Package 'mcptools'

August 22, 2026

Title Model Context Protocol Servers and Clients

Version 1.0.2

Description Implements the Model Context Protocol (MCP). Users can start 'R'-based servers, serving functions as tools for large language models to call before responding to the user in MCP-compatible apps like 'Claude Desktop' and 'Claude Code', with options to run those tools inside of interactive 'R' sessions. On the other end, when 'R' is the client via the 'ellmer' package, users can register tools from third-party MCP servers to integrate additional context into chats.

License MIT + file LICENSE

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

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

Depends R (>= 4.1.0)

Imports cli, ellmer (>= 0.3.0), httpuv, httr2 (>= 1.2.3), jsonlite, nanonext (>= 1.6.0), openssl, processx, promises, rlang, yaml

Suggests callr, knitr, rmarkdown, testthat (>= 3.0.0), withr

VignetteBuilder knitr

Config/Needs/website tidyverse/tidytemplate

Config/testthat/edition 3

Encoding UTF-8

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Simon Couch [aut, cre] (ORCID: <https://orcid.org/0000-0001-5676-5107>), Winston Chang [aut] (ORCID: <https://orcid.org/0000-0002-1576-2126>), Charlie Gao [aut] (ORCID: <https://orcid.org/0000-0002-0750-061X>), Posit Software, PBC [cph, fnd] (ROR: <https://ror.org/03wc8by49>)

Maintainer Simon Couch <simon.couch@posit.co>

Repository CRAN

Date/Publication 2026-08-22 16:50:09 UTC

Contents

client 2
server 4

Index **8**

`client` *R as a client: Define ellmer tools from MCP servers*

Description

These functions implement R as an MCP *client*, so that ellmer chats can register functionality from third-party MCP servers such as those listed here: https://github.com/modelcontextprotocol/servers.

`mcp_tools()` fetches tools from MCP servers configured in the mcptools server config file and converts them to a list of tools compatible with the `$set_tools()` method of ellmer::Chat objects.

Usage

```
mcp_tools(config = NULL)
```

Arguments

`config`	A single string indicating the path to the mcptools MCP servers configuration file. If one is not supplied, mcptools will look for one at the file path configured with the option `.mcptools_config`, falling back to `file.path("~", ".config", "mcptools", "config.json")`.

Value

`mcp_tools()` returns a list of ellmer tools that can be passed directly to the `$set_tools()` method of an ellmer::Chat object. If the file at `config` doesn't exist, an error.

Configuration

mcptools uses the same .json configuration file format as Claude Desktop; most MCP servers will define example .json to configure the server with Claude Desktop in their README files. By default, mcptools will look to `file.path("~", ".config", "mcptools", "config.json")`; you can edit that file with `file.edit(file.path("~", ".config", "mcptools", "config.json"))`.

The mcptools config file should be valid .json with an entry `mcpServers`. That entry should contain named elements, each configuring either a local stdio server with `command` and `args`, or a remote Streamable HTTP server with `url`. Stdio MCP server processes receive an allowlisted environment inherited from the current R process, plus any variables configured in `env`. Configured `env` variables override inherited variables with the same name. Servers that need additional environment variables should list them in `env`.

For example, to configure `mcp_tools()` with GitHub's official MCP Server https://github.com/github/github-mcp-server, you could write the following in that file:

```
{
  "mcpServers": {
    "github": {
      "command": "docker",
      "args": [
        "run",
        "-i",
        "--rm",
        "-e",
        "GITHUB_PERSONAL_ACCESS_TOKEN",
        "ghcr.io/github/github-mcp-server"
      ],
      "env": {
        "GITHUB_PERSONAL_ACCESS_TOKEN": "<add_your_github_pat_here>"
      }
    }
  }
}
```

Connecting to remote (http) servers

For remote Streamable HTTP MCP servers, configure a server with `url`. Static headers can be supplied with `headers`; protocol-owned headers such as `Accept`, `Content-Type`, `MCP-Session-Id`, and `MCP-Protocol-Version` are managed by mcptools and cannot be configured manually. Remote endpoints must use HTTPS; HTTP is allowed only for loopback development servers or for explicit unsafe opt-out with `allow_http`.

