Defaults to '.'. Must be within current working directory.
filter	Optional regex to filter test files. Example: 'helper' matches 'test-helper.R'.
_intent	An optional string describing the intent of the tool use. When the tool is used by an LLM, the model will use this argument to explain why it called the tool.

Value

The output from `devtools::test()`.

See Also

btw_tools()

Other pkg tools: btw_tool_pkg_check(), btw_tool_pkg_coverage(), btw_tool_pkg_document(), btw_tool_pkg_load_all()

`btw_tool_run_r` *Tool: Run R code*

Description

[Experimental] This tool runs R code and returns results as a list of ellmer::Content() objects. It captures text output, plots, messages, warnings, and errors. Code execution stops on the first error, returning all results up to that point.

Usage

```
btw_tool_run_r(code, `_intent` = "")
```

Arguments

code	A character string containing R code to run.
_intent	An optional string describing the intent of the tool use. When the tool is used by an LLM, the model will use this argument to explain why it called the tool.

Details

Configuration Options:

The behavior of the btw_tool_run_r tool can be customized using the following R options:

- btw.run_r.graphics_device: A function that creates a graphics device used for rendering plots. By default, it uses ragg::agg_png() if the ragg package is installed, otherwise it falls back to grDevices::png().
- btw.run_r.plot_aspect_ratio: Aspect ratio for plots created during code execution. Can be a character string of the form "w:h" (e.g., "16:9") or a numeric value representing width/height (e.g., 16/9). Default is "3:2".
- btw.run_r.plot_size: Integer pixel size for the longest side of plots. Default is 768L. This image size was selected to match OpenAI's image resizing rules, where images are resized such that the largest size is 768px. Another common choice is 512px. Larger images may be used but will result in increased token sizes.
- btw.run_r.enabled: Logical flag to enable or disable the tool globally.

These values can be set using options() in your R session or .Rprofile or in a btw.md file under the options section.

```
---
options:
 run_r:
   enabled: true
   plot_aspect_ratio: "16:9"
   plot_size: 512
---
```

Value

A list of ellmer Content objects:

- `ContentText`: visible return values and text output
- `ContentMessage`: messages from `message()`
- `ContentWarning`: warnings from `warning()`
- `ContentError`: errors from `stop()`
- `ContentImageInline`: plots created during execution

Security Considerations

Executing arbitrary R code can pose significant security risks, especially in shared or multi-user environments. Furthermore, neither **shinychat** (as of v0.4.0) or nor **ellmer** (as of v0.4.0) provide a mechanism to review and reject the code before execution. Even more, the code is executed in the global environment and does not have any sandboxing or R code limitations applied.

It is your responsibility to ensure that you are taking appropriate measures to reduce the risk of the LLM writing arbitrary code. Most often, this means not prompting the model to take large or potentially destructive actions. At this time, we do not recommend that you enable this tool in a publicly- available environment without strong safeguards in place.

That said, this tool is very powerful and can greatly enhance the capabilities of your btw chatbots. Please use it responsibly! If you'd like to enable the tool, please read the instructions below.

Enabling this tool

This tool is not enabled by default in `btw_tools()`, `btw_app()` or `btw_client()`. To enable the function, you have a few options:

1. Set the `btw.run_r.enabled` option to `TRUE` in your R session, or in your `.Rprofile` file to enable it globally.
2. Set the `BTW_RUN_R_ENABLED` environment variable to `true` in your `.Renviron` file or your system environment.
3. Explicitly include the tool when calling `btw_tools("run")` (unless the above options disable it).

