

INTEGRATION GUIDE

CalTool AI Integration Guide

Contents

1. Copyright Notice . . . 6
2. Introduction . . . 7
    2.1. Symbols Description . . . 8
    2.2. Contact Information . . . 8
3. MCP in brief . . . 9
    3.1. The problem it solves . . . 9
    3.2. The three roles . . . 9
    3.3. What a server offers . . . 10
    3.4. What it looks like on the wire . . . 10
    3.5. Revisions . . . 11
    3.6. Going further . . . 11
4. Connecting your own LLM . . . 12
    4.1. The arrangement . . . 12
    4.2. Three ways to build the host . . . 12
    4.3. What your model needs . . . 12
    4.4. If the host cannot run on the CalTool PC . . . 13
5. Quick start . . . 14
    5.1. Switch AI access on . . . 14
    5.2. Call it by hand . . . 14
    5.3. Add a model . . . 15
    5.4. If it does not work . . . 16
6. The protocol interface . . . 17
    6.1. Endpoint . . . 17
    6.2. Authentication . . . 17
    6.3. Origin check . . . 17
    6.4. Protocol revisions . . . 18
    6.5. Requests in the current revision . . . 18
    6.6. Methods . . . 20
    6.7. Tool results . . . 20
    6.8. Errors . . . 20
    6.9. Cancelling and timing . . . 21
    6.10. The stdio bridge . . . 22
7. The data model . . . 23
    7.1. Keys . . . 23
    7.2. Kinds of object . . . 23
    7.3. Calibration values . . . 24
    7.4. Time and data . . . 25
    7.5. What the engineer sees . . . 26
8. Permissions and safety . . . 27
    8.1. Access tiers . . . 27
    8.2. What happens when a model writes calibration . . . 28

8.3. The CAN bus, diagnostics and flashing . . . 30
8.4. The token . . . 31
8.5. Prompt injection . . . 31
8.6. What is recorded . . . 31
8.7. Recommended practice . . . 32
9. Tool reference . . . 33
9.1. Read tools . . . 33
9.2. Analyse tools . . . 42
9.3. Workspace tools . . . 47
9.4. Calibration tools . . . 56
9.5. Live ECU tools . . . 58
9.6. CAN trace tools . . . 58
9.7. CAN transmit tools . . . 63
9.8. Diagnostics tools . . . 64
9.9. Flash tools . . . 67
9.10. Resources and prompts . . . 69
10. Typical sequences . . . 71
10.1. Finding out what is there . . . 71
10.2. Analysing a recording . . . 71
10.3. Investigating an event . . . 71
10.4. Zooming in on what was found . . . 72
10.5. Tabulating occurrences . . . 73
10.6. Building a computed channel . . . 73
10.7. Taking a recording . . . 74
10.8. Changing a calibration value . . . 74
10.9. Watching and stimulating a CAN bus . . . 75
10.10. Decoding CAN traffic in a log . . . 76
10.11. A diagnostic session . . . 77
10.12. Flashing an ECU . . . 77
11. Getting good results . . . 79
11.1. The system prompt . . . 79
11.2. Offering fewer tools . . . 79
11.3. Keeping results small . . . 80
11.4. When the model struggles . . . 81
12. What the interface does not do . . . 82
12.1. Not available to a model . . . 82
12.2. Limits of what is there . . . 82
Appendix A: A reference host in Python . . . 84

Document Information

Document Number	100-0-0006
Created By	Alexander Olsson
Date	2026-11-22
Revision	Rev. 00.04
Classification	EXTERNAL
Applies to	Altech CalTool (AI integration over the Model Context Protocol)

Revision History

Revision	Date	Author	Change
00.01	2026-11-03	Alexander Olsson	Initial release
00.02	2026-11-12	Alexander Olsson	`get_state` returns a summary, with detail by `sections`. `get_signal_data` and `find_events` answer the keys they can and list the rest under `failed`. `find_events` takes several keys. New tools `get_snapshot` and `scan_logs`. The data tools read what CalTool shows (a log opened after a recording is no longer hidden by it), take `source`, and say what they read in `dataSource`. New tools `get_signal_grid` and `summarize_events`; condition `equals`; `list_signals` takes `namesOnly`; `get_state` gives the log file's own time span and time window.
00.03	2026-11-18	Alexander Olsson	Keys: a signal decoded from a log's CAN frames is keyed under its DBC (`pwt_rear/EngineSpeed`); a bare name that belongs to several signals, or a key nothing has, is refused by every tool, `open_plot` and `add_to_plot` included. New tool `configure_log_can_decoding`, with a sequence for it; `get_can_frames` reads a log's or a file's frames and checks alive counters and CRCs; decoded signals carry their CAN origin in `list_signals` and `describe_signal`. New tool `read_ecu_calibration`. `get_calibration` takes `changedOnly`, `full`, `includeStale` and `offset`, and gives `origin`, `changed`, `changedBy`, `changedAt` and summaries of large tables. `set_calibration` reads online writes back (`onEcu`, `ecuHolds`) and takes `partial` and `force`. `get_live_values` reads a key that does not stream once from the online ECU. The data tools take `file` and give `samplePeriod`; `get_state` lists saved recordings and the log's CAN buses; `summarize_events` takes `followUps`; `set_raster` says when it takes effect.

© 2026 Altech Electronics Nordic AB. All rights reserved.

Revision	Date	Author	Change
00.04	2026-11-22	Alexander Olsson	New tools `get_plot_view` and `set_plot_view`: an assistant zooms a plot's time axis (a window, a span around an instant, everything, or between the cursors), fits or sets the Y axes, hides, pins, colours and selects signals, and brings the plot on screen and maximizes it for `render_plot`. `set_cursor␣s time window now moves plots that are on screen. ` `add_to_plot` takes `subplot`. New sequence Zooming in on what was found. CalTool is free of charge; error reports and suggestions go to the support address.

© 2026 Altech Electronics Nordic AB. All rights reserved.

1. Copyright Notice

This document is the property of Altech Electronics Nordic AB. The document may not be edited, utilized, reproduced or otherwise exploited, in full or in part, without the written permission of Altech Electronics Nordic AB.

2. Introduction

Altech CalTool can be driven by an AI assistant. An assistant reads the signals, logs and calibration data loaded in the running application, analyses them, and — as far as the engineer at the bench allows — changes the workspace and the calibration, works on the CAN bus, runs diagnostic sessions and reprograms ECUs. This guide is for engineers who connect their own large language model (LLM) to CalTool rather than a ready-made assistant. Your model may run on the same PC, on a server inside your company, or at a provider of your choice.

CalTool does not contain or require a model, and it is not tied to one vendor. It offers an interface in the Model Context Protocol (MCP), an open standard. Anything that speaks MCP can use it, and a program of about 250 lines is enough to connect any model that supports tool calling. Appendix A lists one.

To	Read
Understand MCP in ten minutes	Chapter 3
Decide how your model connects	Chapter 4
See it work on your PC	Chapter 5
Implement against the interface	Chapter 6, Chapter 7, Chapter 9
Assess the risk and set the rules	Chapter 8
Get good results from your model	Chapter 10, Chapter 11
Know what it cannot do	Chapter 12

The CalTool User Guide (100-0-0004) describes CalTool itself and how an engineer works with an assistant. This guide describes the interface underneath and does not repeat how to calibrate or log.



This guide describes the interface as of the revision on the cover. To check that your CalTool has it, send `server/discover` (Chapter 5): the answer lists protocol revision `2026-07-28`, and `tools/list` returns 87 tools. An older CalTool answers only the `initialize` handshake and has fewer tools. The current release is free of charge and is downloaded from www.altechelectronics.io.

The requests and responses in this guide come from real sessions with CalTool on its built-in **Simulation** channel, using a small demonstration ECU. Long answers are cut where marked with ⋯, long numbers are rounded, and some notes and values are reworded for the example. Object names such as `Engine/RevLimit` belong to that demonstration and contain no customer data.

© 2026 Altech Electronics Nordic AB. All rights reserved.

2.1. Symbols Description

The following symbols are used in this integration guide to highlight certain information.



Critical information — the requirements must be followed in order to not affect the performance of the system or product.



Recommended information — the requirements should be followed in order to not affect the performance of the system or product.



Additional information — further information can be found in a referenced Altech guideline or datasheet. It's recommended to study all referenced documents before installing any Altech system or product.

2.2. Contact Information

For information or sales inquiries, please contact info@altechelectronics.io.

For support, and to report an error or suggest an improvement, write to support@altechelectronics.io. Every report is read.

© 2026 Altech Electronics Nordic AB. All rights reserved.

3. MCP in brief

This chapter gives you enough to follow the rest of the guide. Section 3.6 says where to go deeper.

3.1. The problem it solves

An LLM produces text. It cannot open a file or read a sensor. Most models can, however, be told which functions exist and answer with a structured request to call one: this is called *tool calling*. Some program must then run the function and hand the result back to the model.

Without a standard, every pairing of an AI application and a tool needs its own adapter. MCP defines one interface between them, so a tool that implements it works with every application that implements it. It is an open standard, first published by Anthropic in 2024 and now implemented by many vendors and open-source projects.

3.2. The three roles

Host

The application that holds the conversation and calls the model. A chat application, an agent framework, or a program you write.

Client

The part of the host that speaks MCP to one server.

Server

The program that offers capabilities. Here, CalTool.

The model is none of these. **It never connects to CalTool and never sees the protocol.** It sees a list of tool descriptions and returns tool calls, and the host does the rest. This is the most important point for your design: any model with tool calling can work with MCP, including one that has never heard of it.

© 2026 Altech Electronics Nordic AB. All rights reserved.

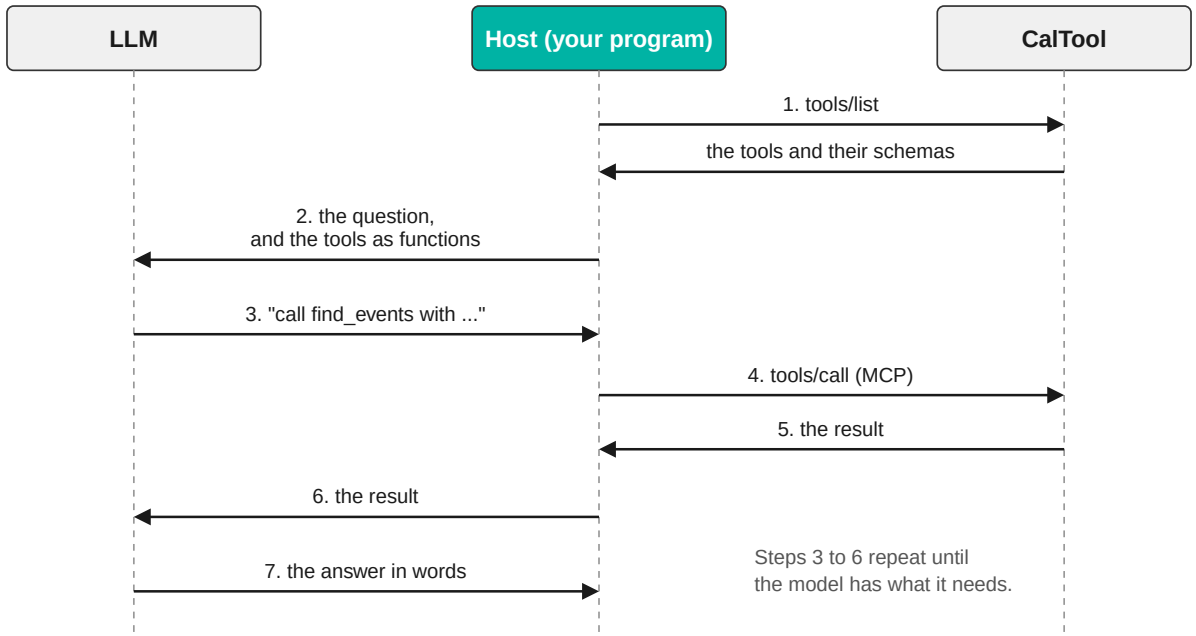

Figure 1. One question, one tool call

3.3. What a server offers

Tools

Functions the model may call, each with a name, a description and a JSON Schema for its arguments. This is where nearly all of CalTool's interface lives.

Resources

Documents the host may read and give to the model. CalTool offers its guides and the session state.

Prompts

Ready-made task descriptions the host may offer to its user, such as "analyse the loaded log".

3.4. What it looks like on the wire

MCP messages are JSON-RPC 2.0: small JSON objects with a method and parameters, answered by a result or an error. They travel over one of two transports:

Streamable HTTP

Each message is an HTTP POST to one URL. CalTool's server speaks this.

stdio

The host starts the server as a child process and exchanges one message per line over its standard input and output. CalTool ships a small bridge program for hosts that can only do this (Section 6.10).

© 2026 Altech Electronics Nordic AB. All rights reserved.

3.5. Revisions

The specification is revised by date. Two generations are in use, and you will meet both when reading about MCP:

- Up to `2025-11-25`, a connection opens with an `initialize` handshake that agrees a revision and may create a session.
- From `2026-07-28`, there is no handshake. Every request states its revision and capabilities, and `server/discover` says what a server offers.

CalTool's server speaks both on the same port, so you can use whichever your software supports (Section 6.4).

3.6. Going further

- The specification and introduction: modelcontextprotocol.io. Start with its *Architecture* page. For this guide's server, the *Tools* and *Streamable HTTP* sections of the specification are the ones that matter.
- Official software development kits (SDKs) exist for Python, TypeScript, C#, Java and other languages, linked from the same site. Use one if you build more than a small host.
- MCP Inspector, the project's debugging tool, lets you list and call CalTool's tools by hand with no model involved. It is linked from the same site.
- Your LLM server's documentation on *tool calling* or *function calling* says how well a given model supports it. This matters more to the result than anything in MCP (Section 4.3).

© 2026 Altech Electronics Nordic AB. All rights reserved.

4. Connecting your own LLM

4.1. The arrangement

CalTool listens only on the local address `127.0.0.1`. The host therefore runs on the CalTool PC, and the model can run wherever you like, because only the host talks to it.

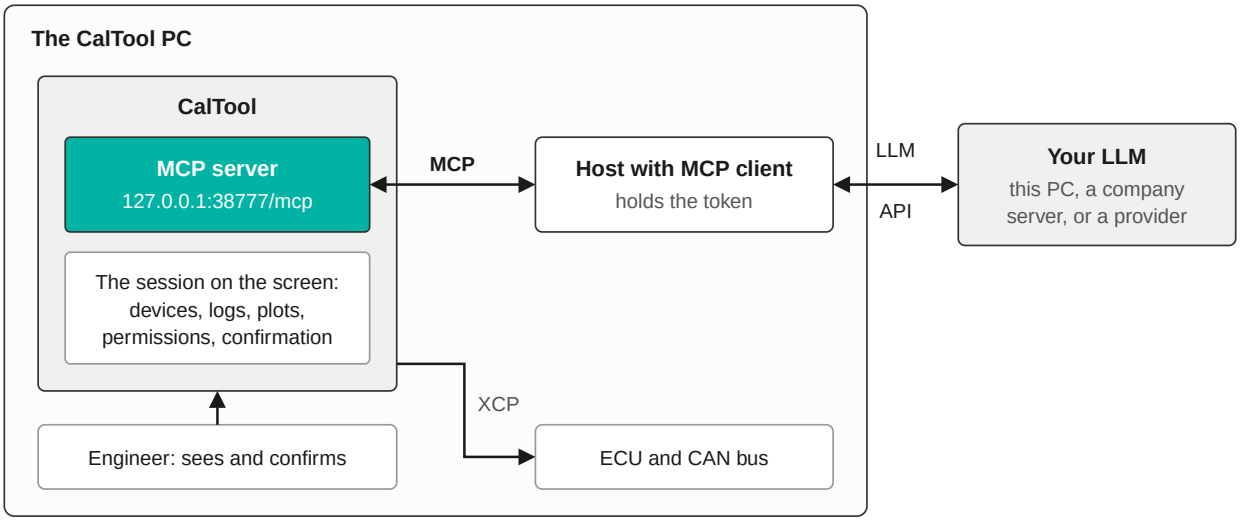

Figure 2. Where each part runs

What this means for your data: the host sends the model whatever CalTool returns, which can include signal names, values, calibration data, file paths and plot images. That data stays inside your company exactly as far as your model does. CalTool itself sends nothing to any AI service.

4.2. Three ways to build the host

Option	Suits	What you do
An existing application with MCP support	Trying things quickly, and interactive chat for engineers	Point it at your model and at CalTool. It must support Streamable HTTP with a custom `Authorization` header, or start a stdio server (Section 6.10).
An SDK	A product of your own: an agent, a test bench workflow, a company tool	Use an official MCP SDK for the client side and your LLM's client library for the model side.
A small program of your own	Isolated PCs, full control of what is sent, and learning how it works	Start from Appendix A, which needs only Python.

The official Python SDK, in versions 1.30 and 2.3, was tested against CalTool's server in each mode it offers (Section 6.4).

4.3. What your model needs

© 2026 Altech Electronics Nordic AB. All rights reserved.

Tool calling

This is the one hard requirement. The server must return structured tool calls, and the model must produce valid arguments reliably. Models differ greatly here, and a capable model can behave poorly when its server's chat template handles tools badly. Test it with the tasks in Chapter 10 before judging.

Context size

The list of all 87 tools is about 84 KiB of JSON, and the 18 read and 5 analyse tools alone about 31 KiB. Allow for that in every request, plus the results. A model with at least 32,000 tokens of context leaves working room; below that, offer fewer tools (Chapter 11). Treat these figures as starting points, as they depend on your model and task.

Parallel calls

Optional. If your model asks for several calls at once, run them one after another unless you know they are independent.

Images

Optional. Only `render_plot` returns a picture. A model without vision can do all its analysis from the numbers.

Deterministic settings

Use a low temperature. A calibration assistant should not be creative with numbers.



Whichever model you use, it is only as right as its reasoning, and it can state wrong values confidently. Calibration results and any change to an ECU remain the responsibility of the engineer. See Chapter 8.

4.4. If the host cannot run on the CalTool PC

CalTool binds to the local address deliberately, and there is no setting to change that. Running the host on the same PC is the recommended arrangement. If your host must run elsewhere, you have to bring the port to it yourself, for example with an SSH tunnel that you start from the CalTool PC:

```
ssh -N -R 38777:127.0.0.1:38777 user@host-machine
```

The host then uses `http://127.0.0.1:38777/mcp` on its own machine. Anyone able to use that forwarded port and holding the token can do what the permissions allow.



Do not expose the port on a network with a port forward or a public tunnel unless you have assessed it as your own security decision. The token is then the only protection, and it travels without encryption wherever the connection is not itself encrypted. Section 8.4 explains how the token works.

© 2026 Altech Electronics Nordic AB. All rights reserved.

5. Quick start

This takes about fifteen minutes and needs no hardware and no model for the first part. You will talk to CalTool by hand first, so that you can see the interface before a model is involved.

5.1. Switch AI access on

1. Start CalTool and load an A2L file with **Add ECU…**. Choose the **Simulation** channel so that no hardware is needed.
2. Open **Settings** (the gear at the bottom of the activity bar) > **AI access** > **Configure…**.
3. Tick **Let an AI assistant use this CalTool**. The status line then reads `Listening on http://127.0.0.1:38777/mcp`.
4. Copy the token. Pick **Cursor, VS Code, Gemini CLI, other** under **Client**: the text box shows it after `Bearer`.

Leave the other permissions as they are for now. Reading and analysing are always allowed while the switch is on.

