



The Serial Console

A field guide to **serial-console-mcp** — how an agent opens a craft port, a CAT radio, a rotator or a microcontroller over a serial cable, says the right line ending, reads until the prompt instead of guessing, and hands the port back cleanly. Everything a terminal program knows, explained for the operator who would rather ask.

FIELD GUIDE · VERSION 0.1 · SEPTEMBER 2026

11

TOOLS

7

CONNECTION
SETTINGS

4

DEVICE
FAMILIES

4

WORKED
SESSIONS

Contents

BEFORE YOU START

A	The tools at a glance	03
B	Installing	04
C	Your first session — a console by conversation	05
D	How the system works	06

THE SETTINGS AND THE RULES

01	Connection settings — the Quick Connect dialog, in words	07
02	The console standard	08
03	The device playbook	09

AT THE BENCH

E	Worked sessions	10
F	When it doesn't answer	12

REFERENCE

G	Tool reference	13
H	About & provenance	14

The Tools at a Glance

One server, eleven tools, one open port at a time. The agent picks the tool; you say what you want in plain language.

TOOL	WHAT IT DOES	WHEN THE AGENT REACHES FOR IT
<code>list_serial_ports</code>	Every port the computer can see, with description and USB hardware id.	First, whenever a port name is unknown or a connect fails
<code>connect</code>	Open a port with baud, data bits, parity, stop bits and flow control. Starts the background reader.	“Connect to the console at 9600”
<code>reconnect_last</code>	Reopen the last port with the same settings, remembered across sessions.	Start of a new chat
<code>send_text</code>	Write an ASCII line with a chosen line ending. Writes only; never reads.	Any command, login, or bare return
<code>send_hex</code>	Write raw bytes given as hex. Writes only.	Icom CI-V and other binary protocols
<code>read_until_prompt</code>	Return buffered output up to a prompt, literal or regex; leaves the rest for the next read.	Every interactive CLI reply
<code>read_available</code>	Return whatever has arrived, as text and hex.	Binary replies, streaming or unsolicited output
<code>query_text</code>	Clear, send, then read until a prompt or until the line goes quiet.	One-shot CAT queries such as <code>ID;</code>
<code>clear_buffer</code>	Discard buffered receive data.	Right before a command whose reply must start clean
<code>status</code>	Port, settings, reader state, bytes buffered.	Pre-flight check; after anything odd
<code>disconnect</code>	Close the port and free it for other programs.	End of session, or before starting a terminal program

WHO READS THIS GUIDE

The agent reads the tool descriptions. This guide is for the human at the bench: what the agent will do on your behalf, which words make it do the right thing, and how to tell when a silent port is a wrong baud rate rather than a dead device.

Installing

The server runs as a host process next to Claude Desktop, not inside its sandbox. That is the whole reason it can see `/dev/cu.*`, `COM4` or `/dev/ttyUSB0`.

With the installer **recommended**

1 Run the installer.

Download `SerialConsoleMCP-Setup.exe` (Windows) or `SerialConsoleMCP-0.1.1.pkg` (Mac) from the GitHub Releases page and click through it. It copies one binary and registers it in `claude_desktop_config.json`, merging with any servers already there and backing up the old file. The installers are unsigned, so expect a Gatekeeper or SmartScreen warning.

2 Fully quit Claude Desktop, then reopen it.

Closing the window is not enough. Windows: right-click the tray icon, Quit. Mac: `⌘Q`. Servers are read at launch.

From PyPI **uv or pipx**

```
uvx serial-console-mcp configure --command uvx --arg serial-console-mcp
# or
pipx install serial-console-mcp
serial-console-mcp configure --command "$(which serial-console-mcp)"
```

Needs Python 3.10 or newer. `configure` writes the Claude Desktop entry, merging with servers already there and backing up the old file; `serial-console-mcp configure --remove` undoes it. Any MCP client can launch the same command over stdio. Developers: `git clone`, `uv sync`, `uv run pytest`.