In your btw.md file, you can explicitly enable the tool by naming it in the tools option

```
---
tools:
  - run_r
---
```

or you can enable the tool by setting the `btw.run_r.enabled` option from the `options` list in `btw.md` (this approach is useful if you've globally disabled the tool but want to enable it for a specific btw chat):

```
---
options:
  run_r:
    enabled: true
---
```

See Also

`btw_tools()`

Examples

```
## Not run:
# Simple calculation
btw_tool_run_r("2 + 2")

# Code with plot
btw_tool_run_r("hist(rnorm(100))")

# Code with warning
btw_tool_run_r("mean(c(1, 2, NA))")

## End(Not run)
```

`btw_tool_sessioninfo_is_package_installed`

Tool: Check if a package is installed

Description

Checks if a package is installed in the current session. If the package is installed, it returns the version number. If not, it suggests packages with similar names to help the LLM resolve typos.

Usage

```
btw_tool_sessioninfo_is_package_installed(package_name, `_intent` = "")
```

Arguments

`package_name`	The name of the package.
`_intent`	An optional string describing the intent of the tool use. When the tool is used by an LLM, the model will use this argument to explain why it called the tool.

Value

A message indicating whether the package is installed and its version, or an error indicating that the package is not installed.

See Also

`btw_tools()`

Other sessioninfo tools: `btw_tool_sessioninfo_package()`, `btw_tool_sessioninfo_platform()`

Examples

```
btw_tool_sessioninfo_is_package_installed("dplyr")@value

tryCatch(
  btw_tool_sessioninfo_is_package_installed("dplry"),
  error = function(err) {
    cat(conditionMessage(err))
  }
)
```

---

btw_tool_sessioninfo_package

Tool: Gather information about a package or currently loaded packages

---

Description

Uses `sessioninfo::package_info()` to provide information about the loaded, attached, or installed packages. The primary use case is to verify that a package is installed; check the version number of a specific packages; or determine which packages are already in use in a session.

Usage

```
btw_tool_sessioninfo_package(
  packages = "attached",
  dependencies = "",
  `_intent` = ""
)
```

Arguments

`packages`	Which packages to show, or `"loaded"` to show all loaded packages, `"attached"` to show all attached packages, or `"installed"` to show all installed packages.
`dependencies`	Whether to include the dependencies when listing package information.
`_intent`	An optional string describing the intent of the tool use. When the tool is used by an LLM, the model will use this argument to explain why it called the tool.

Value

Returns a string describing the selected packages.

See Also

`btw_tools()`, `btw_tool_sessioninfo_platform()`

Other sessioninfo tools: `btw_tool_sessioninfo_is_package_installed()`, `btw_tool_sessioninfo_platform()`

Examples

```
btw_tool_sessioninfo_package("btw")
```

---

btw_tool_sessioninfo_platform

Tool: Describe user's platform

Description

Describes the R version, operating system, and language and locale settings for the user's system. When using `btw_client()` or `btw_app()`, this information is automatically included in the system prompt.

Usage

```
btw_tool_sessioninfo_platform(`_intent` = "")
```

Arguments

`_intent`	An optional string describing the intent of the tool use. When the tool is used by an LLM, the model will use this argument to explain why it called the tool.

Value

Returns a string describing the user's platform.

See Also

`btw_tools()`

Other sessioninfo tools: `btw_tool_sessioninfo_is_package_installed()`, `btw_tool_sessioninfo_package()`

Examples

```
btw_tool_sessioninfo_platform()
```

`btw_tool_skill` *Tool: Load a skill*

Description

Load a skill's specialized instructions and list its bundled resources.

Skills are modular capabilities that extend Claude's functionality with specialized knowledge, workflows, and tools. Each skill is a directory containing a `SKILL.md` file with instructions and optional bundled resources (scripts, references, assets).

When `btw_tool_skill` is included in the chat client's tools, btw automatically injects information about available skills into the system prompt so the model knows which skills are available. If the skill tool is added after client creation, the model can call `btw_tool_skill("")` (empty name) to get the current skill listing.