5.2. Call it by hand

Open Windows PowerShell and define one small function, with your token in place of the placeholder:

```
$url = "http://127.0.0.1:38777/mcp"
$token = "<your token>"

function Invoke-CalTool($method, $params = @{}, $name) {
    $params["_meta"] = @{
        "io.modelcontextprotocol/protocolVersion"    = "2026-07-28"
        "io.modelcontextprotocol/clientCapabilities" = @{}
    }
    $headers = @{
        Authorization          = "Bearer $token"
        "MCP-Protocol-Version" = "2026-07-28"
        "Mcp-Method"           = $method
    }
    if ($name) { $headers["Mcp-Name"] = $name }
    $body = @{ jsonrpc = "2.0"; id = 1; method = $method; params = $params } |
        ConvertTo-Json -Depth 20
    Invoke-RestMethod $url -Method Post -Headers $headers `
        -ContentType "application/json" -Body $body
}
```

Ask what the server speaks and which tools it has:

© 2026 Altech Electronics Nordic AB. All rights reserved.

```
(Invoke-CalTool "server/discover").result.supportedVersions
(Invoke-CalTool "tools/list").result.tools.name
```

Ask what is loaded. This is the same call a model makes first, and everything you see here is what a model would see:

```
$state = Invoke-CalTool "tools/call" @{
    name = "get_state"; arguments = @{}
} "get_state"
$state.result.structuredContent.devices |
    Format-Table name, kind, online, measurements
```

```
name   kind      online measurements
----   ----      ------ ------------
Engine ecu (A2L)  False           17
```

Search for an object:

```
$found = Invoke-CalTool "tools/call" @{
    name = "list_signals"; arguments = @{ query = "Rev*"; limit = 5 }
} "list_signals"
$found.result.structuredContent.signals | Format-Table key, kind, unit
```

```
key             kind           unit
---             ----           ----
Engine/RevLimit characteristic rpm
```

Now look at CalTool: the **AI** chip in the status bar flashed on each call, and the **Activity** list in the dialog shows them. An assistant never works unseen.

5.3. Add a model

Save the program in Appendix A as `caltool_llm_host.py`. It needs Python 3.9 or newer and nothing else. Point it at your LLM server's OpenAI-compatible address and name the model:

```
$env:CALTOOL_MCP_TOKEN = "<your token>"
python caltool_llm_host.py --llm-url http://localhost:8000/v1 `
    --model my-model `
    "What is loaded in CalTool right now? Which objects limit engine speed?"
```

© 2026 Altech Electronics Nordic AB. All rights reserved.

On standard error it prints each tool call as it happens, and on standard output the model's answer. By default it offers the model only the read and analyse tools, so it cannot change anything. Add `--tiers read,analyse,workspace` when you want it to place markers and open plots.

If your LLM server needs a key, set it in the environment variable `LLM_API_KEY`.



Altech tested this program's protocol handling against CalTool, and its model loop against a scripted stand-in for an LLM server. How well it works with your model depends on that model's tool calling, which is why Chapter 10 suggests tasks to try first.

5.4. If it does not work

What you see	What to do
Connection refused	CalTool is not running, or AI access is off. The dialog's status line says which.
`401 Unauthorized`	The token is wrong, or **New token** was pressed. Copy it again.
The status line says the port is in use	Change **Port** in the dialog and use the new port in your URL.
`403 Forbidden`	The request carries a web `Origin` that is not allowed (Section 6.3). This happens with clients that run in a browser.
`400` with code `-32020`	The routing headers do not agree with the body (Section 6.5).
A tool answers that a permission is missing	This is working as designed. The message names the box to tick in the dialog.

© 2026 Altech Electronics Nordic AB. All rights reserved.

6. The protocol interface

This chapter is the exact description of how to reach CalTool's server and what travels over the connection. The tools themselves are in Chapter 9.

6.1. Endpoint

Transport	Streamable HTTP, HTTP/1.1 without TLS, on the loopback interface only
Address	`http://127.0.0.1:38777/mcp`. The port is set in the AI access dialog, from 1024 to 65535.
Starts	When AI access is switched on, and with CalTool if it was on at the last exit. It stops when CalTool closes or the switch goes off.
Scope	One running CalTool, one session: the one on the screen. There is no headless mode, and a second CalTool on the same PC needs another port.
Settings	`%APPDATA%\CalTool\settings.json`, object `AiAccess`. Change them in the dialog.

6.2. Authentication

Every request must carry the token, in one of two ways. Use the first.

```
Authorization: Bearer <token>
```

The second puts the token in the path as `/mcp/<token>`. It exists for clients that accept only a URL, and it is weaker because URLs end up in logs.

A wrong or missing token is answered with `401 Unauthorized` and a `WWW-Authenticate: Bearer` header before anything else is looked at. MCP's OAuth authorization flow is not implemented, since the server is reached only from the same PC.

6.3. Origin check

A web page in a browser on the PC can reach a loopback port, so a request with an `Origin` header is refused with `403 Forbidden` unless that origin is one of these:

- The PC itself: `localhost`, `127.0.0.1` or `[::1]`, on any port.
- `https://chatgpt.com` or `https://chat.openai.com`, for the ChatGPT connector.
- An origin listed in `AllowedOrigins` in the `AiAccess` object of `settings.json`, written as `https://host` or `https://host:port`. There is no control for this in the dialog.

A program that is not a web page sends no `Origin`, and this check does not apply to it. Your own host does not need it unless its user interface calls CalTool directly from a browser.

© 2026 Altech Electronics Nordic AB. All rights reserved.

6.4. Protocol revisions

The server speaks two generations on the same address. Nothing depends on an earlier request, so it tells them apart message by message.

	Current	Handshake
Revisions	`2026-07-28`	2025-11-25, 2025-06-18, 2025-03-26, 2024-11-05
How to start	Send any request. `server/discover` is the usual first one.	Send `initialize`, then the `notifications/initialized` notification.
Where the revision is stated	In every request's `_meta`, and in the `MCP-Protocol-Version` header	In `initialize`, then in the `MCP-Protocol-Version` header. The header is optional here.
`ping`, `logging/setLevel`	Unknown methods	Accepted. The log level is ignored, as the server sends no log messages.
Session	None	The `Mcp-Session-Id` response header is sent, but need not be returned. One value serves the whole server run, and `DELETE` has no effect.
Unknown method	`404` with code `-32601`	`200` with code `-32601`
Unknown resource	Code `-32602`	Code `-32002`

`initialize` answers with the revision you asked for if the server has it, and otherwise with `2025-11-25`. A client that asks `initialize` for `2026-07-28` also gets `2025-11-25`, since that revision has no handshake.

A revision the server does not know, in the header or in `_meta`, is answered with `400` and code `-32022`, listing what it supports:

```
{"jsonrpc":"2.0","id":6,"error":{"code":-32022,
 "message":"Unsupported protocol version",
 "data":{"supported":["2026-07-28","2025-11-25","2025-06-18",
                      "2025-03-26","2024-11-05"],
         "requested":"1900-01-01"}}}
```

Unless your software dictates otherwise, use the current revision: it has less to get wrong.

6.5. Requests in the current revision

One JSON-RPC message per POST. A batch is refused. Send these headers beside `Authorization`:

Header	Value	On
`Content-Type`	`application/json`	Every POST. Otherwise `415`.
`MCP-Protocol-Version`	`2026-07-28`, the same as in `_meta`	Every POST

© 2026 Altech Electronics Nordic AB. All rights reserved.

Header	Value	On
`Mcp-Method`	The message's `method`	Every request
`Mcp-Name`	The tool or prompt `name`, or the resource `uri`	`tools/call`, `prompts/get`, `resources/read`

A value that would not survive HTTP may be sent as `=?base64?<base64 of the UTF-8 text>?=`. Every request's `params` carries `_meta`:

```
"_meta": {
  "io.modelcontextprotocol/protocolVersion": "2026-07-28",
  "io.modelcontextprotocol/clientInfo": {"name": "my-host", "version": "1.0"},
  "io.modelcontextprotocol/clientCapabilities": {}
}
```

`clientInfo` is optional and is not used. A missing or disagreeing header is answered with `400` and code `-32020`. A `_meta` without `clientCapabilities` is answered with `400` and code `-32602`.

A complete request, then, looks like this:

```
POST /mcp HTTP/1.1
Host: 127.0.0.1:38777
Authorization: Bearer <token>
Content-Type: application/json
MCP-Protocol-Version: 2026-07-28
Mcp-Method: tools/call
Mcp-Name: list_signals

{"jsonrpc":"2.0","id":3,"method":"tools/call","params":{"name":"list_signals",
 "arguments":{"query":"Rev*","limit":5},
 "_meta":{"io.modelcontextprotocol/protocolVersion":"2026-07-28",
          "io.modelcontextprotocol/clientCapabilities":{}}}}
```

And the answer, shown wrapped:

```
{"jsonrpc":"2.0","id":3,"result":{
 "content":[{"type":"text",
             "text":"{\"total\":1,\"offset\":0,\"returned\":1,...}"}],
 "isError":false,
 "structuredContent":{"total":1,"offset":0,"returned":1,
   "signals":[{"key":"Engine/RevLimit","kind":"characteristic",
               "shape":"value · rpm","unit":"rpm"}],
   "summary":"1 match(es), 1 returned"},
 "resultType":"complete",
 "_meta":{"io.modelcontextprotocol/serverInfo":
```

© 2026 Altech Electronics Nordic AB. All rights reserved.

```
          {"name":"CalTool","version":"..."}}}}
```

The lists, `server/discover` and `resources/read` also carry the caching fields `ttlMs` and `cacheScope`. Lists may be kept for five minutes, while documents have a `ttlMs` of 0 because one of them is the live session state. `cacheScope` is always `private`.

6.6. Methods

Method	Use
`server/discover`	Revisions, capabilities, server name and version, and `instructions`: a text for the system prompt on how to address objects.
`initialize`	The handshake, handshake generation only. Its result also holds `instructions`.
`tools/list`	All tools, always all of them. Permissions are checked per call, not here.
`tools/call`	Runs one tool.
`resources/list`, `resources/read`, `resources/templates/list`	The documents. There are no templates.
`prompts/list`, `prompts/get`	The ready-made tasks.
`completion/complete`	Answered with an empty list.

The server offers tools, resources and prompts, and declares no change notifications, because the lists do not change while CalTool runs. It never sends a request or a notification of its own: no sampling, no elicitation, no progress messages. Every answer arrives in the response to the POST, always as `application/json`, never as an event stream. A `GET` on the endpoint therefore returns `405`.

6.7. Tool results

Every tool answers with one JSON object, sent twice in the same result: as text in `content[0].text` and as `structuredContent`. **Give your model only one of them.** Sending both costs twice the context for no gain. `render_plot` adds a second `content` item of type `image`, with a base64 PNG in `data`.

Nearly every result has a `summary`, a sentence that also appears in CalTool's activity list. A number that is not finite (NaN, infinity) travels as `null`. Text is not escaped beyond what JSON requires, so `°C` arrives as `°C`.

6.8. Errors

Two kinds of error exist, and they need different handling.

Tool errors are ordinary results with `isError` true, carrying one sentence as text. They cover refused permissions, unknown names, values outside limits and a declined confirmation. **Return them to the model**, which can read them and correct itself:

© 2026 Altech Electronics Nordic AB. All rights reserved.

```
{"content":[{"type":"text",
   "text":"Engine/RevLimit: 99999 is outside the A2L limits 3000 .. 9000."}],
 "isError":true}
```

Protocol errors are JSON-RPC errors, and mean your host did something wrong or the call was stopped. Handle them in your host:

HTTP	Code	Meaning
200	`-32700`	The body is not JSON.
200	`-32600`	The message is not a valid request. Status `400` on a current-revision batch.
200, 404	`-32601`	Unknown method (Section 6.4).
200, 400	`-32602`	Invalid parameters, such as an unknown tool name. Status `400` for incomplete `_meta`.
200	`-32603`	Internal error.
200	`-32800`	The request was cancelled (Section 6.9).
400	`-32020`	Headers and body disagree.
400	`-32022`	Unknown protocol revision.
200	`-32002`	Unknown resource, handshake generation.

Beside these, HTTP answers `401` for the token, `403` for the origin, `405` for `GET` and any method the server does not use, and `415` for a body that is not JSON.

6.9. Cancelling and timing

Send `notifications/cancelled` with the `requestId` of a running request, from another connection. It is answered with `202`. The tool stops at its next check, and the original request is answered with code `-32800`. Your host should ignore that answer. Merely closing the connection does not stop a running call.

- A **confirmation dialog stays open** when its request is cancelled, but if the engineer then confirms, nothing is written.
- Calls can take long. A confirmation lasts until the engineer answers, and `go_online` lasts as long as connecting does and can ask questions in CalTool. Set your timeout accordingly: the bridge and the reference host use ten minutes.
- An action or a flash is not stopped by cancelling or timing out the call that started it, since a client's timeout must not be what aborts a flash half-way. `waitSeconds` says how long the call stays to watch. Follow the run with `get_diagnostics` or `get_flash`, and stop it with `cancel_diagnostic_run` or `cancel_flash`.
- Requests on separate connections run concurrently, though anything touching the session takes its turn on CalTool's user interface thread. A dialog or a long operation in CalTool can delay other

© 2026 Altech Electronics Nordic AB. All rights reserved.

calls.

- An idle connection closes after 120 seconds, and a request body may be at most 16 MiB.

6.10. The stdio bridge

For hosts that can only start a program, CalTool installs `mcp\CalTool.McpBridge.exe` in its program folder. It reads one JSON-RPC message per line from standard input, posts it to the server, and writes the answer as one line to standard output. A notification writes nothing, and logging goes to standard error.

```
{ "mcpServers": { "caltool": {
    "command": "<CalTool's program folder>\\mcp\\CalTool.McpBridge.exe" } } }
```

Pick **Claude Desktop** under **Client** in the AI access dialog to see this entry with the exact path on your PC. With no arguments the bridge reads the port and token from `settings.json` when it starts, so a new token or port needs a restart of the host. `--url` and `--token` override them. If CalTool is not running or AI access is off, each request gets a JSON-RPC error with code `-32000` saying so, rather than silence.

The bridge handles one message at a time. A cancellation therefore waits for the running call to end, and is of no use through the bridge. Prefer HTTP if you can.

© 2026 Altech Electronics Nordic AB. All rights reserved.

7. The data model

7.1. Keys

Everything is addressed by a **key**, which is `Device/Name`. `Device` is the name of the ECU or CAN bus in the workspace, and `Name` is the object's name in the A2L or DBC file.

Form	Meaning
`Engine/RevLimit`	The object `RevLimit` of the device `Engine`.
`RevLimit`	Accepted when the name belongs to exactly one signal or object. When it belongs to several, the tool refuses and lists their keys.
`Engine/Temps[3]`, `Engine/Temps[1][2]`	An element of an array measurement.
`Calc/PowerEstimate`	A computed channel. `Calc` is a pseudo-device.
A log channel's key	As `list_signals` reports it. It carries the workspace device's name when the log's device matches one, and otherwise the log's own device name.
`pwt_rear/EngineSpeed`	A signal decoded from a log's CAN frames. The device part is the file name of the DBC that decoded it. Where one DBC decodes several buses, the bus comes first: `CAN2·pwt_rear/EngineSpeed`. Where two messages of the DBC declare the same signal name, the name is `Message.Signal`.

Every tool reads a name the same way. A bare name can belong to several signals at once, for example an A2L measurement `Engine/EngineSpeed` and a decoded CAN signal `pwt_rear/EngineSpeed`. The tool then refuses it and lists both keys, and does not choose one for the model. A key that nothing in the session has is refused too: `open_plot` and `add_to_plot` do not chart it as an empty row.

Prefer the full key once you have it. A workspace can gain a second device, and a log can be decoded with a DBC that declares the same name.

7.2. Kinds of object

measurement

A signal the ECU can stream, or that a CAN bus carries. It is read-only, except that a `READ_WRITE` measurement can be set like a calibration value.

characteristic

A calibration object: a value, a curve, a map, an array, an enum or a text.

axis

A shared breakpoint table (`AXIS_PTS`) that curves and maps refer to. Changing it changes every table that uses it.

© 2026 Altech Electronics Nordic AB. All rights reserved.

log

A channel of the loaded log. `recording` is a channel of the current recording.

computed

A channel defined by a JavaScript function over other signals.

7.3. Calibration values

All values are physical: what the engineer sees, with the A2L's conversion already applied.

Shape	Read as `value`	Write with
Scalar	A number	`"value": 6400`
Enum	A number, with `text`	`"value": 2` or `"value": "Sport"`
Curve, array, axis	An array	`"values": [1.0, 1.2, 1.5]`, one per element in order
Map	An array of rows	`"rows": [[…], […]]`, with Y as the outer array and X as the inner

A curve or map is always written whole, so read it first and change only the cells you mean to. The breakpoints are in `axes[].breakpoints`, from `describe_signal` or `get_calibration`. Objects with more than two axes cannot be written. `breakpointsAreEstimated` true means CalTool has no real breakpoints for the axis yet and has generated some. Do not reason from those.

`value` is what the desktop holds now. Each value has a reference: a loaded memory image, or what the ECU held when it was connected. `changed` says whether the value differs from that reference. Where it does, `referenceValue` gives the reference and `referenceSource` says where it came from. The comparison is made in the object's own datatype, so a typed 0.1 and the 0.1 a `FLOAT32` holds are the same value.

`origin` says where the desktop value came from: `ecu` means it was read from the ECU and nobody has edited it since, and `desktop` means it was edited or loaded from a file. It does not say what the ECU holds now. For that, `read_ecu_calibration` reads the objects from the online ECU and says whether each matches the desktop.

To find what has been changed, call `get_calibration` with `changedOnly`. It lists every value that differs from its reference or was edited in this run of CalTool. An edit made in this run also carries `changedBy` (`ai` or `user`) and `changedAt`. A table with more than 32 elements in a listing, or 400 when asked for by key, comes as its shape, range and the cells that changed. Pass `full` to get it whole.

`get_calibration` answers `value: null` until a value exists: the device has to go online, or have a calibration file or memory image loaded.

When `set_calibration` writes to an online ECU, CalTool reads every written value back before it answers. `onEcu` true means the ECU holds the new value. `onEcu` false means the ECU did not keep it, and CalTool puts the desktop value back to what the ECU holds. `ecuHolds` then says what that is.

© 2026 Altech Electronics Nordic AB. All rights reserved.

7.4. Time and data

Times are seconds on the log's timeline, and a recording's first sample is at 0. Each channel has its own samples, at its own rate. `get_signal_data` with mode `at` returns the value in force at a given time, and `nearestSampleTime` says how close the actual sample was.

Signal data comes from one of two places: the loaded log or the recording. By default a tool reads what CalTool shows. That is the recording while it captures and after it stops, until a log is opened, and the log from then on. Starting a recording unloads the log, so the two exist together only when a log was opened after a recording. Every data tool takes `source` (`auto`, `log` or `recording`) to choose, and its answer says what it read in `dataSource`. When the other source also holds data, `otherSource` names it, so a model that read the wrong one can tell. `get_state` reports the default as `dataSource`.

A new recording replaces the previous one in memory. `get_state` lists the files saved in this run under `recording.savedRecordings`. The data tools take `file` to read any log file, such as an earlier recording, without changing what CalTool shows. Their answer then says `dataSource: "file"`, and the times are the file's own.

The data tools give each signal's `samplePeriod`: the mean time in seconds between its samples. A time read from a signal is accurate to one period, so an event time from a 100 ms raster can be up to 100 ms late. `get_signal_grid` warns when its step is finer than a signal's period.

To do heavy work on the file itself, with a tool such as asammdf or MATLAB, `get_state` gives the log's `path` and the file's own first and last master time (`fileFirstTime`, `fileLastTime`). Where CalTool shows the log shifted or cropped, `timeWindow` gives the `offset` and the file times kept. A sample recorded at file time f appears in CalTool at f + `offset`.

Live values stream only while a recording captures, as on screen. For that, a measurement must have a raster (`set_raster`), its device must be online, and `start_recording` must have run. CalTool refreshes the streamed values about every 50 ms. A measurement named in `keys` that does not stream, because no recording runs or it has no raster, is read once from its online ECU for that call. `polled` lists those keys. A key with no value at all is listed under `notRead` with the reason, such as the device being offline. Live values are meant for looking, not for control, since a value carries no time stamp and each call also has the delay of your host and model.

A raster change made with `set_raster` while a recording runs takes effect at once: the ECU's DAQ is reconfigured and the measurement is added to the recording. The answer's `takesEffect` says which applies.

A log can also hold raw CAN frames, for example from a bus logger. `get_state` lists its buses under `log.canBuses`. `configure_log_can_decoding` ties a DBC to each bus. Each bus is decoded only with its own DBC, so two buses that use one identifier for different messages stay separate. The decoded signals are log channels with keys as described in Section 7.1. The decoding is saved with the workspace, for that log. `get_can_frames` reads the frames themselves and can check an alive counter and a CRC over every frame.

© 2026 Altech Electronics Nordic AB. All rights reserved.

7.5. What the engineer sees

Pages, cards, plots and markers are the shared space between your model and the engineer. A model that only answers in text leaves the engineer to check its claims by hand. A model that places a marker at the event and opens a plot of the signals lets them verify it at a glance. Card keys such as `P:#3` come from `get_state` and from the tool that opened the card.

© 2026 Altech Electronics Nordic AB. All rights reserved.

