MCPClient

Description

Create one or more MCP client instances from a configuration struct.

Returns

List of MCP client objects that exposes the runtime methods like:

- listTools()
- callTool()
- listPrompts()
- getPrompt(params)
- listResources()
- readResource(uri)

Category

MCP functions

Syntax

MCPClient(configStruct)

History

New in ColdFusion 2025.1

Parameters

The configuration struct contains:

Single server

Field	Data type	Required	Description

`transport`	Struct	Yes	Describes how the client reaches one MCP server.
			• HTTP transport: Use this when the MCP server is exposed over HTTP/streamable HTTP.
			`transport = {`
			`    type = "http",`
			`    url  = "https://remote.mcpservers.org/fetch/mcp"`
			`}`
			  ○ `type: must be "http" for HTTP-based servers.`
			  ○ `url: full URL to the MCP endpoint (e.g. /mcp, /mcp/call, /fetch/mcp), as shown in spec examples.`
			• `stdio transport: Use this when the MCP server runs as a local process spawned by CF (Node/Python/Java/CF):`
			`transport = {`
			`    type    : "stdio",`
			`    command : "/usr/local/bin/node",`
			`    args    : [ "/path/to/server.js" ],`
			`    env     = { apiKey : "sk-xxx" }   // optional, server-specific`
			`};`
			  • `type: "stdio" for subprocess-based servers.`
			  • command: executable to run (e.g. node, python, java).

			• args: arguments array, typically including the MCP server script or jar. • env: (optional) map of environment variables the MCP server needs (API keys, hosts, tokens).
`clientInfo`	Struct	No	Identifies your MCP client to the server during initialization. `clientInfo = { name = "weather MCP Client", version = "1.0.0" };` • `name: descriptive, helps the server/logs identify the client (e.g. “CF Jira Client”).` • `version: version string for your client (semantic versioning is good practice).`
`capabilities`	Struct	No	Tells the server which optional MCP features the client supports and intends to use. `capabilities = { sampling : true, roots : true, elicitation : false };` • `sampling – whether the client supports MCP sampling (asking an LLM to generate intermediate completions).` • `roots – whether the client will provide resource roots (URIs that scope resources).` • `elicitation – whether the client supports interactive elicitation workflows.`
`initializationTimeout`	Numeric	No	Maximum time to wait for the MCP to initialize handshake. (default: 10, max: 3600). `initializationTimeout : 10;  // seconds`

`requestTimeout`	Numeric	No	Max time to wait for tool/resource/prompts calls. (default: 30, max: 3600) `requestTimeout : 10;  // seconds`
`loggingConsumer`	Function	No	Receives structured log events emitted by the MCP server. This is the primary hook for: • Centralized logging of MCP server activity. • Debugging tool calls and resource reads. • Building observability pipelines. *function loggingConsumer(any messages) { ... }* Whenever the server sends logs to the client, for example, on tool calls, errors, or general server events. This is especially useful in production to debug tool behavior without accessing server logs directly.
`samplingConsumer`	Function	No	Receives events related to sampling, when the MCP client/server interacts with an LLM to generate intermediate completions or sampling-based operations. This is useful when you want to: • Inspect intermediate prompts. • Log or audit how sampling is used. *function samplingConsumer(array messages) { ... }* messages is typically an array of sampling message objects, often including: • Role ("assistant", "system", etc.). • Content (text, tool call suggestions, etc.). • Metadata (timestamps, IDs).
`elicitationConsumer`	Function	No	Handles elicitation events, interactive flows where the server requests additional input or clarifications (e.g., the system needs more user info to proceed). This is where you might: • Present follow-up questions in a UI. • Capture additional form fields or user responses. • Decide whether to continue or abort a workflow. function elicitationConsumer(array messages) { ... } Like samplingConsumer, messages is an

			array of elicitation messages (questions, prompts for input, etc.).
`toolsChangeConsumer`	Function	No	Notified when the available tools change for that MCP client. This can happen if: • The MCP server hot-loads new tools. • Some tools are disabled/enabled at runtime. • A multi-tenant configuration and your client's context changes. *function toolsChangeConsumer(array tools) { … }*

Multiple servers

When you want one ColdFusion app to talk to multiple MCP servers at once, you can still use MCPClient(configData). The key difference from the single-server case is that your configData now references a config file that describes multiple servers, and MCPClient will return one client per server.