Skills are discovered from the following locations, in increasing order of priority (later sources override earlier ones when skill names conflict):

1. Skills bundled with the btw package itself
2. Skills from currently **attached** R packages — any package with an `inst/skills/` directory that is loaded via `library()` or `require()`
3. User-level skills (`~/.btw/skills`, `~/.config/btw/skills`, `tools::R_user_dir("btw")/skills`). For backwards compatibility, the legacy `tools::R_user_dir("btw", "config")/skills` path used by briefly by btw 1.2.0 is also included at lower priority.
4. Project-level skills (`.btw/skills/` or `.agents/skills/`)

The default user-level and project-level directories can be replaced by setting the `btw.skills.paths` R option or the `BTW_SKILLS_PATHS` environment variable. When set, the value **entirely replaces** all user-level and project-level directories (items 3 and 4 above). Package-bundled skills and skills from attached packages (items 1 and 2) are always included regardless of this setting. The R option takes precedence over the environment variable. Multiple paths can be provided as a character vector (e.g. `options(btw.skills.paths = c("/path/a", "/path/b"))`) or as a single path-separator-delimited string (`:` on Unix/Mac, `;` on Windows, which is the only form supported by environment variables). Non-existent paths are silently skipped.

Resolution timing: options and environment variables are read at **tool-registration time** (i.e. when `btw_tools()` or `btw_client()` is called). The resolved paths are captured in the tool's closure so that they remain correct even if the options are later modified or go out of scope (for example, when `btw_client()` restores options after returning). If you need different directories for a new session, create a new client.

Usage

```
btw_tool_skill(name, `_intent` = "")
```

Arguments

`name`	The name of the skill to load, or `""` to list all available skills.
`_intent`	An optional string describing the intent of the tool use. When the tool is used by an LLM, the model will use this argument to explain why it called the tool.

Value

A `btw_tool_result` containing the skill instructions and a listing of bundled resources with their paths.

See Also

Other skills: `btw_skill_install_github()`, `btw_skill_install_package()`, `btw_skill_install_project()`

`btw_tool_web_read_url` *Tool: Read a Web Page as Markdown*

Description

Tool: Read a Web Page as Markdown

Usage

```
btw_tool_web_read_url(url, `_intent` = "")
```

Arguments

`url`	The URL of the web page to read.
`_intent`	An optional string describing the intent of the tool use. When the tool is used by an LLM, the model will use this argument to explain why it called the tool.

Details

You can control the maximum time to wait for the page to load by setting the `btw.max_wait_for_page_load_s` option globally in your R session.

Value

Returns a `BtwWebPageResult` object that inherits from ellmer::ContentToolResult containing the markdown content of the web page.

Examples

```
btw_tool_web_read_url("https://www.r-project.org/")
btw_tool_web_read_url(
  "https://posit.co/blog/easy-tool-calls-with-ellmer-and-chatlas/"
)
```

`btw_tools` *Tools: Register tools from btw*

Description

The `btw_tools()` function provides a list of tools that can be registered with an ellmer chat via `chat$register_tools()` that allow the chat to interface with your computational environment. Chats returned by this function have access to the tools:

Group: agent:

Name	Description
`btw_tool_agent_subagent()`	Delegate a task to a specialized assistant that can work independently with its own conversation thread.

Group: cran:

Name	Description
`btw_tool_cran_package()`	Describe a CRAN package.
`btw_tool_cran_search()`	Search for an R package on CRAN.

Group: docs:

Name	Description
`btw_tool_docs_available_vignettes()`	List available vignettes for an R package.
`btw_tool_docs_help_page()`	Get help page from package.
`btw_tool_docs_package_help_topics()`	Get available help topics for an R package.
`btw_tool_docs_package_news()`	Read the release notes (NEWS) for a package.
`btw_tool_docs_vignette()`	Get a package vignette in plain text.

Group: env:

Name	Description
`btw_tool_env_describe_data_frame()`	Show the data frame or table or get information about the structure of a data fra
`btw_tool_env_describe_environment()`	List and describe items in the R session's global environment.

