

Package ‘mcplite’

October 20, 2026

Title Lightweight Stdio MCP Server for R

Version 0.2.0

Description Provides a lightweight Model Context Protocol (MCP) server for exposing 'R' functions as tools over standard input and output ('stdio'). Designed for local, client-launched integrations, with protocol-aware tool definitions and results, JSON Schema helpers, and optional interoperability with 'ellmer'.

License MIT + file LICENSE

URL <https://tosidata.github.io/mcplite/>,
<https://github.com/tosidata/mcplite>

BugReports <https://github.com/tosidata/mcplite/issues>

Depends R (>= 4.1.0)

Imports jsonlite (>= 0.9.16), nanonext (>= 1.6.0), otel

Suggests ellmer (>= 0.3.0), otelsdk, processx, testthat (>= 3.0.0),
withr

Config/testthat/edition 3

Encoding UTF-8

Config/roxygen2/version 8.0.0

Config/Needs/website pkgdown, rmarkdown

NeedsCompilation no

Author Juha Itkonen [aut, cre, cph]

Maintainer Juha Itkonen <juha@tosidata.com>

Repository CRAN

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

Contents

content-blocks	2
mcplite	3

2

content-blocks

mcplite-results 4

mcp_server 6

tool 8

tool-types 10

tool_result 12

Index14

content-blocks	<i>Construct protocol-native MCP content blocks</i>
----------------	---

Description

These constructors create the only content-block objects accepted by `tool_result()`. All R-facing names use snake_case; MCP names such as mimeType and _meta are created only when a result is serialized.

Usage

```
content_text(text, annotations = list(), meta = list())

content_image(data, mime_type, annotations = list(), meta = list())

content_audio(data, mime_type, annotations = list(), meta = list())

content_resource_link(
  uri,
  name,
  title = NULL,
  description = NULL,
  mime_type = NULL,
  size = NULL,
  annotations = list(),
  meta = list()
)

content_resource(
  uri,
  text = NULL,
  blob = NULL,
  mime_type = NULL,
  annotations = list(),
  meta = list()
)
```

Arguments

text	A single text string. For <code>content_resource()</code> , the resource text payload; exactly one of <code>text</code> or <code>blob</code> must be supplied.
annotations	Optional named MCP annotations. Known R-facing fields are <code>audience</code> , <code>priority</code> , and <code>last_modified</code> ; named extension fields are preserved when serializable.
meta	Optional named outer content-block metadata. For <code>content_resource()</code> , metadata is not duplicated into the inner resource contents object.
data	A scalar base64 string containing inline image or audio data.
mime_type	A single MIME type string. Optional for resources and resource links.
uri	A single non-empty author-supplied resource URI.
name	A single non-empty resource-link name.
title	Optional resource-link display title.
description	Optional resource-link description.
size	Optional non-negative whole-number resource size in bytes.
blob	A scalar base64 string containing an embedded resource payload; exactly one of <code>text</code> or <code>blob</code> must be supplied.

Details

Image, audio, and resource blob data must already be scalar base64 strings. The constructors do not read files, download URLs, infer MIME types, encode, decode, or validate base64 data, or invent resource URIs. Tool authors own payload integrity, accurate MIME types, meaningful URIs, authorization, sanitization, and payload size decisions.

Resource links only transport an author-supplied URI. `mcplite` remains a tools-only server and does not make that URI readable or implement `resources/read`. Embedded resources are self-contained content blocks.

Value

A validated content block for use in `tool_result()`.

mcplite

mcplite: Lightweight stdio MCP server for R

Description

Expose R functions as tools for a local Model Context Protocol (MCP) client. Define tools, save them in a script ending with `mcp_server()`, and configure the client to launch that script with `Rscript --vanilla` and an absolute script path. The server blocks until standard input closes.

Define tools and return results

Use `mcplite::tool()` with the [tool-types](#) helpers to describe arguments. Tools remain directly callable R functions. Return ordinary R values for text conversion, or `tool_result()` with `content_text()` and other [content-blocks](#) constructors for structured or rich output. See [mcplite-results](#) for exact return-value mappings and payload JSON options.

Schemas do not validate values or enforce permissions. Tools own domain validation, authorization, safe side effects, and output limits. Keep stdout exclusively for MCP protocol messages; use stderr for diagnostics.

Scope and next steps

mcplite is tools-only and stdio-only; embedded resources and resource links in results do not provide resource endpoints. See `mcp_server()` for server use, supported protocol versions, and optional tracing. Compatible `ellmer::tool()` objects are accepted, but `ellmer` is not required.