8. Permissions and safety

An LLM can be wrong, and text in the data it reads can mislead it. CalTool's rules are therefore enforced in CalTool, not left to the model or the host. They apply to every call, however your host is built.

8.1. Access tiers

Every tool belongs to one tier. `tools/list` states it in each tool's `_meta`, under `io.altechelectronics/caltool-tier`.

Tier	Tools	Permission in the dialog	Scope
`read`	18	None: allowed while on	The session, signals, values, markers.
`analyse`	5	None: allowed while on	Computes over data and stores nothing.
`workspace`	25	**Change the workspace**, on by default	Computed channels, markers, plots, cards, pages, cursors, rasters, logs, recordings, saving files.
`calibration`	3	**Change calibration values on the desktop and in files**, off by default	Values in the desktop store, offline, and calibration files.
`live-ecu`	2	**Write to an online ECU**, off by default	Going online and offline, and any calibration write that reaches a running ECU.
`can-trace`	15	**Use the CAN trace**, off by default	Interface cards, listening, DBC files, log files, and setting up transmit rows, scripts and replays. Sends nothing.
`can-transmit`	5	**Send on a CAN bus**, off by default	Single frames, cyclic messages, running scripts, replaying logs.
`diagnostics`	11	**Run diagnostic sessions**, off by default	UDS and KWP2000: fault codes, data identifiers, memory, the ECU file's actions, catalogue services, unlocking, and requests of the model's own.
`flash`	3	**Reprogram ECUs**, off by default	Replacing an ECU's firmware over XCP or UDS, or through Vector's VX flash tool.

Tools that only report the state of the CAN trace, diagnostics and flashing are in the `read` tier.

AI access itself is off until the engineer switches it on. The permissions are per PC, not per workspace or client. They are checked at each call, so a change in the dialog applies from the next call, also while a host is connected. A refusal is a tool error naming the box to tick.

© 2026 Altech Electronics Nordic AB. All rights reserved.

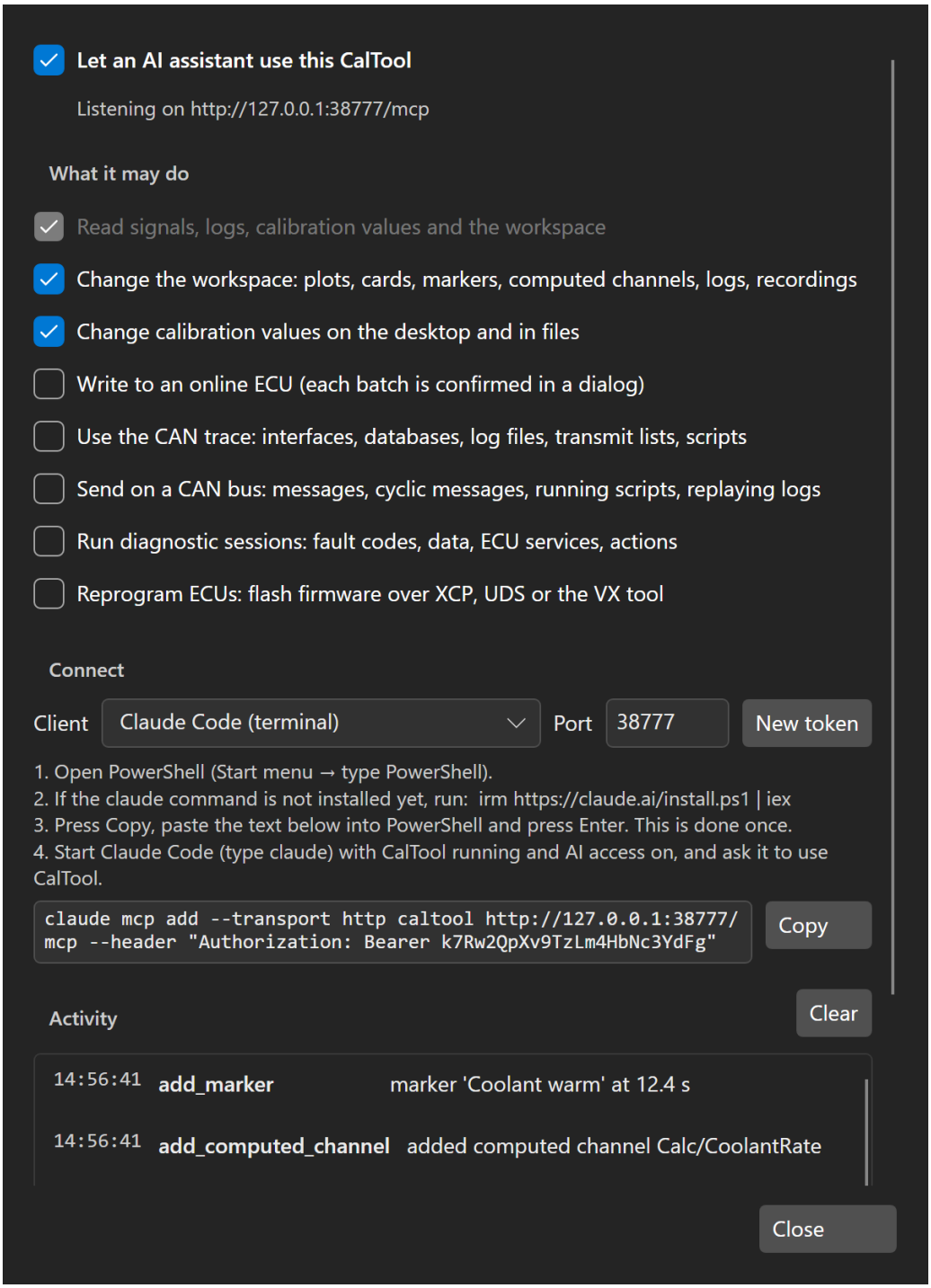

Figure 3. The AI access dialog: the switch, the permissions, the connection and the activity list

8.2. What happens when a model writes calibration

`set_calibration` goes through these steps. Nothing is written until the last one.

1. **Permission.** The `calibration` tier must be allowed.
2. **Validation of the whole batch**, up to 200 changes. Each object must exist and be writable, and its value must have the right shape and lie within the A2L limits. One bad entry refuses the whole batch. With `partial`, keys that do not exist are skipped and listed under `failed` instead. A wrong shape or a value outside the limits still refuses the batch.

© 2026 Altech Electronics Nordic AB. All rights reserved.

3. **Reach.** CalTool works out which ECUs the batch would reach: the devices it names, and any partner in their sync group. Devices in a sync group mirror each other's changes, so a write to an offline device reaches an online partner. Partners' own limits are checked as well.
4. **Live permission and confirmation.** If any ECU would be reached, **Write to an online ECU** must be allowed, and the engineer must confirm the batch in a dialog showing old and new values, partners included, with the reason your host gave in `note`. The dialog waits for an answer, and closing it declines.
5. **Last check.** The batch is abandoned if the request was cancelled while the dialog was open, or if an ECU has come online in the meantime that the engineer did not confirm.
6. **Write.** Offline, values land in the desktop store with change marks on the cards. Online, they are written to the ECU and read back. The answer comes after the read-back: `onEcu` says whether the ECU holds each value, and a value the ECU did not keep is put back in the desktop store as the ECU holds it. A value the desktop already holds is not written again unless the call sets `force`.

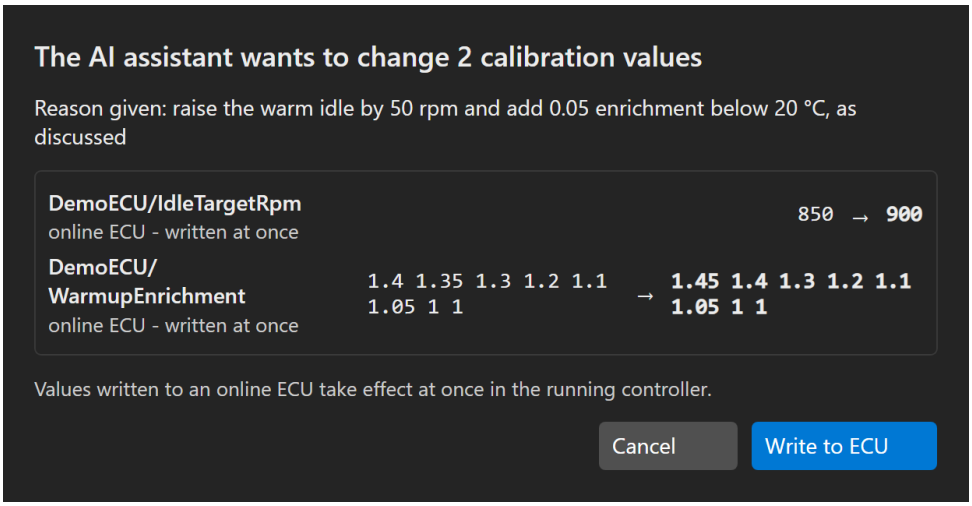

Figure 4. The confirmation of a write to an online ECU

The calibration tools cannot get past this dialog. No argument or permission skips it, and it is not exposed over MCP. The permissions in Section 8.3 reach an ECU by other routes, without it. `load_calibration_file` is refused while the device or a sync partner is online, since a load writes every object.

A value changed offline can still reach an ECU later. When a device connects, CalTool's ordinary comparison of desktop and ECU applies, with the engineer deciding. A sync partner that was offline when the change was mirrored receives it when it next comes online, as with any change made by hand.

Undo. One `set_calibration` batch is one step in CalTool's history. The engineer can undo it with `Ctrl` + `Z`, which online writes the old values back to the ECU. There is no undo tool for the model, by design.



The A2L limits are the only numeric check. A value inside them can still damage an engine or a test bench, and CalTool has no limit on how far one step may move a

© 2026 Altech Electronics Nordic AB. All rights reserved.

value. The confirmation dialog is your protection, and it is only as good as the attention of the person answering it. Allow **Write to an online ECU** only for a session where someone is at the bench and watching.

8.3. The CAN bus, diagnostics and flashing

For these four tiers the tick in the dialog is the whole of the permission. A tool runs as soon as the model calls it, and CalTool does not ask the engineer first. Everything is listed in the **Activity** list, but only after it has happened.

Permission	What a model can do with it
Use the CAN trace	Open any interface at any bitrate. The interface takes part in the bus as an ordinary node, so it acknowledges frames, and at the wrong bitrate it disturbs the bus with error frames. It sends no messages of its own.
Send on a CAN bus	Send any identifier with any payload on any connected interface. A cyclic message, a script or a replay goes on sending after the model has stopped calling, until it is stopped or the interface is disconnected.
Run diagnostic sessions	Request any service the ECU accepts, including writes, routines, resets and unlocking, with `send_diagnostic_request`, and run the ECU file's actions, which can drive actuators. An XCP session on the same channel goes offline for as long as the session lasts.
Reprogram ECUs	Replace the firmware. An image that does not match the device's A2L is warned about and flashed anyway, and a flash that is interrupted can leave the ECU unable to start. Vector's VX flash tool is the one route that still needs the engineer, who picks the device and presses Flash in its window.



Send on a CAN bus, **Run diagnostic sessions** and **Reprogram ECUs** can each change a running ECU without the confirmation that protects calibration writes. A model can put XCP commands on the bus, write through UDS, or replace the calibration with the firmware. If your safety case rests on the engineer confirming each change, leave these three unticked. Allow them only on a bench or a vehicle that is safe to disturb, with someone present who can stop it.

An action in the ECU file can hold a `confirm` step, for example "open the bleed screw". Such a step is put to the person at the ECU in CalTool's window, and the action waits for the answer, because it asks for something a model cannot do.

The model stops a run with `control_can_replay`, `stop_can_script`, `set_can_cyclic`, `cancel_diagnostic_run` and `cancel_flash`. These need the same permission as starting one. The engineer can stop any of them in CalTool at any time, and unticking a permission does not stop what is already running.

© 2026 Altech Electronics Nordic AB. All rights reserved.

8.4. The token

The token is 24 characters from a set of 55, generated at random when AI access is first switched on. It is the whole of the access control on the port, and **anything on the PC holding it can do what the permissions allow**.

- It is stored as plain text in `%APPDATA%\CalTool\settings.json`, as are the permissions. Anyone who can read the Windows user's files can read it. Treat the Windows account as the security boundary.
- **New token** in the dialog replaces it at once. Every client then needs the new one.
- Do not put it in source control or in a prompt. Your host needs it, your model does not.
- Requests on loopback are not encrypted, which is normal for traffic that stays inside the PC.

8.5. Prompt injection

Your model reads text from sources you may not control: A2L descriptions, signal names, log comments, marker comments, file names. Text in that data can be written to look like an instruction, and an LLM cannot reliably tell the difference. This is an unsolved problem across the industry, and applies to every AI integration, not only this one.

What limits the damage here:

- Permissions and the confirmation dialog do not depend on what the model believes. The tiers in Section 8.3 have only the permission, so injected text matters most there.
- Allowing a tier only when the task needs it keeps the possible harm small. An analysis session needs no write permission at all.
- Your host may offer only the tiers the task needs (Section 11.2).

File access deserves your attention. `open_log`, `save_recording`, `save_workspace`, `open_workspace`, `load_calibration_file`, `save_calibration_file`, `start_can_log`, `open_can_trace_file`, `configure_can_interface`, `configure_can_replay`, `diagnostics_connect`, `flash_ecu` and `flash_ecu_uds` take a path and act with the Windows user's rights, with no restriction on the folder. A save overwrites an existing file. An ECU file can name a seed and key DLL, which CalTool loads and runs when the session unlocks, so a model that can point `diagnostics_connect` at a file can run the code that file names. If your model also has tools that send data out of the company, such as web access, mail or a remote API, then a misled model could carry calibration data out through them. **Do not combine CalTool's tools with tools that reach outside in the same session** unless you have assessed that.

8.6. What is recorded

The **Activity** list in the dialog shows each call, reads included: time, tool, and a one-sentence outcome or the reason it was refused. It holds the latest 300 entries, lives only in memory, and does not record arguments. A batch's `note` appears in its outcome.

© 2026 Altech Electronics Nordic AB. All rights reserved.

If you need a lasting record, for example for traceability of a calibration, log it in your host, which sees each call in full with its arguments and answer. The status bar's **AI** chip lights on each request, so the engineer can always see when a client is active.

8.7. Recommended practice

- Start with a host offering `read` and `analyse` only, on a simulated ECU or a copy of a log.
- Give the engineer a way to see what the model is doing: keep the dialog's **Activity** list open at first.
- Require a `note` on every `set_calibration` in your system prompt, since the engineer reads it when deciding.
- Allow writes offline before allowing them online, and save a calibration file before a session that changes values.
- Allow **Use the CAN trace** before **Send on a CAN bus**, and try scripts and transmit rows on the **Simulation** channel.
- Tick **Reprogram ECUs** only for the session that needs it, and keep the previous image at hand.
- Switch AI access off when the session ends.

© 2026 Altech Electronics Nordic AB. All rights reserved.

9. Tool reference

CalTool offers 87 tools. This chapter is generated from what the server itself reports. The description of each tool below is the same text your model receives, and it is written for the model, so it also shows you what your model is told. In `tools/list`, each tool has these fields:

```
{"name": "add_marker",
 "description": "Puts a marker on the timeline, shown as a flag ...",
 "inputSchema": {"type": "object",
                 "properties": {"time": {"type": "number",
                                         "description": "..."}},
                 "additionalProperties": false},
 "annotations": {"readOnlyHint": false, "destructiveHint": false,
                 "openWorldHint": false, "title": "add marker"},
 "_meta": {"io.altechelectronics/caltool-tier": "workspace"}}
```

`inputSchema` is JSON Schema, and no tool accepts arguments other than those listed. In the tables, `name[].field` is a field of each entry in the array `name`. Hints describe a tool for a host's benefit and are not guarantees: *additive* means it adds or changes without removing, *destructive* that it can remove or overwrite something. The permission rules in Chapter 8 apply regardless.

The **Returns** entries list the fields of the JSON object described in Section 6.7.

9.1. Read tools

Always allowed while AI access is on. Changes nothing.

get_state

The session as it stands, as a summary: devices (name, kind, online), the loaded log or recording, cursors, page names with card counts, access permissions. Call this first. 'sections' adds detail only where asked: 'pages' (every card with its signals), 'plots' (plot pane tabs' signals), 'computed' (computed channels and inputs), 'rasters' (each device's rasters), or 'all'.

Argument	Type	Req.	Meaning
`sections`	array of enum		Detail to include; omit for the summary. Each one of: pages, plots, computed, rasters, all.

Hint: read-only. Returns: A summary: `calTool` (version); `workspace` {`path`, `dirty`, `pages`[{`name`, `selected`, `cards` (count)}]}; `devices`[{`name`, `kind`, `file`, `loaded`, `online`, `connecting`, `active`, `status`, `measurements`, `characteristics`, `axes`, `rasters` (count), `readOnly`, `calibrationFile`}]; `dataSource` (what the data tools read by default: `log`, `recording` or `none`); `log` {`path`, `signals`, `start`, `hasCanTraffic`, `canBuses`[{`bus`, `label`, `frames`, `dbc`, `decodedSignals`}] for a log holding CAN frames, `fileFirstTime` and `fileLastTime` (the file's own master times), and either `timeline` (CalTool's times are the file's) or `timeWindow` {`offset`, `fromFileTime`, `toFileTime`, `note`} where CalTool shifts or crops the log} or null; `recording` {`isRecording`,

© 2026 Altech Electronics Nordic AB. All rights reserved.

hasData, hasSession, channels, elapsedSeconds, lastTime, savedTo (where this recording was saved, or null), savedRecordings (the last ten files saved in this run, for the data tools' file argument)}; cursor { a, b, timeWindow}; plotPaneTabs[{name, selected, signals (count)}]; computedChannels (count); markers (count); liveValuesAvailable (count); access {workspaceChanges, calibrationChanges, liveEcuChanges, canTrace, canTransmit, diagnostics, flash}; canTrace {interfaces, connected, frames}; diagnostics {connected, ecuFile, session}; flash {running}; statusBar; detail (how to ask for more); summary. A section named in sections replaces its count with the list: pages gives cards[{key, title, kind, x, y, width, height, signals or objects}]; plots gives each tab's signals as keys; computed gives computedChannels[{key, unit, inputs}]; rasters gives each device's rasters[{name, periodMs}].

list_signals

Searches the loaded A2Ls, DBCs and log for measurements, characteristics (calibration objects), axes, log channels and computed channels. 'query' is a plain substring, a *wildcard or a regex. Returns keys to use with the other tools. kinds ['log'] lists only what the log (or recording) holds; namesOnly keeps the answer short.

Argument	Type	Req.	Meaning
query	string		Name filter: substring, wildcard (*) or regex. Empty lists everything.
kinds	array of enum		Which kinds to list; default all. Each one of: measurement, characteristic, axis, log, computed.
device	string		Only this device's objects.
limit	integer		Maximum entries (default 100, at most 1000).
offset	integer		Skip this many matches, for paging.
namesOnly	boolean		Return only the keys, as one array (default false; limit defaults to 1000 then).
source	string		For kind 'log': Where to read samples: 'log' (the loaded log file), 'recording' (the current or last recording), or 'auto' (default: what the app shows - the recording while it captures or after it stopped, the log once one is opened). One of: auto, log, recording.
file	string		List the channels of this log file instead (full path); nothing on screen changes.

Hint: read-only. Returns: total; offset; returned; signals[{key, kind, unit, and by kind: info, elements, readWrite, raster for a measurement; shape for a characteristic; points for an axis; samples and source for a log channel, and can {bus, busLabel, id, message} for a signal decoded from the log's CAN frames}]; summary; dataSource and otherSource as in get_signal_data when log channels were listed. With namesOnly, keys (an array of keys) replaces signals. The kind of a channel in the current recording is recording. With file: dataSource is file, file, and the file's channels with kind file.

© 2026 Altech Electronics Nordic AB. All rights reserved.

describe_signal

Everything the A2L/DBC says about one object: datatype, conversion, unit, limits, display format, enum states, address, shape and axes of a table, raster assignment, current value.

Argument	Type	Req.	Meaning
key	string	yes	'Device/Name', or a bare name declared by one device.