Verify **thirty seconds**

1 Ask for the ports.

“What serial ports do you see?” A list with USB descriptions means the server is running and the OS sees your adapter. “No serial ports found” with the device plugged in almost always means a charge-only USB cable or a missing driver (FTDI, CP210x, CH340).

2 Open and close a port.

“Connect to the first one, show me the status, then disconnect.” Status should say the reader is running with zero bytes buffered.

ONE PROGRAM PER PORT

A serial port can be open in exactly one program. If a terminal, WSJT-X, a contest logger, the Arduino Serial Monitor or the device's own software holds it, the connect fails with “already in use”. Say *“disconnect”* before you hand the port to something else.

Your First Session — a Console by Conversation

You do not need to know a terminal program. If you can plug in a cable and describe your gear, you can operate it.

What you're actually installing

Claude is an assistant made by Anthropic, running in the Claude Desktop app. **serial-console-mcp** is best thought of as a rig-interface cable with a very patient operator on the other end: Claude types what you ask into the port, watches what comes back, and reports it. Nothing about your device changes; it sees an ordinary terminal.

Operating is a conversation

YOU TYPE	WHAT HAPPENS
What serial ports do you see?	Every port with its USB description, so you can pick the CP210x or FTDI that is your device.
Connect to /dev/cu.usbserial-10 at 9600.	Opens the port 9600 8N1 with no flow control and starts the background reader.
Connect to /dev/cu.BLTH at 38400 with XON/XOFF.	Same, with software flow control on. Every field of a terminal's Quick Connect dialog can be said this way (chapter 01).
Send a return and read until the login prompt.	Writes a bare CR, then reads until <code>login:</code> appears.
Log in as admin and run show version . Show me all of it.	Sends the username, waits for the password prompt, sends the password, waits for the shell prompt, runs the command, reads until the prompt again. Long output comes back whole.
Reconnect to the same port as last time.	The last port and settings are remembered across sessions.
Disconnect.	Closes the port so a terminal program or WSJT-X can have it.

YOU ARE STILL THE OPERATOR

These tools send exactly what you ask, to whatever is on the cable. Claude Desktop asks you to approve each tool call, so you see every command before it runs. Read it. A console session on a router or a rig can reconfigure, reboot, or transmit, and this server does not try to guess which commands are dangerous. If something looks wrong, decline it, and keep a real terminal handy for anything you would not want an assistant to type.

FIRST TIME? SOMETHING HARMLESS.

Start with a read-only command: `show version` on a router, `ID;` on a Kenwood, a frequency read on an Icom. Watch one full cycle, send, prompt, output, before asking for anything that changes state.

How the System Works

A serial console is not question-and-answer. It echoes what you type, prints on its own, and can dump pages. So the server never guesses when a reply is done; it keeps reading, and you tell it what the end looks like.

1 · OPEN

connect starts a reader thread

Every byte the device sends lands in a buffer from this moment on, whether or not anyone is reading.

2 · SEND

send_text · send_hex

Write only. Command + line ending on the wire. Nothing is read here.

3 · READ

read_until_prompt · read_available

Take from the buffer until the prompt appears, or until the line goes quiet. Leftover stays buffered.

4 · CLOSE

disconnect

Stops the reader, releases the port for other programs.

- **The reader owns the port.** One thread drains the OS buffer continuously. Echo, unsolicited syslog, and a three-page `show run` all arrive intact instead of being raced by ad-hoc polling. This is the model `minicom` and `expect` use.
- **Prompt, not timer.** `read_until_prompt` returns the instant the prompt shows up and hands back everything up to and including it. A fixed delay would truncate long output or waste seconds on short output; a prompt does neither.
- **Leftover is never lost.** Bytes after the matched prompt go back to the front of the buffer, so an unsolicited line that arrived mid-read is the first thing the next read sees.
- **Timeouts return what they have.** If the prompt never shows, you still get the partial output with a note to try again or change the prompt.
- **A dead port is announced.** Unplug the adapter and the reader stops; every tool then says so and asks you to reconnect, rather than timing out in silence.
- **Memory is bounded.** The buffer holds four megabytes. A device that streams unread for hours drops its oldest bytes and `status` reports how many.