Group: files:

Name	Description
`btw_tool_files_edit()`	Edit a text file using hashline references for precise, targeted modifications.
`btw_tool_files_list()`	List files or directories in the project.
`btw_tool_files_patch()`	Apply a patch envelope that adds, updates, deletes, or moves files atomically.
`btw_tool_files_read()`	Read the contents of a text file.

btw_tool_files_replace()	Find and replace exact string occurrences in a text file.
btw_tool_files_search()	Search code files in the project.
btw_tool_files_write()	Write content to a text file.

Group: git:

Name	Description
btw_tool_git_branch_checkout()	Switch to a different git branch.
btw_tool_git_branch_create()	Create a new git branch.
btw_tool_git_branch_list()	List git branches in the repository.
btw_tool_git_commit()	Stage files and create a git commit.
btw_tool_git_diff()	View changes in the working directory or a commit.
btw_tool_git_log()	Show the commit history for a repository.
btw_tool_git_status()	Show the status of the git working directory.

Group: github:

Name	Description
btw_tool_github()	Execute R code that calls the GitHub API using gh().

Group: ide:

Name	Description
btw_tool_ide_read_current_editor()	Read the contents of the editor that is currently open in the user's IDE.

Group: pkg:

Name	Description
btw_tool_pkg_check()	Run comprehensive package checks.
btw_tool_pkg_coverage()	Compute test coverage for an R package.
btw_tool_pkg_document()	Generate package documentation.
btw_tool_pkg_load_all()	Load package code to verify it loads correctly.
btw_tool_pkg_test()	Run testthat tests for an R package.

Group: run:

Name	Description
btw_tool_run_r()	Run R code.

Group: sessioninfo:

Name	Description
btw_tool_sessioninfo_is_package_installed()	Check if a package is installed in the current session.

btw_tool_sessioninfo_package()	Verify that a specific package is installed, or find out which packages
btw_tool_sessioninfo_platform()	Describes the R version, operating system, language and locale setting

Group: skills:

Name	Description
btw_tool_skill()	Load a skill's specialized instructions and list its bundled resources.

Group: web:

Name	Description
btw_tool_web_read_url()	Read a web page and convert it to Markdown format.

Usage

```
btw_tools(...)
```

Arguments

...	Optional names of tools or tool groups to include when registering tools. By default all btw tools are included. For example, use "docs" to include only the documentation related tools, or "env", "docs", "session" for the collection of environment, documentation and session tools, and so on. The names provided can be: 1. The name of a tool, such as "btw_tool_env_describe_data_frame". 2. The name of a tool group, such as "env", which will include all tools in that group. 3. The tool name without the btw_tool_ prefix, such as "env_describe_data_frame".

Value

Registers the tools with chat, updating the chat object in place. The chat input is returned invisibly.

Examples

```
# requires an ANTHROPIC_API_KEY
ch <- ellmer::chat_anthropic()

# register all of the available tools
ch1 <- ch$clone()
ch1$register_tools(btw_tools())

# or register only the tools related to fetching documentation
ch2 <- ch$clone()
ch2$register_tools(btw_tools("docs"))
```

```
# ensure that the current tools persist
ch3 <- ch$clone()
ch3$register_tools(c(ch3$get_tools(), btw_tools()))
```

`install_btw_cli` *Install the btw CLI*

Description

Installs the `btw` CLI launcher using `Rapp::install_pkg_cli_apps()`. Rapp is required to build and install the CLI. See `Rapp::install_pkg_cli_apps()` for details on where the launcher is installed and how to manage it.

Usage

```
install_btw_cli(destdir = NULL, ...)
```

Arguments

`destdir`	Directory where the CLI launcher will be installed. If `NULL`, the default location used by `Rapp::install_pkg_cli_apps()` is used.
`...`	Arguments passed on to `Rapp::install_pkg_cli_apps` `lib.loc` Additional library paths forwarded to `base::system.file()` while locating package scripts. Discovery happens at install time; written launchers embed absolute script paths. `overwrite` Whether to replace an existing executable. `TRUE` always overwrites, `FALSE` never overwrites non-Rapp executables, and `NA` (the default) prompts interactively and otherwise skips.