Website guides: [Get started](#), [Write tools](#), and [Return results](#).

Author(s)

Maintainer: Juha Itkonen <juha@tosidata.com> [copyright holder]

Authors:

- Juha Itkonen <juha@tosidata.com> [copyright holder]

See Also

Useful links:

- <https://tosidata.github.io/mcplite/>
- <https://github.com/tosidata/mcplite>
- Report bugs at <https://github.com/tosidata/mcplite/issues>

mcplite-results

Return-value mappings and payload JSON options

Description

Tools can return ordinary R values for text conversion, or `tool_result()` for explicit MCP content and structured data. This topic defines the shared serialization contract. See `tool_result()` for construction and fallback, and `mcp_server()` under "Protocol compatibility" for version projection.

Ordinary return values

Character scalars become literal text and character vectors are joined with newlines. Other JSON-serializable values become JSON in one text content block. Bare lists are ordinary values, not MCP result objects. Ordinary NULL becomes the text "{}"; an ordinary unnamed `list()` becomes "[]".

For per-tool conversion choices, return your own JSON as ordinary character text, for example `as.character(jsonlite::toJSON(value, na = "null"))`. JSON-looking strings and jsonlite json objects are never parsed or inserted as raw JSON. In `structured_content`, they remain string values.

Structured-content mappings

Structured values follow `jsonlite::toJSON()` and the payload options below, with one root convention: a bare empty list at the structured root represents {}. Nested unnamed empty lists retain jsonlite's [] mapping. NULL means absence of the structured-content field, not a JSON null value.

With default options:

R value	Structured-content meaning
<code>list(ok = TRUE)</code>	JSON object
<code>c("a", "b")</code>	JSON array
<code>"done"</code>	JSON string scalar, not parsed as JSON
<code>list()</code> or <code>structure(list(), names = character())</code>	Empty object {}
<code>I(list())</code>	Explicit empty array []
<code>NA</code> or <code>NA_character_</code>	JSON null
<code>NA_integer_</code> , <code>NA_real_</code> , or <code>NA_complex_</code>	JSON string "NA"
<code>NULL</code>	No structured-content field

Classes, dimensions, and explicit `I()` or `jsonlite::unbox()` representations retain their jsonlite mappings. Tool authors are responsible for conformance with any advertised output schema.

Payload JSON options

Configure these package-global R options before constructing results and starting the server, and keep them stable while the server runs:

- `mcplite.json.null`: "null" (default) or "list". Named payload NULL fields remain present, as JSON null or {}, respectively.
- `mcplite.json.na`: unset by default, which omits jsonlite's `na` argument and preserves its class-specific defaults. Set "null" or "string" to select that jsonlite mapping. "null" also maps NaN and infinities to JSON null and retains missing data-frame fields instead of omitting them.
- `mcplite.json.auto_unbox`: TRUE (default) or FALSE. Setting FALSE keeps singleton payload vectors as arrays; explicit `I()` and `jsonlite::unbox()` representations still apply.

Remove an option with `NULL`, for example `options(mcplite.json.na = NULL)`, to restore its default. This omits `na`; it does not forward `na = NULL`. These options apply to ordinary noncharacter return-value JSON and to structured content, its representability checks, generated text fallback, and protocol-version projection. Protocol fields, schemas, annotations, and all result and content metadata retain fixed encoding.

See Also

`mcplite`, `tool_result()`, `content_text()`

Examples

```
local({
  old <- options(mcplite.json.na = "null", mcplite.json.auto_unbox = FALSE)
  on.exit(options(old))
  result <- tool_result(structured_content = list(
    count = 1L,
    values = c(NA_real_, NaN, Inf, -Inf),
    rows = data.frame(id = 1L, missing = NA_real_)
  ))
  result$content[[1]]$text
})
```

mcp_server

Run an MCP server for R tools over stdio

Description

`mcp_server()` runs a **Model Context Protocol** server over standard input and output. It exposes tools created with `mcplite::tool()` so MCP clients can discover and call R functions. Compatible `ellmer::tool()` objects can also be supplied when users already use `ellmer`, but `ellmer` is not required for ordinary `mcplite` tools.

Usage

```
mcp_server(tools, instructions = NULL)
```

Arguments

<code>tools</code>	Tool definitions to expose. Supply one <code>mcplite::tool()</code> object or a list of tool definitions. Compatible <code>ellmer::tool()</code> objects are accepted when supplied directly.
<code>instructions</code>	Optional server instructions to advertise to clients using a protocol version that supports them.