Field	Data type	Required	Description
`configFile`	String	Yes	A file path (string) pointing to a JSON configuration file (for example, mcpServers.json). The JSON file describes one or more MCP servers, each with its own transport settings (type, url / command / args / env).

`clientInfo`	Struct	No	When you use multiple MCP servers via MCPClient(configData) with a configFile, the clientInfo field in configData is a shared identity that is applied to each MCP client created from that config.
			`clientInfo = {`
			`    name    = "Healthcare MCP Client",`
			`    version = "1.0.0"`
			`};`

`capabilities`	Struct	No	When you configure multiple MCP servers via MCPClient(configData) with a configFile, the capabilities field in configData is a shared capabilities descriptor that applies to every MCP client created from that config.
			capabilities = {
			  sampling : true,
			  roots : true,
			  elicitation : false
			};
			
			• sampling: client supports MCP sampling callbacks (LLM-assisted sampling).
			• roots: client understands and will provide roots to scope resources.
			• elicitation: client participates in elicitation (interactive ask-for-more-info flows).

`initializationTimeout`	Numeric	No	Same as that of a single MCP server.
`requestTimeout`	Numeric	No	Same as that of a single MCP server.
`loggingConsumer`	Function	No	Same as that of a single MCP server.
`samplingConsumer`	Function	No	Same as that of a single MCP server.
`elicitationConsumer`	Function	No	Same as that of a single MCP server.

Example

Single server

```
<cfscript>
    transportConfig = {
        "type" = "http",
        "url"  = "https://remote.mcpservers.org/fetch/mcp"
    };

    clientInfo = {
        "name"    : "CF MCP Client with Consumers",
        "version" : "1.0.0"
    };

    capabilities = {
        "sampling"    : true,
        "roots"       : true,
        "elicitation" : true
    };

    configData = {
        "transport":             transportConfig,
        "clientInfo":            clientInfo,
        "capabilities":          capabilities,
        "initializationTimeout": 10,
        "requestTimeout":        10,

        samplingConsumer: function(messages) {
            writeLog(file = "MCP_SAMPLING", text = serializeJSON(messages));
        },

        elicitationConsumer: function(messages) {
            application.lastElicitation = messages;
            writeLog(file = "MCP_ELICITATION", text = serializeJSON(messages));
        },

        loggingConsumer: function(logEvent) {
            var payload = structKeyExists(logEvent, "data") ? logEvent.data :
logEvent;
            writeLog(file = "MCP_LOGS", text = serializeJSON(payload));
        },

        toolsChangeConsumer: function(tools) {
            application.mcpToolsCache = tools;
            writeLog(file = "MCP_CHANGES", text = "Tools changed: " &
arrayLen(tools));
        }
    };
    clients    = MCPClient(configData);
    mcpClient = clients[1];
</cfscript>
```

Multiple servers

```
<cfscript>
    var configData = {
        // Multi-server JSON descriptor
        configFile: getDirectoryFromPath(getCurrentTemplatePath()) &
"mcpServers.json",

        // Shared client identity for all MCP servers in the file
        clientInfo: {
            name    : "Healthcare MCP Client",
            version : "1.0.0"
        },

        // Shared capabilities
        capabilities: {
            sampling    : true,
            roots       : true,
            elicitation : false
        },

        // Shared timeouts
        initializationTimeout : 50,
        requestTimeout        : 50,

        // Optional shared consumers
        samplingConsumer: function(messages) {
            writeLog(file = "MCP_SAMPLING", text = serializeJSON(messages));
        },

        elicitationConsumer: function(messages) {
            application.lastElicitation = messages;
            writeLog(file = "MCP_ELICITATION", text = serializeJSON(messages));
        },

        loggingConsumer: function(logEvent) {
            var payload = structKeyExists(logEvent, "data") ? logEvent.data :
logEvent;
```

```
            writeLog(file = "MCP_LOGS", text = "MCP log: " &
serializeJSON(payload));
        },

        toolsChangeConsumer: function(tools) {
            writeLog(file = "MCP_CHANGES",
                     text = "Tools changed. Count: " & arrayLen(tools));
        },

        resourcesChangeConsumer: function(resources) {
            writeLog(file = "MCP_CHANGES",
                     text = "Resources changed. Count: " & arrayLen(resources));
        }
    };

    // Create one client per server defined in mcpServers.json
    var clients = MCPClient(configData);
    application.mcpClients = clients;  // store for later use
</cfscript>
```

listTools

Description

The listTools function returns the list of all tools exposed by the connected MCP server for a given MCP client. Each tool entry includes metadata such as the tool's name, description, and input schema. This is typically the first method you call after creating an MCP client, to discover what capabilities the server provides.