THE HABIT THAT MATTERS

Tell the agent the prompt. “The prompt is `user@r1>` with a trailing space” turns every read into a clean, complete reply. Without it the agent falls back to reading until the line goes quiet, which works for one-line CAT answers and fails for anything paged.

Connection Settings

Everything in a terminal program's Quick Connect dialog, said in a sentence. The default is **9600-8-N-1** with no flow control, which is what most consoles and much radio gear expect; name only what differs.

DIALOG FIELD	PARAMETER	DEFAULT	VALUES	HOW TO SAY IT
Port	port	required	/dev/cu.usbserial-10 COM4 · /dev/ttyUSB0	“the CP210x one” works if you listed ports first
Baud rate	baud	9600	1200 ... 115200	“at 38400”
Data bits	bytesize	8	5 · 6 · 7 · 8	“7 data bits”
Parity	parity	N	N · E · 0	“even parity”
Stop bits	stopbits	1	1 · 1.5 · 2	“two stop bits”
RTS/CTS	rtscts	off	on · off	“with hardware flow control”
XON/XOFF	xonxoff	off	on · off	“with XON/XOFF”

Shorthand works too. “38400 8N1”, “9600 7E1” and “the usual Cisco console settings” all resolve to the right parameters. The agent repeats the settings back in the connect reply, so a misheard baud rate is visible before the first command goes out.

Flow control, the two lines that are usually wrong

- **Leave both off unless the manual says otherwise.** Consoles, CAT ports and microcontrollers overwhelmingly run without flow control.
- **RTS/CTS needs the wires.** Many console cables and three-wire adapters do not carry RTS and CTS. Turn it on with such a cable and every write waits for a CTS that never comes; the server gives up after two seconds and tells you exactly that.
- **XON/XOFF is for text only.** It treats the bytes 0x11 and 0x13 as flow-control characters and swallows them. Never enable it for Icom CI-V or any other binary protocol; a frequency containing those bytes would silently corrupt.

Remembered across sessions

Every successful connect writes port and settings, flow control included, to a small file in your user profile. “Reconnect to the same port as last time” reads it back. `status` shows the remembered port when nothing is open.

LINE ENDING IS A SEND SETTING, NOT A PORT SETTING

The dialog doesn't ask because a terminal sends whatever you type. Here it matters: `send_text` appends **CR** (most radios, rotators), **LF** (Unix-style consoles, Arduino), **CRLF**, or **nothing** (Kenwood-style CAT ends in `;`). Say “this device wants LF” once and the agent keeps it for the session.

The Console Standard

Six rules the tools are built around. Each one closes a hole that a naive send-and-wait would fall into.

1 Sending never reads.

`send_text` and `send_hex` put bytes on the wire and return. A reply is a separate, explicit read. This is what lets the agent log in (send, wait for `Password:`, send, wait for the shell) without guessing delays.

2 Name the prompt with its trailing space.

The match runs over everything received, including the echo of your own command. `"# "` is safe; a bare `"#"` matches the first `#` in a banner or in `show run | include #`. A regex such as `[\w.-]+[#>] ?$` handles hostnames that change after `configure`.

3 Clear before a command whose reply must be clean.

Syslog, interface flaps and a previous timed-out read leave bytes in the buffer. `send_text(..., clear_buffer_first=true)` or `query_text` discard them so the next read starts at your echo.

4 Text prompts for CLIs, idle reads for everything else.

Routers, switches and shells: `read_until_prompt`. Binary protocols and one-line CAT answers have no prompt: `read_available` or `query_text` with an empty prompt, which returns once the line has been quiet for a beat.