Value

`mcp_server()` is called for its side effect of serving MCP requests. It blocks the current R process until standard input closes.

Run a server

Supply one tool or a list of tools directly. Put the complete setup in a script that defines or sources tools and ends with `mcplite::mcp_server(actual_tool_or_list)`. Configure the MCP client to launch it with `Rscript --vanilla /absolute/path/to/server.R`; use an absolute script path because the client's working directory is not guaranteed. The call blocks until standard input closes, so use a client-launched script rather than an interactive R console.

Tools may return ordinary R values for text conversion or opt into native content and structured output with `tool_result()`. See [mcplite-results](#) for exact mappings and payload JSON options. Set those options before constructing results and starting the server, and keep them stable while it runs. Tools own input validation, output conformance, permissions, and safe side effects; schemas are not authorization or sandboxing.

Keep stdout exclusively for MCP protocol messages. Send diagnostics and telemetry to `stderr` or a remote exporter, not `stdout`.

Server scope

mcplite is a tools-only, stdio-only server. It does not implement JSON-RPC batching, HTTP transports, sessions, prompts, resource listing or reading, subscriptions, cancellation, sampling, elicitation, roots, Tasks, MCP Apps, progress notifications, or server-initiated and other multi-round-trip requests. Embedded resource blocks are self-contained, and resource links do not make their URIs readable through mcplite.

Protocol compatibility

The same server definition supports four MCP versions. A tool remains available when its output schema cannot be represented by the client's protocol. Compatible explicit or generated text content remains available when structured content is omitted. Tool authors still own full schema validity and runtime input/output conformance.

MCP version	Lifecycle	Output schemas	Structured content	Audio, resource links
2026-07-28	Protocol selected per request	Every supported shape	Every supported JSON value	Supported
2025-11-25	initialize first	Direct object-root only	Object values only	Supported
2025-06-18	initialize first	Direct object-root only	Object values only	Supported
2024-11-05	initialize first	Omitted	Omitted	Unsupported

Text, images, embedded resources, annotations, and result metadata work in every supported version. A 2024-11-05 call returning audio or a resource link receives a contained tool-level error instead of partial content or server termination.

Modern 2026-07-28 clients select the protocol with per-request metadata. `server/discover`, `tools/list`, and `tools/call` do not require `initialize`. Discovery advertises 2026-07-28, 2025-11-25, 2025-06-18, and 2024-11-05 in that preference order. Every successful modern result includes `resultType = "complete"` and the server's canonical identity in `_meta["io.modelcontextprotocol/server"]`. Modern `server/discover` and `tools/list` results also include public cache hints with a 300000 millisecond TTL. Modern ping is unsupported. Legacy clients retain ping, `tools/list`, and `tools/call` after initialization.

OpenTelemetry tracing

mcplite automatically creates one OpenTelemetry server span for every parsed MCP request or notification that passes JSON-RPC envelope validation. This includes retained legacy initialization and ping, notifications, modern discovery, tool listing and calls, and valid unknown methods. Blank input, JSON parse failures, and malformed JSON-RPC envelopes do not create MCP operation spans.

Tool authors do not need to call `otel::start_local_active_span()` for the MCP operation or tool invocation. The server span remains active while tool code runs, so optional tool-authored spans can become children without being required. mcplite does not create a redundant automatic tool-execution child span.

Provider and exporter configuration belongs to the standard `otel` and `otelsdk` environment variables and APIs. mcplite does not add telemetry arguments, choose an exporter, or configure a provider. With no exporter configured, tracing is an effective no-op and MCP behavior is unchanged. A safe `stderr` configuration is `OTEL_R_TRACES_EXPORTER=stderr Rscript --vanilla /absolute/path/to/server`; a remote exporter such as OTLP is also suitable.

Do not use a stdout or console exporter with a stdio MCP server. Standard output is reserved exclusively for MCP protocol messages, so telemetry written there will corrupt the protocol stream.

Remote W3C parent context may be supplied in `params._meta.traceparent`, with optional `params._meta.tracestate`. Malformed propagation data is ignored, and `_meta` is not passed to tool functions. By default, spans contain selected low-cardinality operation metadata; raw requests and responses, tool arguments and results, `_meta`, trace headers, and condition messages are not recorded as span attributes.