Returns

An array of structs, where each struct describes a tool.

Category

MCP functions

Syntax

listTools()

Called as an instance method on an MCP client object:

tools = mcpClient.listTools()

History

New in ColdFusion 2025.1

Parameters

None

Example

```
<cfscript>
    transport = {
        "type": "http",
        "url": "https://gitmcp.io/langchain-ai/langgraph"
    }
    mcpClientConfig = {
            "transport": transport,
            "requestTimeout": 30,
            "initializationTimeout": 10
    }
    mcpclient = MCPClient(mcpClientConfig);

    myprompts = mcpClient[1].listtools()
    toolsArray=myprompts.tools

    writeOutput("<b>List of tools</b>" & "<br>")

    for (p in toolsArray) {
        writeOutput(p.name & "<br>")
    }
</cfscript>
```

Output

```
List of tools
fetch_langgraph_documentation
search_langgraph_documentation
search_langgraph_code
fetch_generic_url_content
```

callTool

Description

The callTool function invokes a single MCP tool by name on the connected MCP server, passing structured arguments and returning the tool's response. It is the primary way for your ColdFusion application to execute MCP tools.

Returns

A struct representing the JSON-RPC result.

Category

MCP functions

Syntax

callTool(string toolName, struct config)

Parameters

Field	Data type	Required	Description
`toolName`	String	Yes	The name of the tool to invoke. This must match one of the tool names returned by listTools() for that MCP server.
`config`	Struct	Yes	A struct of arguments to pass to the tool. Keys and types must match the tool's inputSchema as described in its metadata. Example argument struct for get_weather: { location = "New York" }

listResources

Description

The listResources function retrieves the list of all resources exposed by the connected MCP server for the current MCP client. Each resource entry describes an externally accessible piece of read-only data (files, documents, logs, custom URIs, etc.) that the server makes available, typically scoped by roots if your client has configured them.

Returns

An array of structs, where each struct describes a resource.

Category

MCP functions

Syntax

listResources()

Called as an instance method on an MCP client object:

resources = mcpClient.listResources();

History

New in ColdFusion 2025.1

Parameters

None

readResource

Description

The readResource function retrieves the full content of a specific resource from the MCP server, identified by its URI. This method enables your ColdFusion application to access external files, documents, logs, or other read-only resources exposed by the server, provided you have permission. The returned result includes the resource content and metadata such as MIME type and encoding.

Returns

A struct containing the resource content and metadata.

Category

MCP functions

Syntax

readResource(string uri)

Called as an instance method on an MCP client:

resourceData = mcpClient.readResource(resourceUri);

History

New in ColdFusion 2025.1

Parameters

Field	Data type	Required	Description
`uri`	String	Yes	The unique URI of the resource to retrieve, e.g. "file:///logs/app.log" or "healthcare://patient-data/P001/ct-scan".

listPrompts

Description

The listPrompts function retrieves the list of all prompts that the MCP server exposes to your client. Each prompt typically describes a reusable AI interaction pattern, such as code review, summarization, or workflow step, along with its required and optional arguments.

Returns

An array of structs, each describing a prompt with its key metadata.

Syntax

listPrompts()

Call as an instance method on your MCP client:

prompts = mcpClient.listPrompts();

History

New in ColdFusion 2025.1

Parameters

None

getPrompt

Description

The getPrompt function retrieves the full, resolved definition of a specific prompt from the MCP server. While listPrompts() gives you a catalog of prompt names, titles, descriptions, and argument definitions, getPrompt lets you:

- Fetch a prompt by name.
- Optionally provide argument values.
- Receive a fully constructed prompt payload (often including ready-to-use messages) that your application or AI service can send directly to a model.