5 Baud wrong looks like garbage; line ending wrong looks like silence.

A stream of `␣` and box characters means the baud rate. An echo with no reply means the device is waiting for the other line ending. Send a bare return first to draw a fresh prompt; if that also produces nothing, check the cable.

6 Disconnect before handing the port to anything else.

The OS gives a port to one process. A forgotten open port is the most common reason WSJT-X or a terminal program “can't find” a device that is plainly plugged in.

WHERE THE RULES COME FROM

They are the reasons `minicom` and `expect` exist. The server implements them so the agent doesn't have to rediscover them at your bench.

The Device Playbook

Four families cover nearly everything on a serial cable. Settings below are the common factory defaults; the device's manual or menu wins when they differ.

FAMILY	TYPICAL SETTINGS	LINE ENDING	READ WITH	NOTES
Network craft / console ports Juniper • Cisco • Arista • switches , firewalls	9600 8N1	LF or CR	<code>read_until_prompt</code> "login: " . "> " . "# "	Send a bare return to draw a prompt. Turn off paging once (<code>set cli screen-length 0</code> , <code>terminal length 0</code>) or <code>--More--</code> becomes your prompt.
Text CAT radios Kenwood • Elecraft • Yaesu (FT-991, FTdx) • Flex	4800 – 115200 8N1 per the rig's menu	NONE	<code>query_text</code> (empty prompt, idle read)	Commands end in <code>;</code> with no CR: <code>ID;</code> <code>FA;</code> <code>MD;</code> . Replies are one line ending in <code>;</code> . Elecraft echoes nothing; Kenwood may need <code>AI0;</code> to stop auto-info.
Icom CI-V IC-7300 • 7610 • 705 • 9700	19200 8N1 (USB) 9600 via CI-V jack	n/a (binary)	<code>send_hex →</code> <code>read_available</code>	Frames are <code>FE FE <rig> E0 <cmd> ... FD</code> ; the reply shows as hex. Frequency is BCD, low byte first. Never enable XON/XOFF. If the rig echoes, the first frame back is your own.
Rotators, switches, MCUs Yaesu GS-232 • Green Heron • Arduino • ESP32	9600 or 115200 8N1	CR (rotators) LF (Arduino)	<code>read_available</code> or <code>read_until_prompt</code>	GS-232 <code>C</code> returns <code>+0xxx</code> azimuth. Opening a port resets many Arduinos via DTR; wait a second and expect a boot banner. Sketches that print continuously are a <code>read_available</code> job.

TRANSMIT-CAPABLE GEAR

A CAT command can key the radio (`TX;`) and a console command can reboot a router. The server doesn't filter; the approval prompt is your filter. On a rig, start with reads (`ID;` `FA;` `MD;`) and put the radio on a dummy load before any command that could transmit.

Worked Sessions

Four sessions, one per family, showing what you say, what the agent sends, and what comes back. Device replies are representative, not recordings.

1 · Juniper craft port: log in and read a version

9600 8N1 · LF · read_until_prompt

You: "Connect to the CP210x port at 9600, log in as admin, and show me the version."

```

TOOL list_serial_ports → /dev/cu.SLAB_USBtoUART – CP2102N USB to UART Bridge
TOOL connect(port="/dev/cu.SLAB_USBtoUART", baud=9600) → Connected ... (8N1, flow control none)
SEND send_text("", line_ending="LF", clear_buffer_first=true)
READ read_until_prompt("login: ") → Matched after 41 bytes:\n\nr1 (ttyd0)\n\nlogin:
SEND send_text("admin", line_ending="LF")
READ read_until_prompt("Password:") → admin\nPassword:

The agent asks you for the password rather than inventing one; you approve the send like any other.
SEND send_text("show version", line_ending="LF", clear_buffer_first=true)
READ read_until_prompt("> ") → show version\nHostname: r1\nModel: srx320\nJunos: 21.4R3-S4.15\n...\nadmin@r1>

```

WHY IT WORKED A bare LF drew a fresh prompt; each read named the exact next prompt; the buffer was cleared before the command so the reply began at the echo.