Examples

```
if (identical(Sys.getenv("MCPLITE_CAN_BLOCK_PROCESS"), "true")) {
  add_numbers <- tool(
    function(x, y) {
      x + y
    },
    name = "add_numbers",
    description = "Add two numbers and return the result.",
    arguments = list(
      x = type_number("First number."),
      y = type_number("Second number.")
    )
  )

  mcp_server(list(add_numbers))
}
```

Description

`tool()` wraps an R function with MCP metadata. The resulting object can be supplied directly, or in a list, to `mcp_server()`. Ordinary function return values use the default single-text-block conversion; return `tool_result()` to opt into protocol-native content, structured output, or result metadata.

Usage

```
tool(  
  fun,  
  description,  
  ...,  
  arguments = list(),  
  name = NULL,  
  annotations = list(),  
  output_schema = NULL  
)
```

Arguments

<code>fun</code>	Function to expose as a tool.
<code>description</code>	A single string describing when and how to use the tool.
<code>...</code>	Not used. Supply argument schemas with <code>arguments</code> .
<code>arguments</code>	Named list of argument schemas created by the <code>type_*</code> () helpers, or one complete direct object-root schema created by <code>type_from_schema()</code> .
<code>name</code>	Optional tool name. If omitted, <code>tool()</code> uses the symbol name supplied to <code>fun</code> ; anonymous functions must supply <code>name</code> . Tool names must use 1 to 128 characters from letters, digits, underscore, dot, and hyphen.
<code>annotations</code>	Optional named list of MCP tool annotations to advertise.
<code>output_schema</code>	Optional output schema created by a supported <code>type_*</code> () helper. Raw schema lists must be wrapped with <code>type_from_schema()</code> . See "Complete schemas and structured output" for author responsibilities and protocol compatibility.

Value

The supplied function with MCP metadata attached. It remains directly callable as an R function and can be supplied to `mcp_server()`.

Arguments and defaults

Incoming JSON arrays reach `fun` as R lists, not atomic vectors. Schemas do not coerce or validate argument values or supply R defaults. Marking a property `required = FALSE` permits omission in the advertised schema; the R function must provide its own default for calls that omit that argument.

Return values

Ordinary character values become literal text, not parsed JSON. Bare lists are ordinary values, not MCP result objects. See [Return-value mappings](#) for exact conversions, empty and missing values, and payload JSON options. Configure those options before constructing results and starting the server.

Complete schemas and structured output

`arguments` also accepts one complete object-root schema from `type_from_schema()`. Complete input schemas must have direct root `type = "object"`; JSON Schema 2020-12 keywords such as `$schema`, `$defs`, `$ref`, `composition`, and conditionals are preserved.

An `output_schema` advertises the expected `structured_content` shape. A client's protocol can omit an incompatible output schema without removing the tool; see `mcp_server()` under "Protocol compatibility" for the exact version rules. `mcp-lite` checks schema container shape and serializability, not full schema validity or runtime input/output conformance. Tool authors own those guarantees.

Examples

```
add_numbers <- tool(
  function(x, y) {
    x + y
  },
  name = "add_numbers",
  description = "Add two numbers and return the result.",
  arguments = list(
    x = type_number("First number."),
    y = type_number("Second number.")
  )
)
add_numbers(2, 3)
```

tool-types

Define MCP tool schemas

Description

These helpers create common JSON Schema descriptions for tool arguments and output. `type_from_schema()` wraps a complete schema supplied as a list or JSON string without removing JSON Schema 2020-12 keywords. A complete tool input schema can be passed directly as `arguments = type_from_schema(...)` when it has direct root `type = "object"`.

Usage

```
type_boolean(description = NULL, required = TRUE)
```

```
type_integer(description = NULL, required = TRUE)
```

```

type_number(description = NULL, required = TRUE)

type_string(description = NULL, required = TRUE)

type_enum(values, description = NULL, required = TRUE)

type_array(items, description = NULL, required = TRUE)

type_object(
  .description = NULL,
  ...,
  .required = TRUE,
  .additional_properties = FALSE
)

type_from_schema(text = NULL, path = NULL)

type_ignore()

```

Arguments

description	Optional argument description.
required	Whether the argument is listed as required in the parent schema. FALSE permits omission but does not set an R default.
values	Allowed enum values.
items	Type helper describing each array item.
.description	Optional object description.
...	Named properties for object schemas.
.required	Whether the object itself is listed as required in its parent schema.
.additional_properties	Whether to allow additional properties.
text	A JSON Schema as a list or JSON string.
path	Path to a JSON Schema file. Exactly one of text or path must be supplied.