Returns

A struct representing the JSON-RPC result of prompts/get.

Syntax

function getPrompt(struct params)

Called as an instance method on an MCP client object:

promptDef = mcpClient.getPrompt(params);

History

New in ColdFusion 2025.1

Parameters

Field	Data type	Required	Description
`name`	String	Yes	Name of the prompt to fetch (must match one of the names returned by listPrompts()).
`params`	Struct	No	Key/value pairs for prompt arguments (e.g. { code = "..." }); used to fill the template or generate messages.

addRoot

Description

The addRoot function registers a root with your MCP client at runtime. A root is a named URI that provides instruction to the MCP server.

Roots are used to:

- Scope which resources the client can see (security & least privilege).
- Give the server hints about relevant directories, repositories, or data domains.
- Improve performance and relevance when listing or reading resources.

Returns

Void

Syntax

addRoot(struct root)

Called as an instance method on an MCP client:

mcpClient.addRoot(root)

History

New in ColdFusion 2025.1

Parameters

Field	Data type	Required	Description
`name`	String	Yes	Logical name/label for this root.
`uri`	String	Yes	URI for the root (e.g. file:// or custom schemes).

getRoot

Description

The getRoot function retrieves the definition of a single root previously associated with an MCP client. A root is a named URI that tells the server where to look for relevant resources (for example, a directory, repository, or logical data domain).

Use getRoot when you need to:

- Inspect which root a given name resolves to (e.g., "samples" → file:///home/user/samples).
- Validate that a particular root has been registered before using it in your own logic.

Returns

A struct describing the root if found, or null / undefined (depending on implementation) if no root with that name exists.

Syntax

getRoot(string name)

Called as an instance method on an MCP client:

```
root = mcpClient.getRoot("samples");
```

History

New in ColdFusion 2025.1

Parameters

Field	Data type	Required	Description
`name`	String	Yes	Logical name/label for this root.

removeRoot

Description

The removeRoot function removes a previously registered root from your MCP client's configuration. A root is a named URI that tells the MCP server which locations are in scope for resources. Removing a root:

- Narrows or changes the set of resources the client can see.
- Is useful when a directory, repository, or data domain should no longer be accessible in each session.
- Helps enforce least-privilege and dynamic scoping (e.g., per user, per project).

Returns

Boolean

- True: if a root with the given name was found and removed.
- False: if no root with that name existed and thus nothing was changed.

Syntax

removeRoot(string name)

Called as an instance method on an MCP client:

removed = mcpClient.removeRoot("testcases");

History

New in ColdFusion 2025.1

Parameters

Field	Data type	Required	Description
`name`	String	Yes	Logical name of the root to remove.

listRoots

Description

The listRoots function returns the list of all roots currently registered on an MCP client. A root is a named URI. Listing roots is useful for:

- Inspecting what resource scopes are active for a client.
- Debugging why certain resources are (or are not) visible.

Returns

An array of root structs, each describing one root.

Syntax

listRoots()

Called as an instance method on an MCP client:

roots = mcpClient.listRoots();

History

New in ColdFusion 2025.1

Parameters

None

rootsListChangedNotification

Description

The rootsListChangedNotification function is a callback that your application implements and passes into the MCP client configuration. It is invoked whenever the MCP client detects that its list of roots has changed.

You use this notification to:

- Keep an up-to-date, in-memory list of active roots.
- Update UI elements or admin pages that show which directories or data domains are currently in scope.
- Log and audit changes to resource access scope.
- Trigger recalculation or cache invalidation for resource-dependent features.

Returns

Void

Syntax

rootsListChangedNotification(array roots)

History

New in ColdFusion 2025.1

Parameters

Field	Data type	Required	Description
`roots`	Array	Yes	The current list of root descriptors after the change.

MCPServer

Description

The MCPServer(config) function creates and initializes a ColdFusion-based MCP server from a configuration struct. This server exposes CFML tools (CFC methods), prompts (templates), and resources (data like files or documents) via the Model Context Protocol (MCP) so that MCP clients (e.g., ColdFusion apps, Cursor, Claude Desktop) can call them using standard MCP methods like tools/list, tools/call, prompts/get, and resources/read.