Details

After installing the CLI, you will be offered the option to install the `r-btw-cli` skill, which helps AI coding assistants discover and use the `btw` CLI. If you decline or are in a non-interactive session, the skill instructions are copied to the clipboard (or printed) so you can add them to your `CLAUDE.md`, `AGENTS.md`, or other context file manually.

Value

The result of `Rapp::install_pkg_cli_apps()`, invisibly.

mcp *Start a Model Context Protocol server with btw tools*

Description

`btw_mcp_server()` starts an MCP server with tools from `btw_tools()`, which can provide MCP clients like Claude Desktop or Claude Code with additional context. The function will block the R process it's called in and isn't intended for interactive use.

To give the MCP server access to a specific R session, run `btw_mcp_session()` in that session. If there are no sessions configured, the server will run the tools in its own session, meaning that e.g. the `btw_tools(tools = "env")` tools will describe R objects in *that* R environment.

Usage

```
btw_mcp_server(tools = NULL)

btw_mcp_session()
```

Arguments

`tools`	A list of `ellmer::tool()`s to use in the MCP server, defaults to the tools provided by `btw_tools()`. Use `btw_tools()` to subset to specific list of **btw** tools that can be augmented with additional tools. Alternatively, you can pass a path to an R script that returns a list of tools as supported by `mcptools::mcp_server()`.

Value

Returns the result of `mcptools::mcp_server()` or `mcptools::mcp_session()`.

Choosing Tool Groups

When using btw with a coding agent that already has built-in tools for file operations, code execution, or other tasks, you may want to select a subset of btw's tools to avoid overlap. A recommended lightweight configuration for R package development is:

```
btw_mcp_server(btw_tools("docs", "pkg"))
```

This gives the agent access to R documentation (help pages, vignettes, news) and package development tools (testing, checking, documenting) without duplicating file or code execution capabilities the agent may already have.

Depending on your workflow, you may also want to include:

- `"env"`: Tools to inspect R objects and data frames in the session
- `"sessioninfo"`: Tools to check installed packages and platform details
- `"cran"`: Tools to search for and describe CRAN packages

```
btw_mcp_server(list("docs", "pkg", "env", "sessioninfo", "cran"))
```

See `btw_tools()` for a complete list of available tool groups and their contents.

Configuration

To configure this server with MCP clients, use the command `Rscript` and the args `-e "btw::btw_mcp_server()"`. We recommend customizing the tool set as described above to avoid overlap with your client's built-in capabilities and to choose the tools that make the most sense for your workflow.

For Claude Desktop's configuration format:

```
{
  "mcpServers": {
    "r-btw": {
      "command": "Rscript",
    "args": ["-e", "btw::btw_mcp_server(list('docs', 'pkg', 'env', 'sessioninfo', 'cran'))"]
    }
  }
}
```

For Claude Code:

```
claude mcp add -s "user" r-btw -- Rscript -e "btw::btw_mcp_server(list('docs', 'pkg', 'env', 'sessionin
```

For Positron or VS Code, add the following to `.vscode/mcp.json` (for workspace configuration) or to your user profile's `mcp.json` (for global configuration, accessible via Command Palette > **MCP: Open User Configuration**):

```
{
  "servers": {
    "r-btw": {
      "type": "stdio",
      "command": "Rscript",
    "args": ["-e", "btw::btw_mcp_server(list('docs', 'pkg', 'env', 'sessioninfo', 'cran'))"]
    }
  }
}
```

Alternatively, run the **MCP: Add Server** command from the Command Palette, choose **stdio**, then choose **Workspace** or **Global** to add the server configuration interactively.