Hint: read-only. Returns: Always key, kind. Measurement: name, device, description, datatype, conversion, unit, lowerLimit, upperLimit, format, address, readWrite, dimensions, enum[{raw, text}], raster, rasters, liveValue, and for a writable one isCalibration, value. Characteristic: the same descriptive fields plus type, shape, rows, columns, axes[{axis, attribute, inputQuantity, unit, points, axisPtsRef, breakpoints, breakpointsAreEstimated}], readOnly, value, referenceValue, origin (ecu or desktop, as in get_calibration). Axis: inputQuantity, points, conversion, unit, limits, address, note, value. Log channel: name, unit, samples, source, enum, wholeNumbers, and for a signal decoded from CAN frames can {bus, busLabel, id, message, extended, dbc, messageLength, transmitter, cycleTimeMs, startBit, length, byteOrder, signed, factor, offset, minimum, maximum, unit, multiplexing, multiplexorValue, multiplexor, valueTable[{raw, text}], valueTableNote, comment}. Computed channel: as in list_computed_channels.

list_computed_channels

Every computed channel with its inputs, trigger and full script.

No arguments.

Hint: read-only. Returns: channels[{name, key, unit, inputs[{key, alias}], trigger, script}]; summary.

list_markers

The markers on the timeline (log events, connection losses, the user's and the AI's own notes).

No arguments.

Hint: read-only. Returns: markers[{index, name, time, endTime, comment, kind, cause}]; summary.

get_signal_data

Samples of signals from the loaded log or the recording (see 'source'; the answer's dataSource says which was read). mode 'statistics' (min/max/mean/count over the range), 'decimated' (at most maxPoints min/max-preserving points - use this for plotting or an overview), 'samples' (raw samples, capped), or 'at' (the value at one time). Times are seconds on the log's timeline.

Argument	Type	Req.	Meaning
keys	array of string	yes	Signal keys ('Device/Name', 'Calc/Name', or a log channel key).

© 2026 Altech Electronics Nordic AB. All rights reserved.

Argument	Type	Req.	Meaning
mode	string		What to return (default 'statistics'). One of: statistics, decimated, samples, at.
from	number		Start time in seconds (default: the signal's first sample).
to	number		End time in seconds (default: the signal's last sample).
time	number		For mode 'at': the time to read.
maxPoints	integer		For 'decimated' (default 500) and 'samples' (default 2000, at most 20000).
source	string		Where to read samples: 'log' (the loaded log file), 'recording' (the current or last recording), or 'auto' (default: what the app shows - the recording while it captures or after it stopped, the log once one is opened). One of: auto, log, recording.
file	string		Read this log file (full path, e.g. an earlier recording from get_state's savedRecordings) instead of the session's log or recording. Nothing on screen changes. Times are the file's own; a key matches by channel key, else by a name only one channel of the file has.

Hint: read-only. Returns: dataSource (log or recording: what was read) and, when the other source also holds data, otherSource (a sentence naming it); signals[{key, unit, source (recording or log), sampleCount, firstTime, lastTime, samplePeriod, and by mode}]; summary. Mode statistics: from, to, count, min, max, mean, first, last, and note for an enum. Mode decimated: from, to, points as [time, value] pairs, pointCount, format. Mode samples: from, to, samples as [time, value] pairs, format; a key whose range holds more samples than maxPoints goes to failed. Mode at: time, value, text for an enum, nearestSampleTime. At most 50 keys per call. Keys that could not be answered are listed in failed[{key, error}] and the others are still answered; the call fails only when no key can be answered. samplePeriod is the mean time in seconds between the signal's samples: a time taken from it is accurate to one period. With file: dataSource is file, plus file, fileStart and timeline (the times are the file's own).

get_live_values

The latest values of measurements. While a recording captures these are the live readouts. A key with no live value (no recording, or not on a raster) is read once from its ONLINE ECU for this call - e.g. a learned value between runs; 'polled' lists those. Keys omitted = everything live.

Argument	Type	Req.	Meaning
keys	array of string		Measurement keys; omit for all live values (at most 100 are read from the ECU).

Hint: read-only. Returns: values {key: number or null}; recording; polled (the keys read once from the online ECU for this call, not streamed) and polledNote; texts {key: state name} for an enum; notRead [{key, reason}] for keys with no value (not a measurement of a loaded device, device offline, a CAN

© 2026 Altech Electronics Nordic AB. All rights reserved.

bus signal outside a recording, over the per-call read limit, or a failed read); `note` when nothing streams and no keys were named; `summary`.

get_calibration

Calibration values from the desktop store: scalars, curves (values + breakpoints), maps (rows = Y, columns = X), arrays and axes. 'changed' says whether a value differs from its reference (memory image, or the ECU at connect); the reference itself is given only where it differs. changedOnly: true lists exactly what was changed - with changedBy ('ai' or 'user') and changedAt for edits made in this run of CalTool. A large table comes as shape/min/max plus its changed cells unless 'full'. 'origin' is where the desktop value came from ('ecu' = read from the ECU and untouched, 'desktop' = edited or loaded from a file); it does not say what the ECU holds now - read_ecu_calibration does. Unknown keys are listed under 'failed'.

Argument	Type	Req.	Meaning
`keys`	array of string		Characteristic/axis keys. Omit to list a device's values (with 'device' and 'query').
`device`	string		With no keys: the device whose values to list.
`query`	string		With no keys: a name filter (substring, * or regex).
`changedOnly`	boolean		Only values that differ from their reference or were edited in this run (default false).
`full`	boolean		Return large tables whole (default: summarised above 32 elements in a listing, 400 by key).
`includeStale`	boolean		With no keys: also list store entries the loaded A2L no longer declares (default false).
`limit`	integer		With no keys: at most this many (default 50, 200 with changedOnly; max 500).
`offset`	integer		With no keys: skip this many matches, for paging.

Hint: read-only. Returns: With `keys`: `values`[{`key`, `value` or, for a table over 400 elements unless `full`, `valueSummary` {`shape`, `min`, `max`, `mean`}, `origin` (`ecu` = read from the ECU and untouched since, `desktop` = edited or loaded from a file), `changed` (differs from its reference), and where it differs `referenceSource` with `referenceValue` (or, for a large table, `changedCells` {`count`, `format` "[row, column, reference, value]", `cells`, `truncated`} or `referenceSummary`), `changedBy` (`ai` or `user`) and `changedAt` for an edit made in this run of CalTool, `unit`, `shape`, `axes`, `text` for an enum}], and `value` null with a `note` when the desktop holds nothing yet. With `changedOnly`, unchanged keys are left out. Unknown keys go to `failed`[{`key`, `error`}]; the call fails only when no key can be answered. Without `keys`: `total`, `offset`, `values` (the same fields; tables over 32 elements summarised unless `full`; `notInA2l` on a stale entry listed with `includeStale`), `staleSkipped` when entries the loaded A2L no longer declares were left out. Always `summary`. A value is a number (scalar), an array (curve, array, axis) or an array of rows (map). `origin` says where the desktop value came from, not what the ECU holds now - read_ecu_calibration does.

© 2026 Altech Electronics Nordic AB. All rights reserved.

get_snapshot

The values of several signals at one time and at offsets before and after it - what an edge or fault investigation needs after find_events located the instant. Each value is the sample in force at that time (sample-and-hold); null where the signal has no sample yet.

Argument	Type	Req.	Meaning
`keys`	array of string	yes	Signal keys (at most 50).
`time`	number	yes	The instant, in seconds on the log's timeline.
`offsets`	array of number		Offsets from 'time' in seconds (default -1, -0.5, -0.1, 0, 0.1, 0.5, 1; at most 50).
`source`	string		Where to read samples: 'log' (the loaded log file), 'recording' (the current or last recording), or 'auto' (default: what the app shows - the recording while it captures or after it stopped, the log once one is opened). One of: auto, log, recording.
`file`	string		Read this log file (full path, e.g. an earlier recording from get_state's savedRecordings) instead of the session's log or recording. Nothing on screen changes. Times are the file's own; a key matches by channel key, else by a name only one channel of the file has.

Hint: read-only. Returns: `dataSource` (`log` or `recording`: what was read) and, when the other source also holds data, `otherSource` (a sentence naming it); `time`; `offsets` (sorted); `times` (`time` plus each offset); `signals`[{`key`, `unit`, `samplePeriod`, `values`, and `texts` for an enum}], where `values[i]` is the value in force at `times[i]`, or null before the signal's first sample; `format`; `summary`. At most 50 keys and 50 offsets. Keys that could not be answered are listed in `failed`[{`key`, `error`}] and the others are still answered; the call fails only when no key can be answered. `samplePeriod` is the mean time in seconds between the signal's samples: a time taken from it is accurate to one period. With `file`: `dataSource` is `file`, plus `file`, `fileStart` and `timeline` (the times are the file's own).

get_signal_grid

Several signals sampled on ONE common time grid (sample-and-hold), as a compact table with enum values as state names - the call to use for comparing signals over a window. Give 'step' in seconds, or leave it out to spread maxRows rows over from..to.

Argument	Type	Req.	Meaning
`keys`	array of string	yes	Signal keys (at most 50).
`from`	number		Start time in seconds (default: the earliest first sample).
`to`	number		End time in seconds (default: the latest last sample).
`step`	number		Grid spacing in seconds (default: (to - from) / (maxRows - 1)).

© 2026 Altech Electronics Nordic AB. All rights reserved.

Argument	Type	Req.	Meaning
maxRows	integer		At most this many rows (default 200; rows × signals at most 50000).
decodeEnums	boolean		Write enum values as their state names (default true).
source	string		Where to read samples: 'log' (the loaded log file), 'recording' (the current or last recording), or 'auto' (default: what the app shows - the recording while it captures or after it stopped, the log once one is opened). One of: auto, log, recording.
file	string		Read this log file (full path, e.g. an earlier recording from get_state's savedRecordings) instead of the session's log or recording. Nothing on screen changes. Times are the file's own; a key matches by channel key, else by a name only one channel of the file has.

Hint: read-only. Returns: dataSource (log or recording: what was read) and, when the other source also holds data, otherSource (a sentence naming it); from; to (the last row's time); step; columns (t, then each key); units; rows, each [t, a value per column] - the value in force at t, null before a signal's first sample, an enum as its state name with the raw value in brackets unless decodeEnums is false; samplePeriods {key: seconds}; warning when the step is finer than a signal's sample period (that column repeats held values, and a change in it shows up to one period late); format; summary. At most 50 keys and 50 000 cells; a step that would need more rows is refused, saying how many. Keys that could not be answered are listed in failed[{key, error}] and the others are still answered; the call fails only when no key can be answered. With file: dataSource is file, plus file, fileStart and timeline (the times are the file's own).

get_plot_view

What a plot shows right now: its time window and the time span of its data, and per signal the Y-axis range, colour, whether it is drawn, and min/max/mean of its samples inside the window. 'plot' is a plot card key (P:#n), a card title or a plot pane tab name; omit it to list every plot. Use it before set_plot_view to see where you are, and after to see what the zoom shows.

Argument	Type	Req.	Meaning
plot	string		Which plot; omit for all of them.

Hint: read-only. Returns: plots, each {plot (the card key, or the pane tab's name), title, page, onScreen, maximized, timeWindow {min, max} (null before a window was established), dataTimeSpan {first, last}, xAxis (time or a signal), subplots, each {subplot, signals}}. A signal is {key, unit, color, visible, selected, yMin, yMax, axisPinned, dotsOnly, samples, firstTime, lastTime, inWindow {samples, min, max, mean of its samples inside the time window}}; a signal charted by name with nothing behind it yet has missing instead of the sample figures. Then cursorA; cursorB; syncTimeAxes; cursorStatistics; recording; summary.

© 2026 Altech Electronics Nordic AB. All rights reserved.

read_ecu_calibration

Uploads named calibration objects from the ONLINE ECU right now and returns what it holds, beside the desktop value and whether the two match (compared in the object's own datatype, so 0.1 against a FLOAT32 0.1 matches). Nothing is changed on either side. Use it to confirm that a value is on the ECU.

Argument	Type	Req.	Meaning
`keys`	array of string	yes	Characteristic, axis or READ_WRITE measurement keys (at most 100).
`full`	boolean		Return large tables whole instead of as a summary (default false).

Hint: read-only. Returns: `values`[{`key`, `ecuValue` (or `ecuSummary` {`shape`, `min`, `max`, `mean`} for a large table unless `full`), `matchesDesktop` (compared in the object's own datatype), and where they differ `desktopValue` or, for a large table, `differingCells` {`count`, `format` "[row, column, ecu, desktop]", `cells`, `truncated`}; `desktopValue` null when the desktop holds nothing}]; `failed`[{`key`, `error`}] for keys that do not resolve, whose device is not online, or that the ECU did not answer; `summary`. The call fails only when nothing could be read. Nothing is changed on either side.

get_can_trace

The CAN trace pane as it stands: interface cards (channel, bitrate, CAN FD, DBC, connected, bus health), the channels this PC offers, the transmit list, scripts with their output, the replay, the file log.

No arguments.

Hint: read-only. Returns: `interfaces`[as `interface` in configure_can_interface]; `availableChannels`; `totalFrames`; `identifiers`; `hiddenIdentifiers`; `fixedIds`; `decimal`; `fileLog` {`path`, `frames`} or null; `transmit`[as `row` in configure_can_transmit]; `scripts`[as `script` in run_can_script]; `replay` (as in configure_can_replay); `summary`.

get_can_frames

Reads CAN frames. By default what the CAN trace pane captured; source 'log' reads the loaded log's own frames off its file, and 'file' any .mf4/.blf/.txt - neither touches the trace on screen. mode 'latest' gives the newest frame of each identifier with its count and cycle time; 'sequence' frames in time order, paged with 'offset'. 'from'/'to' pick a time window (a log's own timeline). Every frame carries its bus 'channel'. 'check' verifies an alive counter and/or a CRC over every matching frame in the window and returns pass rates and the first failures instead of frames.

Argument	Type	Req.	Meaning
`source`	string		trace (default): what the CAN trace pane captured. log: the raw frames of the loaded log, read off its file. One of: trace, log.

© 2026 Altech Electronics Nordic AB. All rights reserved.

Argument	Type	Req.	Meaning
file	string		Read the frames of this .mf4, .blf or .txt (full path) instead; nothing on screen changes.
mode	string		latest: the newest frame of each identifier with its count and cycle time (default for the trace). sequence: frames in time order (default for a log or file). One of: latest, sequence.
from	number		Only frames at or after this time (s). For a log, its timeline - the one the cursors and markers use.
to	number		Only frames at or before this time (s).
interface	string		Trace only: only this interface card.
channels	array of integer		Only these bus numbers (each frame's 'channel').
ids	array of string		Only these identifiers. Decimal, or hex with 0x.
name	string		Only messages whose DBC name matches (substring, * wildcard or regex).
decode	boolean		Include DBC-decoded signals; default true for latest, false for sequence.
limit	integer		Maximum frames (default 100, at most 1000).
offset	integer		Sequence: skip this many matching frames - from the start of the window for a log or file, from the newest for the trace. The answer's 'nextOffset' continues.
check	object		

Hint: read-only. Returns: `frames`[{`time`, `channel` (the bus), `interface`, `id`, `extended`, `name`, `direction`, `kind`, `length`, `data`, and in mode `latest` `count` and `cycleMs`, with `decode` `signals`[{`name`, `value`, `unit`, `state`}]}]; `summary`. `id` is always hex, `ERR` for error frames. From the trace: `matching`, `nextOffset` when more follow, and a `note` (the trace keeps only the newest frames). From `source` `log` or a `file`: `file`, `decodedWith` (the databases that named the frames), `timeline` where the log's time window applies, `framesScanned`, `nextOffset` when more follow, and `identifiers` in mode `latest`. With `check`, the frames are replaced by `matchingFrames`, `counter` {`checked`, `ok`, `repeats`, `jumps`, `passRatePercent`, `firstFailures`[{`time`, `channel`, `id`, `previous`, `expected`, `got`, `kind`}], `note`} and/or `crc` {`checked`, `ok`, `failed`, `passRatePercent`, `firstFailures`[{`time`, `channel`, `id`, `data`, `stored`, `computed`}]}, plus `unresolved` and `tooShort` where frames could not be checked.

get_diagnostics

The Diagnostics pane as it stands: ECU file, link, session, the fault codes and data identifiers read so far, the file's actions and security levels, the last status and log.

No arguments.

Hint: read-only. Returns: `ecuFile`; `fileSummary`; `fileError`; `cddError`; `protocol`; `channel`; `availableChannels`; `requestId`; `responseId`; `bitrate`; `connected`; `busy`; `session`; `faults`[as in

© 2026 Altech Electronics Nordic AB. All rights reserved.

read_fault_codes]; `faultSummary`; `identifiers`[as in read_data_identifier]; `actions`[{`name`, `steps`, `inputs`[{`name`, `bytes`}]}]; `securityLevels`; `catalogue`; `hasReflash`; `status`; `statusIsError`; `log`; `summary`.

list_ecu_services

Searches the entries of the ECU file's CANdela catalogue (identifiers, routines, sessions, I/O controls) with the services each offers and the inputs they take. run_ecu_service runs one.

Argument	Type	Req.	Meaning
`query`	string		Name, identifier or class: substring, * wildcard or regex.
`limit`	integer		Maximum entries (default 100, at most 1000).
`offset`	integer		Skip this many matches, for paging.

Hint: read-only. Returns: `entries`[{`name`, `id`, `class`, `lastValue`, `services`[{`name`, `request`, `inputs`}]}]; `matching`; `summary`.

get_flash

The Flashing pane as it stands: the targets on offer and their routes, the form, whether a run is going, its progress, verdict and log - and the same for a UDS reflash in the Diagnostics pane.

No arguments.

Hint: read-only. Returns: `targets`[{`name`, `route`, `device`, `firmware`}]; `selectedTarget`; `firmware`; `channel`; `availableChannels`; `bitrate`; `requestId`; `responseId`; `extendedIds`; `host`; `port`; `vxToolFolder`; `vxToolSummary`; `canFlash`; `why`; `running`; `progress` (0 to 1); `status`; `statusIsError`; `log`; `udsReflash` {`ecuFile`, `hasSequence`, `firmware`, `running`, `progress`, `status`, `statusIsError`}; `summary`.

read_guide

Reads one of CalTool's guides for AI clients: 'tools' (how keys, permissions, data, calibration, the CAN trace, diagnostics and flashing work - read before changing anything), 'computed-channels' (the script contract and helpers for evaluate_script / add_computed_channel), 'can-scripts' (the JavaScript API and examples for save_can_script), or 'uds-files' (the .uds ECU description format behind actions and UDS reflashes).

Argument	Type	Req.	Meaning
`topic`	string	yes	Which guide. One of: tools, computed-channels, can-scripts, uds-files.

Hint: read-only. Returns: `topic`; `text` (the guide, as Markdown); `summary`.

9.2. Analyse tools

Always allowed while AI access is on. Computes over the data and stores nothing.

© 2026 Altech Electronics Nordic AB. All rights reserved.

find_events

Finds where a signal satisfies a condition: 'above'/'below'/'equals' a threshold (as intervals - 'equals' 11 on a state signal gives each stay in state 11, its start being the entry), 'crosses_up'/'crosses_down' through it (as instants), 'changes' (any change of value, for enums and flags), 'max'/'min' (the extreme in the range). Use it to locate things before reading data around them. With several 'keys' the events come back as one timeline sorted by time, each naming its key - e.g. 'changes' over a set of state signals shows what switched in which order.

Argument	Type	Req.	Meaning
`key`	string		Signal key (or use 'keys').
`keys`	array of string		Several signal keys, for one merged timeline (at most 50).
`condition`	string	yes	The condition. One of: above, below, equals, crosses_up, crosses_down, changes, max, min.
`threshold`	number		The level for above/below/equals/crosses (an enum's raw value).
`minDuration`	number		For above/below: ignore intervals shorter than this many seconds.
`from`	number		Start time.
`to`	number		End time.
`limit`	integer		At most this many events (default 200, max 2000).
`source`	string		Where to read samples: 'log' (the loaded log file), 'recording' (the current or last recording), or 'auto' (default: what the app shows - the recording while it captures or after it stopped, the log once one is opened). One of: auto, log, recording.
`file`	string		Read this log file (full path, e.g. an earlier recording from get_state's savedRecordings) instead of the session's log or recording. Nothing on screen changes. Times are the file's own; a key matches by channel key, else by a name only one channel of the file has.