Returns

any

The returned object represents the MCP server and is normally:

- Stored in application scope (for example, application.mcpServer) so it can serve JSON-RPC requests.
- Used internally by your HTTP or stdio entry points to handle incoming MCP calls (initialize, tools/list, tools/call, prompts/list, resources/read, etc.).

Syntax

MCPServer(struct config)

Called in CFML as:

mcpServer = MCPServer(config);

History

New in ColdFusion 2025.1

Parameters

Field	Data type	Required	Description

`serverInfo`	Struct	Yes	Identifies the MCP server.
			serverInfo = {
			  name : "ColdFusion Healthcare Server",
			  version : "1.0"
			};
			• name: human-readable name of the MCP server.
			• version: the version of the MCP server.

`capabilities`	Struct	Yes	Declares what the MCP server supports (tools, prompts, resources, logging, etc.).
			`capabilities = {`
			`    tools      : true,`
			`    prompts    : true,`
			`    resources : true,`
			`    serverInfo : serverInfo`
			`// may embed or mirror`

			`serverInfo`
			`};`
			• `tools: whether this server exposes tools.`
			• prompts: whether it exposes prompts.
			• resources: whether it exposes resources.
			• serverInfo: nested server info, depending on implementation.

`tools`	Array of structs	No	Lists the CFCs that implement MCP tools. Each entry describes a CF component that contains one or more public functions to be exposed as tools.
			`tools = [`
			`    { cfc : "mcp.tools.WeatherTools" },`
			`    { cfc : "mcp.tools.HealthcareTools" }`
			`];`
			• `cfc: the fully-qualified CFC name (path resolvable via CF mappings).`

`prompts`	Array of structs	No	Defines prompts that the server exposes for prompts/list and prompts/get.
			`prompts = [`
			`    {`
			`        name        : "generate_discharge_summary",`
			`        title       : "Discharge Summary",`
			`        description : "Generate a patient discharge summary.",`
			`        arguments   : [ { name : "patientName", required : true } ],`
			`        template    : "Generate a detailed discharge summary for patient {patientName}."`
			`    }`
			`];`
			• `Name: prompt name (used in prompts/get).`

			• title: user-facing title of all prompts. • description: what the prompt does. • arguments: array of { name, required } or richer schema. • template: prompt text with placeholders (e.g., {patientName}).

`resources`	Array of structs	No	Defines resources the MCP server exposes via resources/list and resources/read.
			resources = [
			  {
			    uri : "healthcare://patient-data/P001/ct-scan",
			    name : "ct_scan",
			    title : "CT Scan - P001",
			    description : "Chest CT scan for patient P001",
			    mimeType : "application/pdf",
			    readResourceHandler = function(req) {
			      return fileReadBinary(expandPath("/data/ct-scan-P001.pdf"));
			    }
			  }
			];
			• uri: resource identifier (custom scheme or file://).
			• name: short name.
			• title: human-friendly title.
			• description: description.
			• mimeType: MIME type (application/pdf, text/plain, etc.).
			• readResourceHandler: CF function called to produce the content when resources/read is invoked.

`cfcCaching`	Boolean	No	Controls whether CFCs used for tools are cached for performance. `cfcCaching = true` • `true: reuse CFC instances instead of re-instantiating for each call.`

			• false: instantiate fresh CFCs per call (slower, but can be useful for debugging or if you need truly stateless behavior).

handleRequestAndWriteResponse

Description

The handleRequestAndWriteResponse function is a server-side helper you implement (typically in a .cfm or remote CFC method) that:

1. Reads the incoming JSON-RPC request for MCP (from getHttpRequestData().content or similar).
2. Forwards the request struct to your MCP server instance (created via MCPServer(config)) to be handled.
3. Serializes the MCP server's response back to the client and writes it as an HTTP response with application/json content type.

It effectively glues your ColdFusion HTTP endpoint to the MCP server object so external MCP clients (Cursor, Claude Desktop, other CF apps) can call initialize, tools/list, tools/call, prompts/get, resources/read, etc. over HTTP.

Returns

Void

Syntax

handleRequestAndWriteResponse()

For example,

handleRequestAndWriteResponse() access="remote" returntype="void" output="false"

handleRequestAndWriteResponse is implemented in the same file or via a reusable function in a shared CFC.

History

New in ColdFusion 2025.1

Parameters

None