For Continue, include the following in your config file:

```
"experimental": {
  "modelContextProtocolServers": [
    {
      "transport": {
        "name": "r-btw",
        "type": "stdio",
        "command": "Rscript",
        "args": [
          "-e",
        "btw::btw_mcp_server(list('docs', 'pkg', 'env', 'sessioninfo', 'cran'))"
```

```
        ]
      }
    }
  ]
}
```

Additional Examples

`btw_mcp_server()` should only be run non-interactively, as it will block the current R process once called.

To start a server with btw tools:

```
btw_mcp_server()
```

Or to only do so with a subset of btw's tools, e.g. those that fetch package documentation:

```
btw_mcp_server(btw_tools("docs"))

# alternatively a bare list
btw_mcp_server(list("docs"))

# or a list of btw tools and custom tools
btw_mcp_server(list("docs", my_custom_tool))
```

To allow the server to access variables in specific sessions, call `btw_mcp_session()` in that session:

```
btw_mcp_session()
```

See Also

These functions use `mcptools::mcp_server()` and `mcptools::mcp_session()` under the hood. To configure arbitrary tools with an MCP client, see the documentation of those functions.

Examples

```
# btw_mcp_server() and btw_mcp_session() are only intended to be run in
# non-interactive R sessions, e.g. when started by an MCP client like
# Claude Desktop or Claude Code. Therefore, we don't run these functions
# in examples.

# See above for more details and examples.
```

use_btw_md *Create or edit a btw.md context file*

Description

Create or edit a `btw.md` or `AGENTS.md` context file for your project or user-level configuration. These functions help you set up the context files that `btw_client()` and `btw_app()` use to configure chat clients.

`use_btw_md()` creates a new context file with a default template. If the file already exists, it will not overwrite it, but will still ensure the file is added to `.Rbuildignore` if you're in an R package.

`edit_btw_md()` opens an existing context file for editing. Without arguments, it opens the same file that `btw_client()` would use by default.

Usage

```
use_btw_md(scope = "project")

edit_btw_md(scope = NULL)
```

Arguments

`scope`	The scope of the context file. Can be: • `"project"` (default): Creates/opens `btw.md` (by default) or `AGENTS.md` in the project root • `"user"`: Creates/opens `btw.md` in your home directory • A directory path: Creates/opens `btw.md` in that directory • A file path: Creates/opens that specific file For `edit_btw_md()`, `scope = NULL` (default) will find and open the context file that `btw_client()` would use, searching first for `btw.md` and then `AGENTS.md` in the project directory and then for `btw.md` in your home directory.

Value

`use_btw_md()` returns the path to the context file, invisibly. `edit_btw_md()` is called for its side effect of opening the file.

Functions

- `use_btw_md()`: Create a new `btw.md` or `AGENTS.md` context file in the current directory, the project directory or your home directory.
- `edit_btw_md()`: Open an existing `btw.md` or `AGENTS.md` context file for editing.

Additional Examples

```
# Create a project-level btw.md
use_btw_md()

# Create a user-level btw.md
use_btw_md("user")

# Create an AGENTS.md file
use_btw_md("AGENTS.md")

# Edit the context file that btw_client() would use
edit_btw_md()

# Edit a specific context file
edit_btw_md("user")
```

Project Context

You can use a `btw.md` or `AGENTS.md` file to keep track of project-specific rules, guidance and context in your project. Either file name will work, so we'll refer primarily to `btw.md`. These files are used automatically by `btw_client()` and `btw_app()`: they look first for `btw.md` and then for `AGENTS.md`. If both files are present, only the `btw.md` file will be used.

Any time you start a chat client with `btw_client()` or launch a chat session with `btw_app()`, btw will automatically find and include the contents of the `btw.md` or `AGENTS.md` file in the system prompt of your chat. This helps maintain context and consistency across chat sessions.

Use `btw.md` to inform the LLM of your preferred code style, to provide domain-specific terminology or definitions, to establish project documentation, goals and constraints, to include reference materials such or technical specifications, or more. Storing this kind of information in `btw.md` may help you avoid repeating yourself and can be used to maintain coherence across many chat sessions.