When no static `Authorization` header is configured and the server answers an unauthenticated request with a `401` OAuth challenge, mcptools automatically runs the OAuth authorization-code flow, opening a browser for sign-in. Tokens are cached and refreshed, so the browser step happens only until a cached token expires. An `oauth` block is optional and only needed to override the defaults described below.

Remote server entries support these fields:

- `url`: the Streamable HTTP MCP endpoint.
- `headers`: named static headers. Values may use `${ENVVAR}` interpolation.
- `timeout`: a number of seconds for the overall HTTP request timeout.
- `allow_http`: allow credentialed non-loopback HTTP endpoints.
- `ignore_tools`: tool names or `*` wildcards to hide and block.
- `oauth`: OAuth settings.

OAuth settings may include `authorization_server`, `resource`, `scope` with `scope_mode = "override"`, `client_info`, `manual_client_info`, `client_metadata`, `redirect_uri` or `callback_host`/`callback_port`/`callback_path`, `cache_dir`, and `allow_http`. mcptools supports OAuth 2.1 with PKCE: it discovers the authorization server from the protected-resource metadata advertised in a `401` challenge, registers a client dynamically when the server supports it, and caches tokens (refreshing them automatically).

Remote HTTP requests use httr2 and curl. Proxy and corporate CA settings should generally use the standard curl environment variables, such as `HTTPS_PROXY`, `NO_PROXY`, `SSL_CERT_FILE`, and `CURL_CA_BUNDLE`. Stdio server processes inherit these variables through mcptools' default environment allowlist.

```
{
  "mcpServers": {
    "remote-example": {
      "url": "https://remote.mcp.server/mcp",
      "timeout": 30,
      "headers": {
        "Authorization": "Bearer ${REMOTE_MCP_TOKEN}"
      }
    }
  }
}
```

See Also

This function implements R as an MCP *client*. To use R as an MCP *server*, i.e. to provide apps like Claude Desktop or Claude Code with access to R-based tools, see `mcp_server()`.

Examples

```
# setup
config_file <- tempfile(fileext = "json")
file.create(config_file)

# usually, `config` would be a persistent, user-level
# configuration file for a set of MCP server
mcp_tools(config = config_file)

# teardown
file.remove(config_file)
```

`server` *R as a server: Configure R-based tools with LLM-enabled apps*

Description

`mcp_server()` implements a model context protocol server with arbitrary R functions as its tools. Optionally, calling `mcp_session()` in an interactive R session allows those tools to execute inside of that session.

Usage

```
mcp_server(
  tools = NULL,
  ...,
  type = c("stdio", "http"),
  host = "127.0.0.1",
  port = as.integer(Sys.getenv("MCPTOOLS_PORT", "8080")),
  session_tools = TRUE
)

mcp_session()
```

Arguments

tools	Optional collection of tools to expose. Supply either a list of objects created by `ellmer::tool()` or a path to an `.R` file that, when sourced, yields such a list. Defaults to `NULL`, which serves only the built-in session tools when `session_tools` is `TRUE`. Note that **tools are associated with the** `mcp_server()` rather than with `mcp_session()`s; to determine what tools are available in a session, set the `tools` argument to `mcp_server()`.
...	Reserved for future use; currently ignored.
type	Transport type: `"stdio"` for standard input/output (default), or `"http"` for HTTP-based transport.
host	Host to bind to when using HTTP transport. Defaults to `"127.0.0.1"` (localhost) for security. Ignored for stdio transport.
port	Port to bind to when using HTTP transport. Defaults to the value of the `MCPTOOLS_PORT` environment variable, or 8080 if not set. Ignored for stdio transport.
session_tools	Logical value whether to include the built-in session tools (`list_r_sessions`, `select_r_session`) that work with `mcp_session()`. Defaults to `TRUE`. Note that the tools to interface with sessions are still first routed through the `mcp_server()`.

Value

`mcp_server()` and `mcp_session()` are both called primarily for their side-effects.

- `mcp_server()` blocks the R process it's called in indefinitely and isn't intended for interactive use.
- `mcp_session()` makes the interactive R session it's called in available to MCP servers. It returns invisibly the **nanonext** socket used for communicating with the server. Call `close()` on the socket to stop the session.