2 · Kenwood-style CAT: identify the rig and read the VFO

38400 8N1 · no line ending · query_text

You: "Connect to the radio at 38400 and tell me what it is and where it's tuned. Kenwood protocol, commands end in a semicolon."

```

TOOL connect(port="/dev/cu.usbserial-1420", baud=38400)
SEND query_text("ID;", line_ending="NONE")
READ Reply (6 bytes): 'ID021;' Hex: 49 44 30 32 31 3b
SEND query_text("FA;", line_ending="NONE")
READ Reply (14 bytes): 'FA00014074000;'

```

Agent: "ID 021 is a TS-590S. VFO A is on 14.074 MHz."

WHY IT WORKED No prompt exists, so query_text read until the line went quiet; NONE kept the CR off the wire, which Kenwood parsers reject.

3 · Icom CI-V: read the operating frequency in hex 19200 8N1 · send_hex → read_available

You: "Ask the IC-7300 for its frequency. CI-V address 94."

TOOL `connect(port="/dev/cu.usbmodem14201", baud=19200)`

SEND `send_hex("FE FE 94 E0 03 FD", clear_buffer_first=true)`

READ `read_available(1.0)` → Received 11 bytes Hex: fe fe e0 94 03 00 00 07 14 00 fd

Agent: "Command 03 reply, BCD low byte first: 00 00 07 14 00 → 14.070000 MHz."

WHY IT WORKED Binary reply, so the hex view is the answer; XON/XOFF stayed off, so the 0x14 and 0x07 bytes arrived untouched.

4 · Arduino sensor: watch a stream, then stop it 115200 8N1 · LF · read_available

You: "Connect to the Arduino at 115200 and show me a few seconds of what it's printing."

TOOL `connect(port="/dev/cu.usbmodem2101", baud=115200)`

Opening the port toggles DTR and resets the board; the first bytes are its boot banner.

READ `read_available(3.0)` → Received 118 bytes Text: 'BME280 ready\n23.4 C 1012.8 hPa 41%\n23.4 C 1012.8 hPa 41%\n23.5 C ...'

TOOL `status` → Reader running; 87 bytes buffered

TOOL `disconnect` → Disconnected from /dev/cu.usbmodem2101.

WHY IT WORKED A device that talks on its own has no prompt and no query; `read_available` with a longer timeout captures a window, and `disconnect` frees the port for the Arduino IDE.

When It Doesn't Answer

Almost every silent port is one of six things. The server's own messages point at most of them; this table is the rest.

SYMPTOM	MOST LIKELY CAUSE	WHAT TO SAY
No serial ports found	Charge-only USB cable, missing driver, device off. Windows: nothing under Ports (COM & LPT).	Swap the cable, then “list the ports again”
Could not open ... already in use	Another program holds the port: terminal, WSJT-X, logger, IDE serial monitor.	Close it, then “try connecting again”
Port name wasn't found	Names change when you replug; the remembered port is stale.	“List the ports and connect to the right one”
Garbage characters, , box drawing	Wrong baud rate. Occasionally wrong data bits or parity.	“Reconnect at 115200” (or the rig's menu value)
Echo comes back, no reply	Wrong line ending for this device, or a CLI that needs a return to wake.	“Send a return first” · “use LF instead of CR”
Prompt not seen, partial output returned	The prompt string doesn't match: missing trailing space, hostname changed, or <code>--More--</code> paging.	“The prompt is admin@r1> with a space” · “turn off paging”
Write failed: Write timeout	RTS/CTS is on and the cable doesn't carry CTS.	“Reconnect without hardware flow control”
Background reader has stopped	Adapter unplugged or port taken by another program mid-session.	“Disconnect and reconnect”
Claude says it has no serial tools	Claude Desktop wasn't fully quit after install, or the config entry is missing.	Quit with <code>⌘Q</code> / tray Quit, reopen, new chat

TWO QUESTIONS THAT SORT MOST OF IT

“What's the status?” tells you whether a port is open, at what settings, whether the reader is alive, and how much is buffered. “Read whatever's waiting” shows raw bytes as text and hex, which is how you tell a baud-rate problem (garbage) from a line-ending problem (silence).