Hint: read-only. Returns: `dataSource` (`log` or `recording`: what was read) and, when the other source also holds data, `otherSource` (a sentence naming it); `key` (one signal) or `keys` (several); `condition`; `threshold`; `from`; `to`; `samplesScanned`; `eventCount`; `events`; `truncated`; `samplePeriods` {key: seconds}; `note`; `summary`. An event is {`time`, `endTime`, `duration`, `extreme`, and `openAtEnd` if still true at the end} for above/below/equals, {`time`, `value`, `previous`} for a crossing, {`time`, `from`, `to`, `fromText`, `toText`} for changes, {`time`, `value`} for max/min. With several keys the events of all signals are sorted by `time` and each carries its `key`; `limit` applies to the merged list. Keys that could not be answered are listed in `failed`[{`key`, `error`}] and the others are still answered; the call fails only when no key can be answered. Event times are sample times, accurate to the signal's sample period. With `file`: `dataSource` is `file`, plus `file`, `fileStart` and `timeline` (the times are the file's own).

© 2026 Altech Electronics Nordic AB. All rights reserved.

summarize_events

For each occurrence of a condition on a trigger signal (as in find_events - e.g. condition 'equals', threshold 11 on a state signal: every stay in state 11), the chosen signals' values at the event's start and end and their min/max/mean over it. One call gives a table such as 'every gear shift with its torque, speed and duration'. 'before'/'after' widen each window. 'followUps' adds timing: for each occurrence, the time from its start until each follow-up condition on another signal first holds (e.g. from shift request until neutral, until sync ok, until gear in), searched up to the next occurrence or 'within' seconds.

Argument	Type	Req.	Meaning
`trigger`	string	yes	The signal whose condition defines the events.
`condition`	string	yes	The condition. One of: above, below, equals, crosses_up, crosses_down, changes.
`threshold`	number		The level for above/below/equals/crosses (an enum's raw value).
`minDuration`	number		For above/below/equals: ignore intervals shorter than this many seconds.
`keys`	array of string		The signals to summarise per event (at most 30; include the trigger to see it too). May be empty when followUps are given.
`followUps`	array of object		Conditions to time from each occurrence's start (at most 12).
`followUps[].key`	string	yes	The signal to watch.
`followUps[].condition`	string	yes	What to wait for. One of: above, below, equals, crosses_up, crosses_down, changes.
`followUps[].threshold`	number		The level for above/below/equals/crosses (an enum's raw value).
`followUps[].label`	string		Column name in the answer (default: key, condition and threshold).
`within`	number		Search each follow-up at most this many seconds after the occurrence's start (default: up to the next occurrence).
`valuesAtFollowUps`	boolean		Also give the 'keys' signals' values at each follow-up's instant (default false).
`before`	number		Seconds before each event's start to include in min/max/mean (default 0).
`after`	number		Seconds after each event's end to include (default 0).
`from`	number		Only events from this time.
`to`	number		Only events up to this time.
`limit`	integer		At most this many events (default 50, max 500).

© 2026 Altech Electronics Nordic AB. All rights reserved.

Argument	Type	Req.	Meaning
source	string		Where to read samples: 'log' (the loaded log file), 'recording' (the current or last recording), or 'auto' (default: what the app shows - the recording while it captures or after it stopped, the log once one is opened). One of: auto, log, recording.
file	string		Read this log file (full path, e.g. an earlier recording from get_state's savedRecordings) instead of the session's log or recording. Nothing on screen changes. Times are the file's own; a key matches by channel key, else by a name only one channel of the file has.

Hint: read-only. Returns: dataSource (log or recording: what was read) and, when the other source also holds data, otherSource (a sentence naming it); trigger (its key); condition; threshold; window (what each event's statistics cover); eventCount; truncated; events[{time, endTime and duration for an interval, openAtEnd, from/to (state names) for a change, values{key: {atStart, atEnd, min, max, mean}}, followUps{label: {delay (seconds from the event's start until the follow-up condition first held; 0 with alreadyTrue when it held at the start), and the chosen signals' values at that moment} or null when it did not hold before the next event or within within}}]; samplePeriods {key: seconds}; format; summary. An enum's values are state names and it has no mean. With followUps, keys may be empty. The trigger failing fails the call; another key that could not be answered is listed in failed[{key, error}]. With file: dataSource is file, plus file, fileStart and timeline (the times are the file's own).

scan_logs

Runs statistics or an event search for signals across a set of log files WITHOUT opening them as the session's log - the loaded log, plots and recording stay as they are. Give 'folder' (with 'pattern') and/or 'paths'. Times are each file's own (0 = its first sample); 'start' is the file's recording time. A signal is matched by key, else by a name only one channel of that file has (the answer says which channel was read).

Argument	Type	Req.	Meaning
folder	string		Folder to scan.
pattern	string		File pattern in the folder (default *.mf4).
recursive	boolean		Include subfolders (default false).
paths	array of string		Specific files, instead of or besides a folder.
keys	array of string	yes	Signal keys or names (at most 20).
mode	string		'statistics' (default: count/min/max/mean/first/last per signal) or 'events' (find_events per signal). One of: statistics, events.
condition	string		For 'events': the condition, as in find_events. One of: above, below, equals, crosses_up, crosses_down, changes, max, min.

© 2026 Altech Electronics Nordic AB. All rights reserved.

Argument	Type	Req.	Meaning
threshold	number		For 'events': the level for above/below/crosses.
minDuration	number		For 'events' above/below: ignore shorter intervals (seconds).
limit	integer		For 'events': at most this many events per signal per file (default 20, max 500).
maxFiles	integer		At most this many files, newest first (default 50, max 500).

Hint: read-only. Returns: mode; filesScanned; filesWithData; files[{file, start, signals, missing[{key, error}], or error for a file that could not be read}]; condition and threshold in mode events; note when more files matched than maxFiles; summary. A signal is {key, channel (the channel read, where it differs from the key asked for), unit, firstTime, lastTime} with count, min, max, mean, first, last in mode statistics, or eventCount, events (as in find_events) and truncated in mode events. Times are seconds from each file's own first sample. The session's log and recording are not changed.

evaluate_script

Runs a computed-channel script over the loaded log or recording WITHOUT saving it, and returns statistics and a decimated result. The script defines function step(t, x) and runs once per sample of the trigger input; x.<alias> holds each input; helpers prev('alias', n), prevOut(n), dt(), first(), changed, posEdge, negEdge. Call read_guide('computed-channels') first. Use this to try an analysis; add_computed_channel to keep it.

Argument	Type	Req.	Meaning
inputs	array of object	yes	Input signals.
inputs[].key	string	yes	Signal key.
inputs[].alias	string		Identifier used in the script (default: derived from the name).
script	string	yes	JavaScript defining function step(t, x) returning the output value.
trigger	string		Key of the input whose samples drive the script (default: the first input).
maxPoints	integer		Decimated points to return (default 200).
source	string		Where to read samples: 'log' (the loaded log file), 'recording' (the current or last recording), or 'auto' (default: what the app shows - the recording while it captures or after it stopped, the log once one is opened). One of: auto, log, recording.

Hint: read-only. Returns: dataSource (log or recording: what was read) and, when the other source also holds data, otherSource (a sentence naming it); succeeded; error; failedSamples; sampleCount; and with samples firstTime, lastTime, min, max, mean, points as [time, value] pairs; summary. Nothing is

© 2026 Altech Electronics Nordic AB. All rights reserved.

stored.

render_plot

Renders a plot as a PNG image exactly as shown (curves, legend with readouts, axes, cursors). 'plot' is a plot card key (P:#n), a plot card's title, or a plot pane tab name; omitted = the selected plot pane tab or the first plot card. Only works in the running app window.

Argument	Type	Req.	Meaning
`plot`	string		Which plot. Zoom and arrange it first with set_plot_view (show=true brings it on screen).

Hint: read-only. Returns: `content` holds the text {`bytes`, `summary`} and then an item of type `image` with the base64 PNG in `data`. Refused for a plot that is not on screen: set_plot_view with `show` brings it there.

9.3. Workspace tools

Allowed by the permission **Change the workspace**. Everything here shows on screen.

add_computed_channel

Creates a computed channel (a JavaScript step(t, x) over other signals) the user can plot and record like any signal. Its key becomes 'Calc/<name>'. Try the script with evaluate_script first; call read_guide('computed-channels') for the script contract.

Argument	Type	Req.	Meaning
`name`	string	yes	Channel name (no '/').
`unit`	string		Unit text, optional.
`inputs`	array of object	yes	Input signals.
`inputs[].key`	string	yes	Signal key.
`inputs[].alias`	string		Identifier used in the script (default derived from the name).
`trigger`	string		Key of the input whose samples drive the script (default the first).
`script`	string	yes	JavaScript defining function step(t, x).
`plot`	boolean		Also open a plot card showing the channel (default false).

Hint: additive. Returns: `summary` naming the new key.

update_computed_channel

Replaces a computed channel's definition (script, inputs, unit, name). Plots showing it follow the name.

© 2026 Altech Electronics Nordic AB. All rights reserved.

Argument	Type	Req.	Meaning
`name`	string	yes	The existing channel's name.
`newName`	string		A new name, optional.
`unit`	string		Unit text.
`inputs`	array of object	yes	Input signals.
`inputs[].key`	string	yes	Signal key.
`inputs[].alias`	string		Alias.
`trigger`	string		Trigger input key.
`script`	string	yes	JavaScript defining function step(t, x).

Hint: additive. Returns: `summary` naming the key.

remove_computed_channel

Deletes a computed channel.

Argument	Type	Req.	Meaning
`name`	string	yes	The channel's name.

Hint: destructive. Returns: `summary`.

add_marker

Puts a marker on the timeline, shown as a flag on every plot - the way to point the user at a finding. A comment explains it; an endTime makes it a span.

Argument	Type	Req.	Meaning
`time`	number		Time in seconds (default: the cursor, or 'now' while recording).
`name`	string		Short label on the flag (default 'Marker n').
`comment`	string		Explanation shown in the flag's tooltip and the Markers list.
`endTime`	number		End of a span, optional.

Hint: additive. Returns: The marker as in list_markers, with `summary`.

update_marker

Changes a marker's name, comment or time. The marker is picked by its index (list_markers) or its exact name.

© 2026 Altech Electronics Nordic AB. All rights reserved.

Argument	Type	Req.	Meaning
index	integer		Marker index from list_markers.
name	string		Marker name, if no index.
newName	string		New label.
comment	string		New comment.
time	number		New time.
endTime	number or null		New end time (a span); null makes it a single instant.

Hint: additive. Returns: The marker as in list_markers, with summary.

remove_marker

Removes a marker by index (list_markers) or exact name.

Argument	Type	Req.	Meaning
index	integer		Marker index.
name	string		Marker name, if no index.

Hint: destructive. Returns: summary.

set_cursor

Moves the plot cursor(s): every plot and readout follows cursor A; cursor B makes the legends show min/max/delta between the two. Omit a value to leave it; null clears B. To zoom a plot, use set_plot_view.

Argument	Type	Req.	Meaning
a	number		Cursor A time in seconds.
b	number or null		Compare cursor B time; null removes it.
timeWindowMin	number		Also set every plot's visible time window start (set_plot_view zooms one plot and does more).
timeWindowMax	number		Also set every plot's visible time window end.

Hint: additive. Returns: a; b; summary. Refused while a recording captures. timeWindowMin and timeWindowMax zoom every plot; set_plot_view zooms one plot and does more.

add_page

Adds a workspace page (tab) and selects it.

Argument	Type	Req.	Meaning
name	string	yes	Page name.

© 2026 Altech Electronics Nordic AB. All rights reserved.

Hint: additive. Returns: `summary`.

select_page

Selects a workspace page by name or 1-based index, and shows the workspace if another pane is active.

Argument	Type	Req.	Meaning
`page`	string	yes	Page name or index.

Hint: additive. Returns: `summary`.

open_plot

Opens a new plot card on the selected page (or 'page') charting the given signals - measurements, log channels or computed channels. Measurements get their default raster so they are recorded. Returns the card key for add_to_plot / render_plot.

Argument	Type	Req.	Meaning
`keys`	array of string	yes	Signal keys to chart.
`title`	string		A title for the card, optional.
`page`	string		Page name or index; default the selected page.
`x`	number		Card position, optional.
`y`	number		Card position, optional.
`width`	number		Card width, optional.
`height`	number		Card height, optional.

Hint: additive. Returns: The card as in get_state, including its `key` for add_to_plot and render_plot; `summary`. Every key is resolved before the card opens: a key nothing has, a calibration object, or a bare name that names several signals refuses the call, naming the keys to choose from, and no card is opened.

add_to_plot

Adds signals to an existing plot card (by key or title) or plot pane tab (by name). 'subplot' puts them in a stacked subplot sharing the time axis - signals with different scales read better one above the other.

Argument	Type	Req.	Meaning
`plot`	string	yes	Plot card key (P:#n), card title, or pane tab name.
`keys`	array of string	yes	Signal keys.
`subplot`	integer		Which stacked subplot: 1 is the top plot (the default); one more than the plot has adds a new subplot at the bottom.

© 2026 Altech Electronics Nordic AB. All rights reserved.

Hint: additive. Returns: `summary` listing what the plot now shows and in how many subplots. A key nothing has, a calibration object, or a bare name that names several signals refuses the call, naming the keys to choose from, and nothing is added. `subplot` outside 1..count+1 is refused.

remove_from_plot

Removes signals from a plot.

Argument	Type	Req.	Meaning
`plot`	string	yes	Plot card key, card title, or pane tab name.
`keys`	array of string	yes	Signal keys as the plot lists them (get_state).

Hint: destructive. Returns: `summary` listing what the plot now shows.

open_readouts

Opens a card of live/at-cursor value readouts for measurements on the selected page (one list card). Measurements get their default raster.

Argument	Type	Req.	Meaning
`keys`	array of string	yes	Measurement keys.
`page`	string		Page name or index.
`x`	number		Position.
`y`	number		Position.

Hint: additive. Returns: The card as in get_state; `summary`.

open_calibration_cards

Opens calibration cards for characteristics and axes on the selected page: scalars as one list card, each curve/map/array/axis as its own card.

Argument	Type	Req.	Meaning
`keys`	array of string	yes	Characteristic or axis keys.
`page`	string		Page name or index.
`x`	number		Position.
`y`	number		Position.

Hint: additive. Returns: `cards`, each as in get_state; `summary`.

close_card

Closes a workspace card by its key (get_state lists them).

© 2026 Altech Electronics Nordic AB. All rights reserved.

Argument	Type	Req.	Meaning
`key`	string	yes	Card key, e.g. P:#3 or ML:#2.

Hint: destructive. Returns: `summary`.

set_raster

Assigns measurements to a DAQ raster (so they are streamed and recorded) or takes them off logging with raster null. describe_signal lists a measurement's rasters. The answer's 'takesEffect' says whether a running recording picks the change up now or it applies from the next recording.

Argument	Type	Req.	Meaning
`keys`	array of string	yes	Measurement keys.
`raster`	string or null	yes	Raster (DAQ event) name, 'default' for the device's default, or null to take them off logging.

Hint: additive. Returns: `takesEffect` (whether a running recording picks the change up now - an online device’s DAQ is reconfigured within a second and a newly assigned measurement gets its recording channel - or it applies from the next recording); `summary`.

open_log

Opens a log file (MDF4 .mf4, .csv, .mat, a GoPro .mp4) as the session's log; a log's devices named like workspace devices pair automatically, others keep the log's names.

Argument	Type	Req.	Meaning
`path`	string	yes	Full path to the file.

Hint: additive. Returns: `summary` with the number of signals.

close_log

Closes the loaded log and discards a stopped recording. Refused with unsaved markers or an unsaved recording unless 'discard' is true.

Argument	Type	Req.	Meaning
`discard`	boolean		Discard unsaved markers / recording.

Hint: destructive. Returns: `summary`.

start_recording

Starts a DAQ recording of every raster-assigned measurement of the online devices (plus ticked CAN trace buses and cameras). Needs a device online or a trace bus; the user may be asked about DAQ overload. A save in progress is finished first. Refused while a stopped recording is unsaved, unless

© 2026 Altech Electronics Nordic AB. All rights reserved.

'discard' is true.

Argument	Type	Req.	Meaning
`discard`	boolean		Replace a stopped recording that was never saved.

Hint: additive. Returns: `summary`: recording started, or why it did not.

stop_recording

Stops the recording. The data stays in memory for analysis (the cursor lands on its last sample) until saved or closed.

No arguments.

Hint: additive. Returns: `summary` with the last time and the number of channels.

save_recording

Saves the stopped recording as an MDF4 file with a measurement comment.

Argument	Type	Req.	Meaning
`path`	string	yes	Target .mf4 path.
`name`	string		Comment: name/author.
`project`	string		Comment: project.
`measurementObject`	string		Comment: what was measured.
`comment`	string		Comment text.

Hint: destructive. Returns: `summary`.

open_workspace

Opens a workspace (.ctws), replacing the session: devices, pages, computed channels. Refused while a device is online or a recording captures; refused with unsaved work unless 'discard' is true.

Argument	Type	Req.	Meaning
`path`	string	yes	Full path to the .ctws file.
`discard`	boolean		Discard unsaved workspace changes, markers and recording.

Hint: destructive. Returns: `summary` with CalTool's status line, which names anything it could not restore.

© 2026 Altech Electronics Nordic AB. All rights reserved.

save_workspace

Saves the workspace (.ctws) to its current file, or to 'path'.

Argument	Type	Req.	Meaning
path	string		Target path; default the open workspace file.

Hint: destructive. Returns: summary.

set_plot_view

Zooms and arranges a plot so it shows the behaviour you want to look at or point out; follow it with render_plot to see the picture. Time axis: timeMin/timeMax, or timeCenter/timeSpan around an instant (an event from find_events), or fitTime for everything, or zoomToCursors. The window is clamped to the data, and while 'Sync time' is on (the default) every plot follows. Y axes: fitY fits every drawn signal to its samples inside the window; 'signals' sets one signal's range, visibility, axis pin, dots or colour. 'select' picks the signal whose Y axis is shown. 'show' brings the plot's page on screen and 'maximize' lets the card fill the page. Returns the plot as get_plot_view describes it.

Argument	Type	Req.	Meaning
plot	string		Plot card key (P:#n), card title, or pane tab name; default the plot on the selected page.
timeMin	number		Time window start in seconds.
timeMax	number		Time window end in seconds.
timeCenter	number		Centre of the time window; with timeSpan.
timeSpan	number		Width of the time window in seconds; with timeCenter, or alone to keep the current centre.
fitTime	boolean		Show the whole time span of the plot's data.
zoomToCursors	boolean		Zoom to the span between cursor A and B, with a little room either side.
fitY	boolean		Fit the Y axis of every drawn signal to its samples inside the time window.
signals	array of object		Per-signal changes.
signals[].key	string	yes	The signal, as the plot lists it (get_plot_view) or its short name.
signals[].yMin	number		Y-axis lower end; give yMax too.
signals[].yMax	number		Y-axis upper end.
signals[].fitY	boolean		Fit this signal's Y axis to its samples inside the time window.
signals[].resetY	boolean		Back to the default Y range (the A2L limits, else the data).
signals[].visible	boolean		Draw the curve (false hides it; the legend row stays).
signals[].pinAxis	boolean		Keep this signal's Y axis shown beside the selected one's.

© 2026 Altech Electronics Nordic AB. All rights reserved.

Argument	Type	Req.	Meaning
signals[].dotsOnly	boolean		Draw the samples as dots without the line.
signals[].color	string		Line colour as #RRGGBB.
select	string		The signal to select: its Y axis is the one shown and its curve is drawn thicker.
xAxisSignal	string or null		Plot against this signal instead of time (an X-Y plot); null puts time back.
syncTimeAxes	boolean		Whether zooming one plot moves every plot (applies to all plots).
cursorStatistics	array of enum		What the legends show between cursor A and B (applies to all plots); empty for none. Each one of: delta, min, max, average.
show	boolean		Select the plot's page and show the workspace, so the plot is on screen for render_plot.
maximize	boolean		Let the plot card fill its page (false puts it back at its own size).

Hint: additive. Returns: The plot as get_plot_view describes it, after the change, with summary saying what was done and note when the plot is not on screen. The time window that comes back is the one the plot shows: a window reaching outside the data is held to it. While syncTimeAxes is on, every plot takes the same window. Refused for a signal the plot does not chart (the answer lists what it charts), for timeMax not greater than timeMin, for zoomToCursors without both cursors, for fitTime on a plot with no data, and for show or maximize on a plot pane tab.

configure_log_can_decoding

Ties a DBC to each CAN bus of the loaded log (the Logs pane's Pair DBC), so every signal the database describes becomes a log channel - plotted, read and searched like any other (list_signals kinds ['log']). A bus decodes only its own frames: two buses sharing an identifier stay apart. 'set' replaces the whole decoding, 'add' changes only the buses named, 'list' reports it, 'clear' removes it. Saved with the workspace for this log. With 'file', decodes another MF4 for the data tools' 'file' argument instead; nothing on screen changes. Per bus the answer gives frames, frames decoded, messages and signals, the database's messages that never appeared, and example keys.