Write in markdown and structure the file in any way you wish, or use `btw_task_create_btw_md()` to help you create a project context file for an existing project with the help of an AI agent.

Chat Settings

You can also use the `btw.md` file to choose default chat settings for your project in a YAML front matter block at the top of the file. In this YAML block you can choose settings for the default ellmer chat `client`, e.g. `provider`, `model`, as well as choose which `btw_tools()` to use in `btw_client()` or `btw_app()`.

Chat client settings

Use the `client` field to set options for the chat client. This can be a single string in `provider` or `provider/model` format – as used by `ellmer::chat()` – or a list of client options with `provider` and `model` fields, as well as any other options supported by the underlying `ellmer::chat_*()` function you choose. Note that `provider` maps to the `ellmer::chat_*()` function, while `model` maps to the `model` argument of that function.

- Using ellmer's default model for a provider:

```
---
client: openai
---

---
client:
  provider: openai
---
```

- Using a specific model:

```
---
client: anthropic/claude-4-5-sonnet-latest
---

---
client:
  provider: anthropic
  model: claude-4-5-sonnet-latest
---
```

- Using additional client options:

```
---
client:
  provider: ollama
  model: "gpt-oss:20b"
  echo: output
  base_url: "http://my-company.example.com:11434"
---
```

Tool Settings

The top-level `tools` field is used to specify which btw tools are included in the chat. This should be a list of tool groups or tool names (with or without the `btw_tool_` prefix). See `btw_tools()` for a list of available tools and tool groups.

Here's an example `btw.md` file:

```
---
client: claude/claude-4-5-sonnet-latest
tools: [docs, env, files, git, ide, search, session, web]
---

Follow these important style rules when writing R code:

* Prefer solutions that use {tidyverse}
* Always use `<-` for assignment
* Always use the native base-R pipe `|>` for piped expressions
```

Selective Context

One use-case for `btw.md` is to provide stable context for an on-going task that might span multiple chat sessions. In this case, you can use `btw.md` to hold the complete project plan, with background information, requirements, and specific tasks to be completed. This can help maintain continuity across chat sessions, especially if you update the `btw.md` file as the project progresses.

In this use case, however, you might want to hide parts of the project plan from the system prompt, for example to hide completed or future tasks when their description would distract the LLM from the current task.

You can hide parts of the `btw.md` file from the system prompt by wrapping them in HTML `<!-- HIDE -->` and `<!-- /HIDE -->` comment tags. A single `<!-- HIDE -->` comment tag will hide all content after it until the next `<!-- /HIDE -->` tag, or the end of the file. This is particularly useful when your system prompt contains notes to yourself or future tasks that you do not want to be included in the system prompt.

Project or User Scope

For project-specific configuration, store your `btw.md` file in the root of your project directory. You can even have multiple `btw.md` files in your project, in which case the one closest to your current working directory will be used. This makes it easy to have different `btw.md` files for different sub-projects or sub-directories within a larger project.

For global configuration, you can maintain a `btw.md` file in your home directory (at `btw.md` or `.config/btw/btw.md` in your home directory, using `fs::path_home()`). This file will be used by default when a project-specific `btw.md` file is not found. Note that **btw** only looks for `btw.md` in your home directory if no project-specific `btw.md` or `AGENTS.md` file is present. It also does not look for `AGENTS.md` in your home directory.

Interactive Setup

For an interactive guided setup, consider using `btw_task_create_btw_md()` to use an LLM to help you create a `btw.md` file for your project.

See Also

Project context files are discovered automatically and included in the system prompt by `btw_client()`. See `btw_tools()` for a list of available tools.

Examples

```
# See additional examples in the sections above

withr::with_tempdir({
  withr::with_tempfile("btw_md_tmp", fileext = ".md", {
    use_btw_md(btw_md_tmp)
  })
})
```

Index