Configuration

Local server (default, via stdio):

`mcp_server()` can be configured with MCP clients via the `Rscript` command. For example, to use with Claude Desktop, paste the following in your Claude Desktop configuration (on macOS, at `file.edit("~/Library/Application Support/Claude/claude_desktop_config.json")`):

```
{
  "mcpServers": {
    "r-mcptools": {
      "command": "Rscript",
      "args": ["-e", "mcptools::mcp_server()"]
    }
  }
}
```

Or, to use with Claude Code, you might type in a terminal:

```
claude mcp add -s "user" r-mcptools Rscript -e "mcptools::mcp_server()"
```

Remote server (via http):

To run an HTTP server instead, use `type = "http"`:

```
# Start HTTP server on default port (8080)
mcp_server(type = "http")

# Or specify custom host and port
mcp_server(type = "http", host = "127.0.0.1", port = 9000)
```

The server will listen for HTTP POST requests containing JSON-RPC messages.

Posit Connect:

To deploy an HTTP MCP server to Posit Connect, add a `_server.yml` file to the project directory:

```
engine: mcptools
tools: tools.R
```

The `tools` file should return a list of `ellmer::tool()` objects. Deploy the project as an R API and mark it as MCP content:

```
rsconnect::deployAPI(".", contentCategory = "mcp")
```

Use the Connect content URL with `/mcp` appended as the MCP endpoint. For example, if the content URL is `https://connect.example.com/content/abc123/`, use `https://connect.example.com/content/abc12` mcptools accepts requests at any path, so the content URL itself also works. If you cannot set `contentCategory = "mcp"` during deployment, set the MCP category in Connect after deploying and set minimum processes to at least 1.

Connect deployments run tools inside the deployed R process by default. Session discovery with `mcp_session()` is intended for local desktop R sessions and is disabled by default on Connect.

mcp_server() is not intended for interactive use.

The server interfaces with the MCP client. If you'd like tools to have access to variables inside of an interactive R session, call `mcp_session()` to make your R session available to the server. Place a call to `mcptools::mcp_session()` in your `.Rprofile`, perhaps with `usethis::edit_r_profile()`, to make every interactive R session you start available to the server.

The server and its sessions find each other through per-user IPC sockets on the local machine, so only sessions running as the same user on the same host are discoverable. On Linux and macOS the sockets live in an owner-only directory; set the `MCPTOOLS_SOCKET_DIR` environment variable (before the package loads) to override its location.

When several sessions are available, the server connects to the session whose working directory matches its own; clients launched inside a project, as with Claude Code or Posit Assistant, thus connect to that project's session. Failing that, the server connects to the only running session, if there is exactly one. When several sessions are running and none matches, tools execute in the server's own R process until the client selects a session with the `select_r_session` tool.

On Windows, you may need to configure the full path to the Rscript executable. Examples for Claude Code on WSL and Claude Desktop on Windows are shown at https://github.com/posit-dev/mcptools/issues/41#issuecomment-3036617046.

See Also

- The "R as an MCP server" vignette at `vignette("server", package = "mcptools")` delves into further detail on setup and customization.
- These functions implement R as an MCP *server*. To use R as an MCP *client*, i.e. to configure tools from third-party MCP servers with ellmer chats, see `mcp_tools()`.

Examples

```
# should only be run non-interactively, and will block the current R process
# once called.
if (identical(Sys.getenv("MCPTOOLS_CAN_BLOCK_PROCESS"), "true")) {
# to start a server with a tool to draw numbers from a random normal:
library(ellmer)

tool_rnorm <- tool(
  rnorm,
  "Draw numbers from a random normal distribution",
  n = type_integer("The number of observations. Must be a positive integer."),
  mean = type_number("The mean value of the distribution."),
 sd = type_number("The standard deviation of the distribution. Must be a non-negative number.")
)

mcp_server(tools = list(tool_rnorm))

# can also supply a file path as `tools`
readLines(system.file("example-ellmer-tools.R", package = "mcptools"))

mcp_server(tools = system.file("example-ellmer-tools.R", package = "mcptools"))
}

if (interactive()) {
  mcp_session()
}
```

Index

client, 2
close(), *5*

ellmer::Chat, *2*
ellmer::tool(), *5*, *6*

mcp_client *(*client*)*, 2
mcp_server *(*server*)*, 4
mcp_server(), *4*, *5*
mcp_session *(*server*)*, 4
mcp_session(), *6*
mcp_tools *(*client*)*, 2
mcp_tools(), *7*

server, 4