Tool Reference

Parameters as the agent sees them. Defaults in brackets.

TOOL	PARAMETERS	RETURNS
<code>list_serial_ports</code>	none	One line per port: device, description, hardware id
<code>connect</code>	<code>port · baud [9600] · bytesize [8] · parity N E O [N] · stopbits 1 1.5 2 [1] · rtscts [false] · xonxoff [false]</code>	Settings echoed back, or the reason it failed with a hint
<code>reconnect_last</code>	none	Same as connect, using the remembered settings
<code>send_text</code>	<code>data · line_ending CR CRLF LF NONE [CR] · clear_buffer_first [false]</code>	Byte count sent
<code>send_hex</code>	<code>hex_bytes · clear_buffer_first [false]</code>	Bytes sent, as hex
<code>read_until_prompt</code>	<code>prompt [#] · timeout [10 s] · regex [false]</code>	Output through the prompt; on timeout, whatever arrived
<code>read_available</code>	<code>read_timeout [1 s]</code>	Bytes as text and hex, after the line goes quiet
<code>query_text</code>	<code>data · line_ending [CR] · prompt [none] · read_timeout [5 s]</code>	The reply; with a prompt, same as read_until_prompt
<code>clear_buffer</code>	none	Number of bytes discarded
<code>status</code>	none	Port, settings, flow control, reader state, bytes buffered and dropped
<code>disconnect</code>	none	Confirmation; always releases the port

Behaviour worth knowing

- **Non-ASCII text** in `send_text` is sent as `?`; use `send_hex` for raw bytes.
- **Display decoding** is UTF-8 with replacement characters; prompt matching is byte-exact, so binary noise before a prompt does not shift the match.
- **Write timeout** is two seconds; **reader poll** is a tenth of a second; **idle settle** for prompt-less reads is 150 ms.
- **Remembered connection** lives at `~/Library/Application Support/serial-console-mcp/last_connection.json` (Mac), `%APPDATA%\serial-console-mcp\` (Windows), `~/.config/serial-console-mcp/` (Linux).

About & Provenance

This is an experimental 0.1.1. The guide says what the server does and what has been verified; it does not claim more.

serial-console-mcp began as a generic console tool for network craft ports and radio gear and was rewritten for this release around the background-reader model in chapter D. The 0.1.1 code is verified by a 37-case test suite driving a simulated serial port, by a live MCP transport handshake, and by continuous integration on Linux, macOS and Windows. The worked sessions in chapter E are representative walk-throughs of the tool flow, not recordings from hardware; a live-port pass is the next milestone, and this guide will say so when it has happened.

RESOURCE	WHERE
serial-console-mcp	github.com/sbrunner-atx/serial-console-mcp
Installers	GitHub Releases · SerialConsoleMCP-Setup.exe · SerialConsoleMCP-0.1.1.pkg
PyPI	pypi.org/project/serial-console-mcp · uvx serial-console-mcp
Server and tests	src/serial_console_mcp/server.py · tests/test_serial_console.py
Live-port harness	_hosttest/run_live.py <steps.json>
This guide's sources	docs/brand/ (HTML + CSS, WeasyPrint)
Sibling servers	fldigi-mcp · wsjtx-mcp · n3fjp-mcp · mcp-host-bridge

Corrections and additions are welcome — open an issue on the repository.

73

This is private, personal work by Stefan Brunner, AE5VG — built for amateur radio operators and anyone else with a console cable. Not affiliated with any equipment vendor or any employer. MIT licensed. GL es 73 de AE5VG sk.