Argument	Type	Req.	Meaning
action	string		set (default), add, list or clear. One of: set, add, list, clear.
mappings	array of object		Bus → database.
mappings[].bus	integer	yes	The log's bus number (get_state's log.canBuses, or 'list').
mappings[].dbc	string or null		Full path to the .dbc; empty or null leaves the bus undecoded.

© 2026 Altech Electronics Nordic AB. All rights reserved.

Argument	Type	Req.	Meaning
`useTraceInterfaces`	boolean		Buses not named in 'mappings' take the DBC of the CAN trace card at the same position (bus n → card n, the rule open_can_trace_file uses). Default true when 'mappings' is empty.
`copyFromLog`	boolean		With 'file': start from the loaded log's decoding, bus by bus.
`file`	string		Decode this MF4 (full path) for the data tools' 'file' argument instead of the loaded log.

Hint: additive. Returns: `file`; `buses`[{`bus`, `label`, `frames`, `decodedBy` (`this bus's database`, `any paired database describing the identifier` or `not decoded`), and when decoded `dbc`, `framesDecoded`, `messages`, `signals`, `neverSeenCount` with `neverSeen` (up to 20 of the database's messages that never appeared, as "0x123 Name"), `exampleKeys` (up to five)}]; `decodedSignals`; `keys` (how decoded signals are keyed); `notes` (a bus that took a trace card's database); `summary`. With `file`, also `note`: the signals are read with the data tools' `file` argument and nothing on screen changes. Frames decoded far below frames on a bus is the sign of a database that does not belong to it.

9.4. Calibration tools

Allowed by the permission **Change calibration values on the desktop and in files**.

set_calibration

Changes calibration values. Each change names a characteristic (or READ_WRITE measurement, or axis) and gives 'value' (scalar), 'values' (curve/array: one number per element) or 'rows' (map: array of rows, Y outer, X inner). Values are physical, checked against the A2L's limits and shape. Offline: lands in the desktop store (marked ▲/▼ in the cards). Online: written to the ECU after the user confirms the batch in a dialog. A device in a sync group changes its partners too: they are checked against their own limits, and an online partner counts as an online write even when the named device is offline. An online write is read back from the ECU before the answer: 'onEcu' says whether it holds the value. 'force' writes a value the desktop already holds (to push it to the ECU again); 'partial' applies the keys that exist and lists unknown ones under 'failed' instead of refusing the batch.

Argument	Type	Req.	Meaning
`changes`	array of object	yes	The changes to apply together.
`changes[].key`	string	yes	Characteristic/axis key.
`changes[].value`	number or string		Scalar value, or for an enum its state's raw index or state text.
`changes[].values`	array of number		Curve or array values.
`changes[].rows`	array of array of number		Map rows (Y outer, X inner).

© 2026 Altech Electronics Nordic AB. All rights reserved.

Argument	Type	Req.	Meaning
note	string		Why - shown to the user in the confirmation and the activity log.
force	boolean		Write values the desktop already holds too (default false: they are skipped as 'already that value').
partial	boolean		Apply the changes whose key exists and list unknown keys under 'failed' (default false: one unknown key refuses the batch). Shape and limit errors always refuse it.

Hint: destructive. Returns: changes[{key, applied, and if applied old, new, toEcu, mirroredTo[{key, old, new, toEcu}], or if not reason (for instance the desktop already holds that value: force writes it again)}]; summary. For a write to an online ECU the answer comes after the write was read back: onEcu (true when the ECU holds the new value, false when it did not keep it - the desktop is then put back to what the ECU holds - null when it could not be read back) and ecuHolds (the value read back, or why there is none). With partial, keys that do not resolve are skipped and listed in failed[{key, error}]. The batch is validated whole: one bad entry (an unknown key without partial, a wrong shape, a value outside the limits) refuses all of it and writes nothing.

load_calibration_file

Loads a .cdfx or .dcm file into a device's values in the desktop store, offline only: refused while the device or a sync partner is online, because a load writes every object straight to the ECU. To change an online ECU use set_calibration, which the user confirms.

Argument	Type	Req.	Meaning
path	string	yes	Full path to a .cdfx or .dcm file.
device	string		Device name; default the Calibration pane's selected device.

Hint: destructive. Returns: summary with how many objects matched and how many the file lacked; warnings.

save_calibration_file

Saves a device's calibration values from the desktop store to a .cdfx or .dcm file.

Argument	Type	Req.	Meaning
path	string	yes	Target path ending in .cdfx or .dcm.
device	string		Device name; default the Calibration pane's selected device.

Hint: destructive. Returns: summary with the number of objects written.

© 2026 Altech Electronics Nordic AB. All rights reserved.

9.5. Live ECU tools

Allowed by the permission **Write to an online ECU**.

go_online

Connects a device (XCP session: EPK check, calibration compare, DAQ setup). The user may be asked about a calibration sync or an EPK mismatch in CalTool.

Argument	Type	Req.	Meaning
`device`	string	yes	Device name.

Hint: destructive. Returns: `online`; `status`; `detail`; `summary`. Only `summary` if the device was already online.

go_offline

Disconnects a device (a recording continues if other devices still log).

Argument	Type	Req.	Meaning
`device`	string	yes	Device name.

Hint: destructive. Returns: `summary`.

9.6. CAN trace tools

Allowed by the permission **Use the CAN trace**. Sets up and listens. Nothing here sends a frame.

configure_can_interface

Sets up a CAN trace interface card, or adds one when 'interface' is left out. Channel, bitrate and CAN FD can only change while the card is disconnected. Channel names are in get_can_trace; 'Simulation' works without hardware.

Argument	Type	Req.	Meaning
`interface`	string		The card to change; leave out to add a new one.
`name`	string		A new name for the card.
`dbc`	string		Full path to a .dbc for this card; an empty string removes it.
`recordWithMeasurements`	boolean		Whether this card's frames go into a measurement recording.
`channel`	string		A CAN channel by name.
`bitrate`	integer		Nominal bitrate in bit/s.
`canFd`	boolean		Open the channel for CAN FD.

© 2026 Altech Electronics Nordic AB. All rights reserved.

Argument	Type	Req.	Meaning
dataBitrate	integer		CAN FD data bitrate in bit/s.
nominalSamplePoint	number		Nominal sample point in percent (default 80).
dataSamplePoint	number		Data sample point in percent (default 80).

Hint: additive. Returns: interface {name, channel, bitrate, bitrateSharedNote, canFd, dataBitrate, dbc, dbcSummary, connected, reconnecting, status, statusIsError, health, errorFrames, recordWithMeasurements}; summary.

remove_can_interface

Removes a disconnected interface card and its share of the trace.

Argument	Type	Req.	Meaning
interface	string	yes	The card.

Hint: destructive. Returns: summary.

connect_can_interface

Starts listening on an interface card, or on every disconnected card. Listening sends nothing. The first card to connect restarts the trace.

Argument	Type	Req.	Meaning
interface	string		The card; leave out for all.

Hint: additive. Returns: interfaces[as in configure_can_interface]; summary. An error only when no interface connected.

disconnect_can_interface

Stops an interface card, or every card. Scripts using it stop and its cyclic messages end.

Argument	Type	Req.	Meaning
interface	string		The card; leave out for all.

Hint: additive. Returns: summary.

clear_can_trace

Empties the captured frames (not the setup).

No arguments.

Hint: destructive. Returns: summary.

© 2026 Altech Electronics Nordic AB. All rights reserved.

set_can_id_filter

Shows or hides identifiers in the trace and in get_can_frames. Without 'ids' it applies to every identifier seen so far on the card, or on all cards.

Argument	Type	Req.	Meaning
`show`	boolean	yes	true shows them, false hides them.
`ids`	array of string		Identifiers. Decimal, or hex with 0x.
`interface`	string		Only this card.

Hint: additive. Returns: `summary` (how many identifiers changed).

set_can_trace_view

The trace's display: one row per identifier or one per frame, hex or decimal.

Argument	Type	Req.	Meaning
`fixedIds`	boolean		true: one row per identifier. false: one row per frame.
`decimal`	boolean		true: decimal. false: hex.

Hint: additive. Returns: `summary`.

start_can_log

Streams every captured frame to a file as it arrives; the format follows the extension (.blf, .mf4, .txt). A log already running is closed first.

Argument	Type	Req.	Meaning
`path`	string	yes	Full path of the log file.

Hint: destructive. Returns: `summary`.

stop_can_log

Stops the file log and finishes the file.

No arguments.

Hint: additive. Returns: `summary`.

open_can_trace_file

Replaces the trace with the frames of a .blf, .mf4 or .txt file. Every interface must be disconnected.

© 2026 Altech Electronics Nordic AB. All rights reserved.

Argument	Type	Req.	Meaning
path	string	yes	Full path to the file.

Hint: destructive. Returns: frames; summary.

configure_can_transmit

Adds a row to the transmit list, or changes the row at 'index'. Build it from a DBC 'message' (which sets the identifier, length and frame kind) with physical 'signals', 'counters' and 'checksums', or raw with 'id' and 'data'. Nothing is sent until send_can_message or set_can_cyclic. A running cycle picks changes up on its next send.

Argument	Type	Req.	Meaning
index	integer		The row to change (from get_can_trace); leave out to add one.
interface	string		The card to send on.
message	string		A message of that card's DBC, by name.
id	string		Identifier of a raw frame. Decimal, or hex with 0x.
extended	boolean		29-bit identifier.
fd	boolean		CAN FD frame.
length	integer		Payload bytes: 0-8, and 12, 16, 20, 24, 32, 48, 64 for FD.
data	array or string		Payload bytes. An array of numbers (a string item may be 0x-prefixed), or one string of hex digits such as "22 F1 86".
cycleMs	number		Cycle time in ms, 1 to 3600000.
signals	object		Signal name to physical value, or to a state name for a signal with a value table.
counters	array of object		Signals filled by a rolling counter on every send.
counters[].signal	string	yes	Signal name.
counters[].step	integer		Added on every send; default 1.
counters[].min	integer		Default 0.
counters[].max	integer		Default the signal's largest value.
checksums	array of object		Signals filled by a checksum on every send.
checksums[].signal	string	yes	Signal name.
checksums[].algorithm	string	yes	crc8, crc8h2f, crc16, crc32, crc32p4, e2e-p01, e2e-p02, e2e-p05 or e2e-p11.
checksums[].dataId	string		E2E Data ID; 16 comma-separated values for Profile 2.

Hint: additive. Returns: row {index, interface, message, id, extended, fd, length, data, cycleMs, cyclic, canSend, valid, sent, status, signals[{name, value, unit, fill, states, present, problem}]}; summary.

© 2026 Altech Electronics Nordic AB. All rights reserved.

remove_can_transmit

Removes a row of the transmit list, stopping its cycle.

Argument	Type	Req.	Meaning
`index`	integer	yes	The row.

Hint: destructive. Returns: `summary`.

save_can_script

Stores a CAN script under a name, replacing the source of a script of that name; a running script keeps its old source until it is run again. Refused if the source does not parse. read_guide('can-scripts') has the API.

Argument	Type	Req.	Meaning
`name`	string	yes	The script's name.
`source`	string	yes	JavaScript.

Hint: additive. Returns: `summary`.

remove_can_script

Stops and removes a script.

Argument	Type	Req.	Meaning
`name`	string	yes	The script.

Hint: destructive. Returns: `summary`.

configure_can_replay

Sets up the replay of a CAN log (.blf, .mf4, .txt): the file, which file channel goes out on which interface card, which identifiers are sent, speed, looping. Sends nothing; control_can_replay starts it.

Argument	Type	Req.	Meaning
`path`	string		Full path to the log.
`close`	boolean		Close the loaded log.
`map`	array of object		Channel mapping; channels not named stay as they are.
`map[].channel`	integer	yes	The file's channel number.
`map[].interface`	string or null		The card, or null to leave the channel unsent.
`sendAll`	boolean		Tick every identifier first.
`only`	array of string		Send only these identifiers. Decimal, or hex with 0x.

© 2026 Altech Electronics Nordic AB. All rights reserved.

Argument	Type	Req.	Meaning
exclude	array of string		Do not send these identifiers. Decimal, or hex with 0x.
speed	number		1 is the log's own timing, 2 twice as fast.
loop	boolean		Start over at the end.
includeTransmitted	boolean		Also send frames the logger itself transmitted.

Hint: additive. Returns: replay {file, summary, selection, channels[{channel, name, frames, interface}], speed, loop, includeTransmitted, running, paused, progress, status, isError}; summary.

9.7. CAN transmit tools

Allowed by the permission **Send on a CAN bus**. There is no confirmation of each call.

send_can_message

Sends one frame now: the transmit row at 'index' (its counters and checksums are applied), or a raw frame from 'interface', 'id' and 'data'. The interface must be connected.

Argument	Type	Req.	Meaning
index	integer		A transmit row.
interface	string		The card, for a raw frame.
id	string		Identifier of a raw frame. Decimal, or hex with 0x.
extended	boolean		29-bit identifier.
fd	boolean		CAN FD frame.
data	array or string		Payload bytes of a raw frame. An array of numbers (a string item may be 0x-prefixed), or one string of hex digits such as "22 F1 86".

Hint: destructive. Returns: summary, and row as in configure_can_transmit when a row was sent.

set_can_cyclic

Starts or stops the cyclic sending of a transmit row at its cycle time.

Argument	Type	Req.	Meaning
index	integer	yes	The row.
running	boolean	yes	true starts, false stops.

Hint: destructive. Returns: row as in configure_can_transmit; summary.

run_can_script

Starts a stored script. The interfaces it uses must be connected. The answer includes what it printed

© 2026 Altech Electronics Nordic AB. All rights reserved.

in its first moments; get_can_trace has the rest.

Argument	Type	Req.	Meaning
`name`	string	yes	The script.

Hint: destructive. Returns: `script` {`name`, `running`, `status`, `isError`, `output`, `source`}; `summary`. An error if the script stopped with a fault in its first moments.

stop_can_script

Stops a script, or every script.

Argument	Type	Req.	Meaning
`name`	string		The script; leave out for all.

Hint: destructive. Returns: `summary`.

control_can_replay

Starts, pauses, resumes or stops the replay. The mapped interfaces must be connected.

Argument	Type	Req.	Meaning
`action`	string	yes	What to do. One of: start, pause, resume, stop.

Hint: destructive. Returns: `replay` as in configure_can_replay; `summary`.

9.8. Diagnostics tools

Allowed by the permission **Run diagnostic sessions**. There is no confirmation of each call.

diagnostics_connect

Opens the diagnostic session: the channel, StartCommunication under KWP2000, the file's session (extended without a file), TesterPresent while it stays open. Any field given is applied first. An XCP session on the same channel steps aside until the link closes. Works without a file on the identifiers alone.

Argument	Type	Req.	Meaning
`ecuFile`	string		Full path to a .uds or .cdd file; an empty string works without one.
`requestId`	string		Request CAN identifier, where the file does not fix it.
`responseId`	string		Response CAN identifier, where the file does not fix it.
`extendedIds`	boolean		29-bit identifiers.
`protocol`	string		Without a file: the dialect. One of: uds, kwp2000.

© 2026 Altech Electronics Nordic AB. All rights reserved.

Argument	Type	Req.	Meaning
variant	string		The CANdela variant to use.
channel	string		A CAN channel by name.
bitrate	integer		Nominal bitrate in bit/s.
canFd	boolean		Open the channel for CAN FD.
dataBitrate	integer		CAN FD data bitrate in bit/s.
nominalSamplePoint	number		Nominal sample point in percent (default 80).
dataSamplePoint	number		Data sample point in percent (default 80).

Hint: destructive. Returns: session; summary.

diagnostics_disconnect

Closes the diagnostic session and returns the ECU to its default session.

No arguments.

Hint: destructive. Returns: summary.

read_fault_codes

Reads the stored fault codes, explained from the ECU file and the SAE list.

Argument	Type	Req.	Meaning
mask	string		Which faults (UDS only); default all stored. One of: all, confirmed, pending, testFailed.

Hint: destructive. Returns: faults[{code, status, active, text, failureType, textSource}]; summary.

clear_fault_codes

Clears the ECU's stored fault codes, then reads back what it kept.

No arguments.

Hint: destructive. Returns: faults as in read_fault_codes (what the ECU kept); summary.

read_data_identifier

Reads a data identifier: one the ECU file declares (decoded as it says), or any by its hex number (4 digits; 2 digits is a KWP2000 local identifier), returned raw.

Argument	Type	Req.	Meaning
id	string		The file's name for it, or its identifier in hex.

© 2026 Altech Electronics Nordic AB. All rights reserved.

Argument	Type	Req.	Meaning
`all`	boolean		Read every identifier in the list instead.

Hint: destructive. Returns: `identifier` {`name`, `id`, `value`, `raw`, `isError`}, or with `all` `identifiers` of the same shape; `summary`.

read_ecu_memory

ReadMemoryByAddress: up to 4095 bytes (255 under KWP2000) as a hex dump.

Argument	Type	Req.	Meaning
`address`	string	yes	Start address. Decimal, or hex with 0x.
`length`	integer	yes	Bytes to read.

Hint: destructive. Returns: `dump` (address, hex and text per 16 bytes); `summary`.

run_diagnostic_action

Runs an 'action' block of the ECU file. 'inputs' answers its {ask:...} values. A 'confirm' step in the action is put to the person at the ECU in CalTool's window. cancel_diagnostic_run stops it.

Argument	Type	Req.	Meaning
`name`	string	yes	The action's name.
`inputs`	object		Input name to value (decimal, or hex with 0x).
`waitSeconds`	integer		How long this call waits for the run to end. The run goes on after the call returns; poll the state tool. Default 120.

Hint: destructive. Returns: `running` (false once it has ended); `log` (the pane's latest lines); `summary`.

run_ecu_service

Runs one service of a catalogue entry from list_ecu_services, building the request from the file's own layout and decoding the answer.

Argument	Type	Req.	Meaning
`entry`	string	yes	The entry's name or identifier.
`service`	string		The service's name; may be left out when the entry has one.
`inputs`	array of number or string		One value per input of the service, in order: a number, or a state's name.

Hint: destructive. Returns: `value`; `raw`; `fields`[{`name`, `value`, `raw`}]; `summary`.

© 2026 Altech Electronics Nordic AB. All rights reserved.

unlock_ecu

SecurityAccess with the seed & key the ECU file's 'security' line describes.

Argument	Type	Req.	Meaning
`level`	string		The level in hex; may be left out when the file has one.

Hint: destructive. Returns: `summary`.

send_diagnostic_request

Sends your own request bytes over the open session and returns the response. Anything the other tools do not cover: a write, a routine, a session change. A negative response comes back as an error naming the NRC.

Argument	Type	Req.	Meaning
`request`	string	yes	Request in hex, such as "22 F1 86".

Hint: destructive. Returns: `response` (hex, starting with the positive response identifier); `summary`. A negative response is an error naming the NRC.

cancel_diagnostic_run

Stops the action or reflash the Diagnostics pane is running.

No arguments.

Hint: destructive. Returns: `summary`.

9.9. Flash tools

Allowed by the permission **Reprogram ECUs**. There is no confirmation of each call.

flash_ecu

Reprograms an ECU over XCP on CAN or Ethernet, or hands the image to Vector's VX flash tool. 'target' is a name from get_flash; a device target presets its link. Sessions in the way go offline and come back. An image that does not match the device's A2L is warned about and flashed anyway.

Argument	Type	Req.	Meaning
`target`	string	yes	A target from get_flash, or a device's name for its usual route.
`firmware`	string		Full path to a .hex, .s19 or .bin; default the target's own image.
`baseAddress`	string		Load address of a .bin.

© 2026 Altech Electronics Nordic AB. All rights reserved.