Details

When `$schema` is absent, MCP treats schemas as JSON Schema 2020-12. Authors are responsible for schema validity and for runtime input/output conformance, domain validation, authorization or access checks, side-effect safety, output sanitization, and rate limiting where needed. List input preserves the supplied R list shape: use a named empty list for `{}` and an unnamed empty list for `[]`.

Incoming JSON arrays remain R lists. These helpers do not coerce or validate argument values or supply R defaults. If an argument may be omitted, give the R function a default for it.

`type_ignore()` omits an argument from the advertised schema. It does not reject a caller-supplied argument or enforce permissions.

Value

A lightweight mcplite tool type object.

Examples

```
type_string("A label.")

type_array(type_integer(), description = "Integer values.")

type_object(
  .description = "A labeled score.",
  label = type_string(),
  score = type_number(required = FALSE)
)

type_from_schema(list(
  type = "string",
  minLength = 1
))
```

tool_result

Construct protocol-native MCP tool results

Description

tool_result() is the explicit opt-in boundary for returning protocol-native MCP results. Its content must be created with one of the content_*() constructors; ordinary R values, including bare lists and JSON strings, retain the single-text-block conversion used by mcplite::tool().

Usage

```
tool_result(
  content = list(),
  structured_content = NULL,
  is_error = FALSE,
  meta = list()
)
```

Arguments

content	One content block created by a content_*() constructor, or an ordered list of such blocks. Omitting content permits automatic text fallback generation from structured_content; explicit content = list() suppresses fallback and is invalid.
structured_content	Optional R value representable as a JSON object, array, or scalar. NULL means absence; a bare empty list at this root represents {}. See mcplite-results for other mappings and payload options. mcplite checks JSON representability but does not validate this value against a tool's advertised output schema.

is_error	Whether the result reports a tool error.
meta	Optional named result metadata. MCP wire names such as <code>_meta</code> are created only during serialization.

Value

A protocol-native tool result for return from a tool function.

Content and fallback

When `structured_content` is supplied and `content` is omitted, `tool_result()` generates exactly one JSON text fallback block. Explicitly supplied content is preserved as-is and no fallback is appended. Keep that content useful on its own: a client's protocol may omit structured content or restrict its shape. See [mcp_server\(\)](#) under "Protocol compatibility".

See [Return-value mappings](#) for exact R-to-JSON mappings and payload options. JSON-looking strings and jsonlite json objects remain string values in `structured_content`; they are not parsed as raw JSON. Tool authors own conformance with any advertised output schema.

Tool errors

`is_error = TRUE` reports a tool-level error while retaining the normal MCP tool-result envelope.

See Also

[mcplite-results](#), [content_text\(\)](#), [content_image\(\)](#), [content_audio\(\)](#), [content_resource_link\(\)](#), and [content_resource\(\)](#).

Examples

```
tool_result(content_text("Done."))

tool_result(
  content = list(
    content_text("Generated the plot."),
    content_image("base64-data", "image/png")
  )
)

tool_result(structured_content = list(ok = TRUE, count = 2L))
```

Index

`content-blocks`, [2](#), [4](#)
`content_audio (content-blocks)`, [2](#)
`content_audio()`, [13](#)
`content_image (content-blocks)`, [2](#)
`content_image()`, [13](#)
`content_resource (content-blocks)`, [2](#)
`content_resource()`, [13](#)
`content_resource_link (content-blocks)`,
[2](#)
`content_resource_link()`, [13](#)
`content_text (content-blocks)`, [2](#)
`content_text()`, [4](#), [6](#), [13](#)

`jsonlite::toJSON()`, [5](#)
`jsonlite::unbox()`, [5](#)

`mcp_server`, [6](#)
`mcp_server()`, [3](#), [4](#), [9](#), [10](#), [13](#)
`mcplite`, [3](#), [6](#)
`mcplite-package (mcplite)`, [3](#)
`mcplite-results`, [4](#), [4](#), [7](#), [12](#), [13](#)

Return-value mappings, [10](#), [13](#)

`tool`, [8](#)
`tool-types`, [4](#), [10](#)
`tool_result`, [12](#)
`tool_result()`, [2–4](#), [6](#), [7](#), [9](#)
`type_array (tool-types)`, [10](#)
`type_boolean (tool-types)`, [10](#)
`type_enum (tool-types)`, [10](#)
`type_from_schema (tool-types)`, [10](#)
`type_from_schema()`, [9](#), [10](#)
`type_ignore (tool-types)`, [10](#)
`type_integer (tool-types)`, [10](#)
`type_number (tool-types)`, [10](#)
`type_object (tool-types)`, [10](#)
`type_string (tool-types)`, [10](#)