Argument	Type	Req.	Meaning
`requestId`	string		XCP master CAN identifier.
`responseId`	string		XCP slave CAN identifier.
`extendedIds`	boolean		29-bit identifiers.
`host`	string		IP address or host name, for XCP on Ethernet.
`port`	integer		Port, for XCP on Ethernet.
`ipProtocol`	string		For XCP on Ethernet. One of: tcp, udp.
`vxToolFolder`	string		The folder holding VxEcuFlash.exe and the .vxp container.
`waitSeconds`	integer		How long this call waits for the run to end. The run goes on after the call returns; poll the state tool. Default 0.
`channel`	string		A CAN channel by name.
`bitrate`	integer		Nominal bitrate in bit/s.
`canFd`	boolean		Open the channel for CAN FD.
`dataBitrate`	integer		CAN FD data bitrate in bit/s.
`nominalSamplePoint`	number		Nominal sample point in percent (default 80).
`dataSamplePoint`	number		Data sample point in percent (default 80).

Hint: destructive. Returns: `running` (false once it has ended); `log` (the pane's latest lines); `summary`. An error if the run ended within the wait and failed.

flash_ecu_uds

Runs the 'flash' sequence of the ECU file over UDS. Any field given is applied first; a session that is open is closed, and the ECU restarts through its bootloader.

Argument	Type	Req.	Meaning
`ecuFile`	string		Full path to a .uds or .cdd file; an empty string works without one.
`requestId`	string		Request CAN identifier, where the file does not fix it.
`responseId`	string		Response CAN identifier, where the file does not fix it.
`extendedIds`	boolean		29-bit identifiers.
`protocol`	string		Without a file: the dialect. One of: uds, kwp2000.
`variant`	string		The CANdela variant to use.
`channel`	string		A CAN channel by name.
`bitrate`	integer		Nominal bitrate in bit/s.
`canFd`	boolean		Open the channel for CAN FD.
`dataBitrate`	integer		CAN FD data bitrate in bit/s.
`nominalSamplePoint`	number		Nominal sample point in percent (default 80).

© 2026 Altech Electronics Nordic AB. All rights reserved.

Argument	Type	Req.	Meaning
`dataSamplePoint`	number		Data sample point in percent (default 80).
`firmware`	string	yes	Full path to a .hex, .s19 or .bin.
`baseAddress`	string		Load address of a .bin.
`waitSeconds`	integer		How long this call waits for the run to end. The run goes on after the call returns; poll the state tool. Default 0.

Hint: destructive. Returns: `running` (false once it has ended); `log` (the pane's latest lines); `summary`. An error if the run ended within the wait and failed.

cancel_flash

Stops a running flash. The ECU may be left with an incomplete image.

No arguments.

Hint: destructive. Returns: `summary`.

9.10. Resources and prompts

The resources hold the same guides as `read_guide`. Many hosts never hand resources to a model, which is why the tool exists. The prompts are task descriptions in plain language, naming the tools to use, and can serve as a starting point for your own.

Resources

URI	MIME type	Content
`caltool://guide/tools`	text/markdown	How signals are addressed, what the access tiers mean, and how to analyse, visualise and calibrate through the tools.
`caltool://guide/computed-channels`	text/markdown	Reference for the JavaScript step(t, x) scripts behind computed channels and evaluate_script: helpers, idioms, examples.
`caltool://guide/can-scripts`	text/markdown	Reference and worked examples for the JavaScript behind the CAN trace pane's scripts.
`caltool://guide/uds-files`	text/markdown	The .uds file format: link settings, reflash sequences, actions, data identifiers and fault texts.
`caltool://workspace/state`	application/json	The same answer as the get_state tool, as a document.

Prompts

Name	Arguments	What it asks for
`analyse-log`	`focus`	Analyse the loaded log or recording: find patterns, anomalies and interesting events, mark them on the timeline and explain them.

© 2026 Altech Electronics Nordic AB. All rights reserved.

Name	Arguments	What it asks for
`explain-calibration`	`key` (required)	Explain a calibration object: what it does, its shape, axes and current values, and how it relates to the measurements around it.
`review-recording-health`	none	Check a recording for connection losses, overloads and gaps and report them.

© 2026 Altech Electronics Nordic AB. All rights reserved.

10. Typical sequences

These are the tool calls a capable model makes for common tasks, from a real session. Use them to test your model, and in your system prompt if it needs guidance. They double as acceptance tests: a model that cannot complete the first one will not manage the others.

10.1. Finding out what is there

"What is loaded, and what limits engine speed?"

```
get_state       {}
list_signals    {"query": "Rev*", "limit": 5}
  -> {"total":1,"offset":0,"returned":1,"signals":[{"key":"Engine/RevLimit",
      "kind":"characteristic","shape":"value · rpm","unit":"rpm"}], ...}
describe_signal {"key": "Engine/RevLimit"}
  -> {"key":"Engine/RevLimit","name":"RevLimit","device":"Engine",
      "kind":"characteristic","description":"Rev limiter","type":"Value",
      "shape":"value · rpm","unit":"rpm",
      "conversion":{"name":"CM_rpm","type":"Linear","factor":1,"offset":0},
      "lowerLimit":3000,"upperLimit":9000,"address":"0x40000102",
      "axes":[],"readOnly":false, ...}
```

10.2. Analysing a recording

"Where does engine speed peak? Mark it, and show me." Statistics first, because they are cheap, then events to locate things, then a small amount of data around them.

```
get_signal_data {"keys": ["Engine/EngineSpeed"]}
  -> {"dataSource":"recording","signals":[{"key":"Engine/EngineSpeed","unit":"rpm",
      "source":"recording","sampleCount":60,"firstTime":0,"lastTime":5.9,
      "count":60,"min":436,"max":7598,"mean":4794.1, ...}], ...}
find_events     {"key": "Engine/EngineSpeed", "condition": "max"}
  -> {"eventCount":1,"events":[{"time":1.2,"value":7598}],
      "truncated":false, ...}
add_marker      {"time": 1.2, "name": "Peak speed",
                 "comment": "Highest engine speed in the run: 7598 rpm"}
open_plot       {"keys": ["Engine/EngineSpeed", "Engine/EngineLoad"],
                 "title": "Peak speed"}
set_cursor      {"a": 1.2}
render_plot     {}
```

10.3. Investigating an event

"What happened when the engine state changed, and does it happen in the other runs?" One merged timeline of what changed, the values around the instant, then the same question to a folder of logs.

© 2026 Altech Electronics Nordic AB. All rights reserved.

`scan_logs` reads the files without replacing the log the user has open. The answers below show the shape; their values are examples.

```
find_events     {"keys": ["Engine/EngineState", "Engine/DiagStatus"],
                 "condition": "changes"}
  -> {"keys":["Engine/EngineState","Engine/DiagStatus"],"eventCount":3,
      "events":[{"time":2.1,"from":1,"to":2,"fromText":"...",
                 "toText":"...","key":"Engine/EngineState"},
                {"time":2.14,"from":0,"to":4,"key":"Engine/DiagStatus"},
                ...], "truncated":false, ...}
get_snapshot    {"keys": ["Engine/EngineSpeed", "Engine/EngineLoad"],
                 "time": 2.1, "offsets": [-0.2, -0.1, 0, 0.1, 0.2]}
  -> {"time":2.1,"times":[1.9,2.0,2.1,2.2,2.3],"signals":[
      {"key":"Engine/EngineSpeed","unit":"rpm","values":[...]},
      {"key":"Engine/EngineLoad","unit":"%","values":[...]}], ...}
scan_logs       {"folder": "C:\\Logs\\Bench", "keys": ["Engine/EngineSpeed"],
                 "mode": "events", "condition": "above", "threshold": 7000}
  -> {"mode":"events","filesScanned":12,"filesWithData":12,"files":[
      {"file":"C:\\Logs\\Bench\\Run_03.mf4","start":"...",
       "signals":[{"key":"Engine/EngineSpeed","eventCount":2,
                   "events":[...]}]}, ...], ...}
```

When one of several keys cannot be answered, for example a calibration value among signals, the answer holds the others and names that key under `failed`.

10.4. Zooming in on what was found

"Show me the overshoot." A plot of the whole run shows that something happened; the engineer wants to see what. `set_plot_view` moves the plot's time window and fits the Y axes to the samples inside it, and its answer already says what the data does there. `show` and `maximize` put the plot on screen at full size, which `render_plot` needs. The answers below show the shape; their values are examples.

```
find_events     {"key": "Engine/EngineSpeed", "condition": "above",
                 "threshold": 7000}
  -> {"eventCount":1,"events":[{"time":41.2,"value":7012,"end":41.9}], ...}
open_plot       {"keys": ["Engine/EngineSpeed", "Engine/EngineLoad"],
                 "title": "Overshoot"}
  -> {"key":"P:#3","kind":"plot", ...}
add_to_plot     {"plot": "P:#3", "keys": ["Engine/ThrottlePosition"],
                 "subplot": 2}
set_plot_view   {"plot": "P:#3", "timeCenter": 41.5, "timeSpan": 4,
                 "fitY": true, "select": "Engine/EngineSpeed",
                 "show": true, "maximize": true}
  -> {"plot":"P:#3","page":"Analysis","onScreen":true,"maximized":true,
      "timeWindow":{"min":39.5,"max":43.5},
```

© 2026 Altech Electronics Nordic AB. All rights reserved.

```
      "dataTimeSpan":{"first":0,"last":312.4},"xAxis":"time",
      "subplots":[{"subplot":1,"signals":[
        {"key":"Engine/EngineSpeed","unit":"rpm","color":"#2F6FDE",
         "visible":true,"selected":true,"yMin":5870,"yMax":7069,
         "inWindow":{"samples":400,"min":5927,"max":7012,"mean":6480.2}},
        ...]}, ...],
      "summary":"page 'Analysis' shown; maximized; time 39.5..43.5 s; ..."}
set_cursor      {"a": 41.2, "b": 41.9}
render_plot     {"plot": "P:#3"}
```

`timeMin` and `timeMax` set a window directly, `fitTime` shows the whole run again, and `zoomToCursors` zooms to the span between cursor A and B. One signal's range, visibility, colour or dots go in `signals`: `{"key": "Engine/EngineLoad", "yMin": 0, "yMax": 100}` or `{"key": "…", "visible": false}`. While **Sync time** is on, which is the default, every plot follows a zoom. `get_plot_view` reads a plot's window and ranges without changing them.

10.5. Tabulating occurrences

"Give me every stretch where the engine was running, with its speed and load." One call returns a row per occurrence, and a second gives the signals side by side over one of them. Condition `equals` finds each stay in one state; its start is the moment the state was entered. The answers show the shape; their values are examples.

```
summarize_events {"trigger": "Engine/EngineState", "condition": "equals",
                  "threshold": 3, "minDuration": 0.2,
                  "keys": ["Engine/EngineSpeed", "Engine/EngineLoad"]}
  -> {"dataSource":"log","trigger":"Engine/EngineState","eventCount":4,
      "events":[{"time":12.4,"endTime":14.9,"duration":2.5,"values":{
        "Engine/EngineSpeed":{"atStart":3810,"atEnd":1450,
                              "min":1450,"max":3810,"mean":2604.2},
        "Engine/EngineLoad":{"atStart":2,"atEnd":0,"min":0,"max":2,
                             "mean":0.4}}}, ...], ...}
get_signal_grid  {"keys": ["Engine/EngineState", "Engine/EngineSpeed"],
                  "from": 12, "to": 15, "step": 0.5}
  -> {"dataSource":"log","columns":["t","Engine/EngineState",
      "Engine/EngineSpeed"],"units":["s",null,"rpm"],
      "rows":[[12,"Idle (2)",850],[12.5,"Running (3)",3640], ...], ...}
```

An enum's values are its state names with the raw value in brackets. In `summarize_events` an enum has no `mean`.

10.6. Building a computed channel

"Give me engine speed in revolutions per second." Try the script without storing it, and keep it once it works. A model should call `read_guide` with topic `computed-channels` first, which documents the

© 2026 Altech Electronics Nordic AB. All rights reserved.

helpers for previous values, time steps and edges.

```
read_guide      {"topic": "computed-channels"}
evaluate_script {"inputs": [{"key": "Engine/EngineSpeed", "alias": "rpm"}],
                 "script": "function step(t, x) { return x.rpm / 60; }",
                 "maxPoints": 6}
  -> {"succeeded":true,"error":null,"failedSamples":0,"sampleCount":60,
      "min":7.27,"max":126.63,"mean":79.9,"points":[...], ...}
add_computed_channel {"name": "EngineRevsPerSecond", "unit": "1/s",
                 "inputs": [{"key": "Engine/EngineSpeed", "alias": "rpm"}],
                 "script": "function step(t, x) { return x.rpm / 60; }",
                 "plot": true}
  -> {"summary":"added computed channel Calc/EngineRevsPerSecond"}
```

Numbers above are shortened, and scripts are JavaScript. A script runs once per sample of its trigger input, inside CalTool.

10.7. Taking a recording

"Record speed and load for a few seconds." This needs the `workspace` tier, and `live-ecu` for going online.

```
go_online       {"device": "Engine"}
  -> {"online":true,"status":"Online via Simulation (loopback).",
      "detail":"EPK verified.\r\nWorking page.\r\n17 objects updated ...", ...}
set_raster      {"keys": ["EngineSpeed", "EngineLoad"], "raster": "default"}
start_recording {}
get_live_values {"keys": ["Engine/EngineSpeed"]}
stop_recording  {}
  -> {"summary":"recording stopped at 5.9 s with 2 channel(s)"}
save_recording  {"path": "C:\\Logs\\run01.mf4",
                 "comment": "Speed and load, bench run 1"}
go_offline      {"device": "Engine"}
```

A model has no clock of its own and cannot wait. Your host decides how time passes between `start_recording` and `stop_recording`, for example with a tool of its own that sleeps.

10.8. Changing a calibration value

"Raise the rev limit to 6400 rpm." The model reads first, then writes with a reason, and reports what happened.

```
get_calibration {"keys": ["Engine/RevLimit"]}
set_calibration {"changes": [{"key": "Engine/RevLimit", "value": 99999}],
```

© 2026 Altech Electronics Nordic AB. All rights reserved.

```
                "note": "..."}
  -> isError: "Engine/RevLimit: 99999 is outside the A2L limits 3000 .. 9000."
set_calibration {"changes": [{"key": "Engine/RevLimit", "value": 6400}],
                "note": "Raised on request, for the high-speed sweep"}
  -> {"changes":[{"key":"Engine/RevLimit","applied":true,
                  "old":"0","new":"6400","toEcu":true,"onEcu":true}],
      "summary":"1 value(s) changed, written to the online ECU - Raised on
       request, for the high-speed sweep; all 1 read back from the ECU and
verified"}
```

`toEcu` true means it went to a running ECU, after the engineer's confirmation. Offline it is false, and the `summary` says it landed in the desktop store. `onEcu` true means CalTool read the value back and the ECU holds it. For a device in a sync group, `mirroredTo` lists what changed on the partners.

Later in the session, the model can check the ECU and list what has been changed so far:

```
read_ecu_calibration {"keys": ["Engine/RevLimit"]}
  -> {"values":[{"key":"Engine/RevLimit","ecuValue":6400,"matchesDesktop":true}],
      "summary":"1 object(s) read from the ECU, 1 matching the desktop"}
get_calibration {"device": "Engine", "changedOnly": true}
  -> {"total":1,"offset":0,"values":[{"key":"Engine/RevLimit","value":6400,
      "origin":"desktop","changed":true,"referenceSource":"EcuRead",
      "referenceValue":0,"changedBy":"ai","changedAt":"2026-11-18T19:10:53"}],
      "summary":"1 of 1 changed value(s)"}
```

`read_ecu_calibration` reads from the ECU and changes nothing. `changedBy` says whether the model or the engineer made a change.

10.9. Watching and stimulating a CAN bus

"What is the engine sending? Then request 25 Nm every 10 ms." This needs the `can-trace` tier to listen and `can-transmit` to send. A transmit row built from a DBC message takes physical values and state names, and CalTool fills in the counter and the checksum on every send.

```
configure_can_interface {"interface": "Bus 1", "name": "Vehicle",
                 "channel": "Simulation", "dbc": "C:\\Dbc\\Demo.dbc"}
connect_can_interface {}
  -> {"interfaces":[{"name":"Vehicle","connected":true,
      "status":"Tracing Simulation (loopback).", ...}],
      "summary":"1 of 1 interface(s) connected"}
get_can_frames  {"name": "EngineStatus"}
  -> {"frames":[{"time":1.4595,"interface":"Vehicle","id":"0x110",
      "name":"EngineStatus","direction":"Rx","length":8,
      "data":"AE 69 AE 9C 01 00 00 00","count":74,"cycleMs":20.0,
      "signals":[{"name":"EngSpeed","value":6763.5,"unit":"rpm"},
```

© 2026 Altech Electronics Nordic AB. All rights reserved.

```
                {"name":"Gear","value":1,"state":"First"}, ...]}], ...}
configure_can_transmit {"message": "VCU_TorqueReq", "cycleMs": 10,
                "signals": {"TorqueEnable": "On", "TorqueReq": 25},
                "counters": [{"signal": "AliveCounter", "max": 14}],
                "checksums": [{"signal": "Crc", "algorithm": "e2e-p01",
                               "dataId": "0x200"}]}
  -> {"row":{"index":1,"id":"0x200","length":8,"valid":true, ...},
      "summary":"Transmit row 1 added"}
set_can_cyclic  {"index": 1, "running": true}
  -> {"summary":"Sending every 10 ms", ...}
set_can_cyclic  {"index": 1, "running": false}
```

For logic a transmit row cannot express, the model calls `read_guide` with topic `can-scripts`, then `save_can_script` and `run_can_script`. Identifiers and bytes are decimal unless written with `0x`, except a `data` string such as `"AA BB"`, which is hex.

10.10. Decoding CAN traffic in a log

"This bench log holds the inverter's CAN bus. Decode it with pwt_rear.dbc and check the alive counter." The model ties the DBC to the bus, then reads the decoded signals by their keys.

```
get_state       {}
  -> {..., "log":{"path":"C:\\Logs\\BenchRun.mf4","hasCanTraffic":true,
      "canBuses":[{"bus":1,"label":"CAN 1 · CAN1","frames":200}], ...}, ...}
configure_log_can_decoding {"mappings": [{"bus": 1,
                "dbc": "C:\\Dbc\\pwt_rear.dbc"}]}
  -> {"buses":[{"bus":1,"frames":200,"dbc":"C:\\Dbc\\pwt_rear.dbc",
      "decodedBy":"this bus's database","framesDecoded":200,"messages":1,
      "signals":3,"neverSeenCount":1,"neverSeen":["0x171 MotorLimits"],
      "exampleKeys":["pwt_rear/AliveCounter","pwt_rear/EngineSpeed",
                     "pwt_rear/MotorTemp"]}],
      "decodedSignals":3, ...}
get_signal_data {"keys": ["EngineSpeed"]}
  -> isError: "'EngineSpeed' names 2 different signals: Engine/EngineSpeed,
      pwt_rear/EngineSpeed. Use one of these keys."
get_signal_data {"keys": ["pwt_rear/EngineSpeed"]}
  -> {"dataSource":"log","signals":[{"key":"pwt_rear/EngineSpeed","unit":"rpm",
      "sampleCount":200,"samplePeriod":0.01,"min":1500,"max":4499.75, ...}], ...}
get_can_frames  {"source": "log", "ids": ["0x170"],
                "check": {"counter": "AliveCounter"}}
  -> {"counter":{"checked":199,"ok":197,"repeats":1,"jumps":1,
      "passRatePercent":98.995,"firstFailures":[{"time":1.2,"channel":1,
      "id":"0x170","previous":7,"expected":8,"got":9,"kind":"jump"}, ...]},
      "matchingFrames":200,
      "summary":"BenchRun.mf4: counter 197/199 ok (1 repeats, 1 jumps)"}
```

© 2026 Altech Electronics Nordic AB. All rights reserved.

The decoded signal `pwt_rear/EngineSpeed` and the A2L measurement `Engine/EngineSpeed` share a name. The bare name is therefore refused, and the model uses the key. `neverSeen` lists the messages of the DBC that the bus never carried. If most of a DBC is missing from the log, the DBC probably belongs to another bus. Decoding needs the `workspace` tier. Reading the log's frames is a read tool and leaves the CAN trace on screen alone.

10.11. A diagnostic session

"Which faults are stored, and what is the VIN?" This needs the `diagnostics` tier. An action's values come from the arguments, and a refused request comes back as a tool error naming the NRC.

```
diagnostics_connect {"ecuFile": "C:\\Ecu\\simulated-ecu.uds",
                     "channel": "Simulation"}
  -> {"session":"Extended session",
      "summary":"Connected to Simulated ECU on Simulation (loopback)."}
read_fault_codes {}
  -> {"faults":[{"code":"P0171","status":"confirmed, test failed",
      "active":true,"text":"System Too Lean Bank 1","textSource":"SAE"},
      {"code":"P1A23-1C","status":"pending","active":false,
      "text":"Fuel pump relay drive voltage out of range",
      "failureType":"Circuit voltage out of range",
      "textSource":"ECU file"}], ...}
read_data_identifier {"id": "VIN"}
  -> {"identifier":{"name":"VIN","id":"F190",
      "value":"ALTECHSIM00000001", ...}, ...}
run_diagnostic_action {"name": "Set fuel pump duty"}
  -> isError: "The action needs a value for: duty (1 byte(s))."
run_diagnostic_action {"name": "Set fuel pump duty",
                     "inputs": {"duty": 40}}
  -> {"running":false,"summary":"Set fuel pump duty done.", ...}
send_diagnostic_request {"request": "22 12 34"}
  -> isError: "Service 0x22 rejected: requestOutOfRange (NRC 0x31)"
diagnostics_disconnect {}
```

Without a file, the identifiers alone are enough for fault codes, any data identifier and memory. Pass `requestId` and `responseId` to `diagnostics_connect`.

10.12. Flashing an ECU

"Flash this build." This needs the `flash` tier. Check first that the image belongs to the ECU. `flash_ecu` runs over XCP or the VX tool, and `flash_ecu_uds` follows the `flash` sequence in the ECU file.

```
get_flash       {}
  -> {"targets":[{"name":"Generic XCP","route":"XCP over CAN", ...},
      {"name":"Vector VX flash tool", ...}], "running":false, ...}
```

© 2026 Altech Electronics Nordic AB. All rights reserved.

```
flash_ecu       {"target": "Generic XCP",
                 "firmware": "C:\\Builds\\EngineDemo.hex",
                 "channel": "Simulation", "waitSeconds": 60}
  -> {"running":false,
      "log":"... Verified 0x40000000..0x40004000 (XcpAdd14 0x000156A3).\r\n
             ... Resetting slave...\r\n... Flashing complete - slave reset.",
      "summary":"Flashing complete - slave reset."}
```

Without `waitSeconds` the call returns at once with `running` true, and `get_flash` gives the `progress` and verdict. For a device target, CalTool connects to the device after a successful flash, which is what proves the image took.

© 2026 Altech Electronics Nordic AB. All rights reserved.

11. Getting good results

Most of what decides the quality of an integration sits in your host, not in the protocol.

11.1. The system prompt

Put the server's `instructions` in the system prompt. That text comes with `server/discover` and explains keys and the order to work in. Then have your model read the guide once: either give it the text of `read_guide` with topic `tools`, or tell it to call the tool first. Add your own rules. These have proved sensible:

```
Report values and times exactly as the tools return them.
Never estimate a value you can read.
Check the units a tool reports before comparing or converting.
Read a curve or map before changing it, and change only the cells you were
asked to.
Give a note with every set_calibration, saying why, in one sentence. The
engineer reads it.
When a tool refuses something, tell the user the reason. Do not look for a way
around it.
Mark findings on the timeline, and open a plot of them, so that the engineer
can check them.
```

11.2. Offering fewer tools

Each tool's description costs context in every request, and a smaller choice helps a weaker model choose well. Read the tier from each tool's `_meta` and offer the ones the task needs, as the reference host does. CalTool checks permissions regardless, so this is about quality and defence in depth, not your only safeguard.

Suggested sets:

Task	Tools
Log analysis	The `read` and `analyse` tiers.
Log analysis that shows its findings	The above, plus `add_marker`, `open_plot`, `add_to_plot`, `set_plot_view`, `set_cursor`, `add_page`, `select_page`.
Building computed channels	`get_state`, `list_signals`, `describe_signal`, `read_guide`, `evaluate_script`, `add_computed_channel`, `update_computed_channel`, `list_computed_channels`, `get_signal_data`.
Explaining calibration	`get_state`, `list_signals`, `describe_signal`, `get_calibration`, `read_ecu_calibration`, `read_guide`.
Decoding CAN traffic in a log	`get_state`, `configure_log_can_decoding`, `list_signals`, `describe_signal`, `get_can_frames`, and the `read` and `analyse` tiers.
Watching a CAN bus	`get_can_trace`, `get_can_frames`, and the `can-trace` tier.

© 2026 Altech Electronics Nordic AB. All rights reserved.

Task	Tools
Stimulating a CAN bus	The above, plus the `can-transmit` tier and `read_guide`.
Reading faults and ECU data	`get_diagnostics`, `list_ecu_services`, `diagnostics_connect`, `diagnostics_disconnect`, `read_fault_codes`, `read_data_identifier`, `read_ecu_memory`.
Reprogramming an ECU	`get_state`, `get_flash`, and the `flash` tier. Offer it only for this task.

11.3. Keeping results small

The tools have limits of their own, but a careless call can still fill a small context. Have your host enforce a maximum result length and tell the model when it cut something, as the reference host does.

Tool	How to keep it small
`get_signal_data`	Use `statistics` first. `decimated` returns at most `maxPoints`, and keeps each section's highest and lowest sample, so peaks survive. Around 200 points is enough to see a shape.
`samples` mode	For short ranges only. It refuses, and says so, when the range holds more than `maxPoints`.
`list_signals`	Use `query` and `kinds`; `kinds: ["log"]` lists only what the log or recording holds. It returns 100 entries by default, and `total` says how many matched. Page with `offset`. `namesOnly` returns just the keys, up to 1000 by default.
`get_signal_grid`	Use it to compare signals over a window: one row per time step and one column per signal, instead of a `get_signal_data` call per signal. Rows times signals is at most 50 000; leave `step` out and set `maxRows` to fit.
`summarize_events`	Use it for a table of occurrences, such as every gear shift with its speeds and torque, instead of `find_events` followed by a `get_snapshot` per event. `limit` defaults to 50.
`get_calibration`	Name the `keys`, or use `changedOnly` to list only what was changed. Without them it lists up to 50 values of a device. A table over 32 elements in a listing (400 by key) comes as its shape, range and changed cells unless `full` is set.
`read_ecu_calibration`	Name few `keys`: each is read from the ECU over the bus. A large table comes as a summary with the cells that differ from the desktop unless `full` is set.
`find_events`	`limit` defaults to 200, and `truncated` says if there were more. `minDuration` removes short blips.
`get_snapshot`	Use it instead of `samples` to look around an event: it returns one value per signal and offset.
`scan_logs`	Its answer grows with files times signals. Narrow `pattern`, name few `keys`, and keep `limit` low in `events` mode.
`get_state`	It returns a summary with counts. Name the `sections` you need (`pages`, `plots`, `computed`, `rasters`); with `pages` its size grows with the workspace.
`list_computed_channels`	It returns full scripts.

© 2026 Altech Electronics Nordic AB. All rights reserved.

Tool	How to keep it small
`get_can_frames`	Filter by `ids` or `name`. `decode` adds every signal of each frame, and defaults to on in `latest` mode. It returns 100 frames by default.
`get_can_trace`	It returns each script's source and the latest output.
`list_ecu_services`	Use `query`. A CANdela file can hold hundreds of entries.

Old tool results in the conversation also cost context. In a long session, replace them with short summaries.

11.4. When the model struggles

Symptom	Likely cause and remedy
It answers without calling a tool, or describes the call as text	The server is not returning structured tool calls. Check the model's support and the server's chat template.
It sends arguments that are not valid JSON, or arrays as strings	A common failing of weaker models. Return the error to the model and let it retry, or use your LLM server's constrained tool-call output if it has one.
It repeats the same failing call	Limit the steps, as the reference host does, and check that it receives the tool error's text.
It invents signal names	Require `list_signals` before anything else, and return unknown-name errors to it. These errors say how to find the right name. Every tool refuses a key that does not exist.
It uses a bare name that belongs to several signals	The tool refuses and lists the keys. Return the error, and tell the model to use full keys from `list_signals`. A decoded CAN signal and an A2L measurement often share a name.
It reasons from breakpoints or values that are placeholders	Tell it to heed `breakpointsAreEstimated` and `value: null` with `note`.
Everything is slow	Calls run on CalTool's user interface thread, and an open dialog in CalTool can hold them up. Check the window.

© 2026 Altech Electronics Nordic AB. All rights reserved.

12. What the interface does not do

12.1. Not available to a model

These CalTool functions are used by the engineer in the window and have no tool:

- Adding or removing devices and setting up connections. To prepare a session, load a saved workspace with `open_workspace`.
- Writing memory images. Calibration files in `.cdfx` and `.dcm` format are available.
- Video and maps.

If your use needs one of these, contact Altech.

12.2. Limits of what is there

- It needs a running CalTool with its window, and an engineer to answer dialogs. It is not a batch or server interface.
- The server serves one session and does not separate clients. Several clients can connect, and they share the session and the permissions. The activity list does not say which client made a call.
- CalTool announces nothing by itself: a host learns of changes by asking. There are no subscriptions.
- `render_plot` works only for a plot on screen. `set_plot_view` with `show` brings it there; `get_signal_data` gives the numbers without a picture.
- The cursor cannot be moved while a recording captures.
- There is no limit on how far one change may move a value, beyond the A2L limits.
- Scripts run once per sample in CalTool, so a very long log takes a while to evaluate.
- Text-valued characteristics and objects with more than two axes cannot be written.
- From the CAN trace, `get_can_frames` sees the most recent frames only: 5000 in sequence, and the newest of each identifier. For a complete capture use `start_can_log`, then read the file with `source: "log"` or `file`.
- A CAN decoding set up with `configure_log_can_decoding` belongs to one log. Opening another log starts undecoded. A DBC message is matched by its exact identifier, so a J1939 message whose source address varies is not decoded. `scan_logs` does not decode CAN frames.
- `changedBy` and `changedAt` cover edits made since CalTool was started. They are not saved.
- A model cannot create a new ECU file or a seed and key DLL through the interface. It uses the ones on the PC, or sends its own requests with `send_diagnostic_request`.
- The CAN, diagnostics and flash tools have been exercised on the **Simulation** channel. Test them against your own hardware before relying on them.
- The server does not implement the 2026-07-28 revision's optional features, such as extensions

© 2026 Altech Electronics Nordic AB. All rights reserved.

and input-required results. It needs none of them.

For the wider product roadmap, or if your company runs another AI platform, contact Altech.

© 2026 Altech Electronics Nordic AB. All rights reserved.

Appendix A: A reference host in Python

This is a complete MCP host in one file: the client half of MCP, the conversion of tools to the format most LLM servers use, and the loop between them. It uses only the Python standard library, so it also runs on an isolated PC. It speaks to the model through the OpenAI-compatible `/chat/completions` API, which vLLM, llama.cpp's server, Ollama and LM Studio offer, and which many other servers do too. For a different API, only the function `chat` and the message format around it change.

It is meant as a starting point to read and adapt, not as a finished product. What you would add for production use:

- A record of each call for traceability (Section 8.6).
- Conversation memory, and summaries of old results (Section 11.3).
- A way for the host to let time pass during a recording (Section 10.7).
- Cancellation of a running call (Section 6.9).

Altech tested it against CalTool's server, and its loop against a scripted stand-in for an LLM server, not against a real model.

```
#!/usr/bin/env python3
"""Connect your own LLM to CalTool: a minimal MCP host in one file.

The model never talks to CalTool. It asks for tool calls, and this program
carries them out over MCP and returns the results. Any server with an
OpenAI-compatible /chat/completions endpoint that supports tool calling will
do, for example vLLM, llama.cpp's server, Ollama or LM Studio.

Python 3.9 or newer, standard library only, so it runs on an isolated PC too.

    set CALTOOL_MCP_TOKEN=<the token from CalTool's AI access dialog>
    python caltool_llm_host.py --llm-url http://localhost:8000/v1 ^
        --model my-model "What is loaded in CalTool right now?"

By default the model is offered only the tools that read and analyse. Add
--tiers read,analyse,workspace to let it change the workspace as well. CalTool
still applies its own permissions to every call, whatever this program offers.
"""

import argparse
import json
import os
import sys
import urllib.error
import urllib.request

PROTOCOL_VERSION = "2026-07-28"
```

© 2026 Altech Electronics Nordic AB. All rights reserved.

```
CLIENT_INFO = {"name": "caltool-llm-host", "version": "1.0"}
TIER_KEY = "io.altechelectronics/caltool-tier"
SYSTEM_PROMPT = (
    "You work in the user's running CalTool session through its tools. "
    "Report values and times exactly as the tools return them, and say so "
    "when a tool refuses something rather than working around it."
)


class McpError(Exception):
    """CalTool answered with a JSON-RPC error."""


class CalToolClient:
    """The client half of MCP: one JSON-RPC message per POST, no session."""

    def __init__(self, url, token):
        self.url = url
        self.token = token
        self.next_id = 0

    def request(self, method, params=None, name=None):
        self.next_id += 1
        params = dict(params or {})
        params["_meta"] = {
            "io.modelcontextprotocol/protocolVersion": PROTOCOL_VERSION,
            "io.modelcontextprotocol/clientInfo": CLIENT_INFO,
            "io.modelcontextprotocol/clientCapabilities": {},
        }
        message = {
            "jsonrpc": "2.0",
            "id": self.next_id,
            "method": method,
            "params": params,
        }
        headers = {
            "Content-Type": "application/json",
            "Accept": "application/json, text/event-stream",
            "Authorization": "Bearer " + self.token,
            "MCP-Protocol-Version": PROTOCOL_VERSION,
            "Mcp-Method": method,
        }
        if name is not None:
            headers["Mcp-Name"] = name
        data = json.dumps(message).encode("utf-8")
        request = urllib.request.Request(self.url, data, headers)
        try:
            # Long enough for an engineer to answer the confirmation dialog.
            with urllib.request.urlopen(request, timeout=600) as response:
```

© 2026 Altech Electronics Nordic AB. All rights reserved.

```
                answer = json.load(response)
        except urllib.error.HTTPError as error:
            body = error.read().decode("utf-8", "replace")
            if error.code == 401:
                raise McpError("CalTool refused the token. "
                               "Copy it again from the AI access dialog.")
            try:
                answer = json.loads(body)
            except ValueError:
                raise McpError(f"HTTP {error.code}: {body}")
        except urllib.error.URLError as error:
            raise McpError(f"Cannot reach CalTool at {self.url}. "
                           f"Is AI access switched on? ({error.reason})")
        if "error" in answer:
            problem = answer["error"]
            raise McpError(f"{problem['code']}: {problem['message']}")
        return answer["result"]

    def discover(self):
        return self.request("server/discover")

    def list_tools(self):
        return self.request("tools/list")["tools"]

    def call_tool(self, name, arguments):
        params = {"name": name, "arguments": arguments}
        return self.request("tools/call", params, name=name)


def to_llm_tool(tool):
    """An MCP tool in the OpenAI-compatible format. Both use JSON Schema."""
    return {
        "type": "function",
        "function": {
            "name": tool["name"],
            "description": tool["description"],
            "parameters": tool["inputSchema"],
        },
    }


def text_of(result, limit):
    """What the model reads of a tool result.

    A result holds the same JSON twice, as text in content and as
    structuredContent. Send only one, or a large answer costs its context
    twice. An isError result is ordinary text the model can act on.
    """
    items = result.get("content", [])
```

© 2026 Altech Electronics Nordic AB. All rights reserved.

```
    text = "\n".join(c["text"] for c in items if c.get("type") == "text")
    if len(text) > limit:
        text = (text[:limit] + f"\n[Cut at {limit} characters. Ask for less: "
                "a shorter time range, or fewer points.]")
    return text


def images_of(result):
    """render_plot answers with a PNG, for a vision model as a data URL."""
    return [
        f"data:{c.get('mimeType', 'image/png')}; base64,{c['data']}"
        for c in result.get("content", [])
        if c.get("type") == "image"
    ]


def chat(args, messages, tools):
    """One round with the LLM over the OpenAI-compatible chat API."""
    body = {
        "model": args.model,
        "messages": messages,
        "tools": tools,
        "temperature": 0,
    }
    headers = {"Content-Type": "application/json"}
    if os.environ.get("LLM_API_KEY"):
        headers["Authorization"] = "Bearer " + os.environ["LLM_API_KEY"]
    url = args.llm_url.rstrip("/") + "/chat/completions"
    data = json.dumps(body).encode("utf-8")
    request = urllib.request.Request(url, data, headers)
    with urllib.request.urlopen(request, timeout=args.llm_timeout) as response:
        return json.load(response)["choices"][0]["message"]


def run(args, caltool, question):
    """The host loop: ask the model, run its tool calls, until it answers."""
    server = caltool.discover()
    tiers = set(args.tiers.split(","))
    offered = [
        t for t in caltool.list_tools()
        if t.get("_meta", {}).get(TIER_KEY) in tiers
    ]
    allowed = {t["name"] for t in offered}
    tools = [to_llm_tool(t) for t in offered]
    print(f"[{len(offered)} tools offered: {args.tiers}]", file=sys.stderr)

    system = SYSTEM_PROMPT + "\n\n" + server.get("instructions", "")
    messages = [
        {"role": "system", "content": system},
```

© 2026 Altech Electronics Nordic AB. All rights reserved.

```
        {"role": "user", "content": question},
    ]
    for _ in range(args.max_steps):
        reply = chat(args, messages, tools)
        messages.append(reply)
        calls = reply.get("tool_calls") or []
        if not calls:
            return reply.get("content") or ""

        pictures = []
        for call in calls:
            name = call["function"]["name"]
            try:
                raw = call["function"].get("arguments") or "{}"
                arguments = json.loads(raw)
                if name not in allowed:
                    text = f"The tool '{name}' is not offered in this session."
                else:
                    print(f"[{name} {json.dumps(arguments)}]", file=sys.stderr)
                    result = caltool.call_tool(name, arguments)
                    text = text_of(result, args.max_result_chars)
                    pictures += images_of(result)
            except ValueError:
                text = "The arguments were not valid JSON. Send them again."
            except McpError as error:
                text = f"The call failed: {error}"
            messages.append(
                {"role": "tool", "tool_call_id": call["id"], "content": text})

        # A tool message holds text only, so pictures follow as a user message.
        if pictures and args.vision:
            content = [{"type": "text", "text": "The plot you asked for."}]
            content += [
                {"type": "image_url", "image_url": {"url": p}} for p in pictures
            ]
            messages.append({"role": "user", "content": content})
    return f"Stopped after {args.max_steps} steps without a final answer."


def main():
    parser = argparse.ArgumentParser(
        description="Connect an OpenAI-compatible LLM to CalTool over MCP.")
    parser.add_argument("question", help="What to ask, in plain language.")
    parser.add_argument("--mcp-url", default="http://127.0.0.1:38777/mcp",
                        help="CalTool's MCP endpoint.")
    parser.add_argument("--llm-url", required=True,
                        help="Base URL of the LLM server, ending in /v1.")
    parser.add_argument("--model", required=True,
                        help="Model name as the LLM server knows it.")
```

© 2026 Altech Electronics Nordic AB. All rights reserved.

```
    parser.add_argument("--tiers", default="read,analyse",
                        help="Tool tiers to offer: read, analyse, workspace, "
                             "calibration, live-ecu, can-trace, can-transmit, "
                             "diagnostics, flash.")
    parser.add_argument("--vision", action="store_true",
                        help="The model accepts images: send it render_plot's "
                             "PNG.")
    parser.add_argument("--max-steps", type=int, default=20,
                        help="Rounds with the model before giving up.")
    parser.add_argument("--max-result-chars", type=int, default=20000,
                        help="Cut a tool result at this length.")
    parser.add_argument("--llm-timeout", type=int, default=600,
                        help="Seconds to wait for one model reply.")
    args = parser.parse_args()

    # A Windows console's default encoding cannot print every unit or arrow.
    sys.stdout.reconfigure(encoding="utf-8", errors="replace")
    sys.stderr.reconfigure(encoding="utf-8", errors="replace")

    token = os.environ.get("CALTOOL_MCP_TOKEN")
    if not token:
        sys.exit("Set CALTOOL_MCP_TOKEN to the token shown in CalTool's "
                 "AI access dialog.")
    try:
        print(run(args, CalToolClient(args.mcp_url, token), args.question))
    except McpError as error:
        sys.exit(str(error))
    except urllib.error.URLError as error:
        sys.exit(f"Cannot reach the LLM at {args.llm_url}: {error}")


if __name__ == "__main__":
    main()
```

© 2026 Altech Electronics Nordic AB. All rights reserved.