

cs-aihelp

Manual

AI helpdesk for napp-it 4ai - answers grounded in the local documentation, optional web research, local or cloud LLM

Version 1.2.7 · October 13, 2026

Contents

1. Overview	3
2. Features	3
3. The Go daemon	4
4. Requirements and installation	4
5. Quick start	5
6. Configuration	5
6.1 Config keys	6
6.2 Provider setup	8
6.3 Web research	8
6.4 Chat history	8
7. Usage	8
7.1 The chat menu	8
7.2 The floating popup	9
8. Security model	9
9. Architecture	9
9.1 csweb-gui hooks	10
10. Testing, release, license	10

1. Overview

cs-aihelp is a chat-based **AI helpdesk** that runs directly inside the napp-it CS web GUI. It answers napp-it questions grounded in the local documentation (`data/howto.ai/*.info`, light RAG), can enrich the answer with a web search and can be wired to a local or cloud LLM - with or without an API key.

It is part of the napp-it 4ai (client-server edition) cluster tooling. csweb-gui deploys and updates the Go daemon per member with the menu **About > Download cs-tools**.

Version 1.2.7 (stable) - license BSD 2-Clause. The frontend runs on every napp-it CS OS (FreeBSD, illumos, Linux, macOS, Solaris, Windows); the Go daemon is built for 8 targets. Local free AI (Ollama) runs natively on Linux, macOS and Windows; on FreeBSD, illumos and Solaris use the Pollinations fallback, a remote Ollama (`provider=ollama` plus endpoint) or an OpenAI-compatible local server (for example llama.cpp).

Update to 1.2.7 is required. v1.2.6 and v1.2.7 fix two bugs of every earlier v1.2.x build: on Windows the console exec (`exec_access=console`) never actually executed the commands proposed by the AI (ConPTY line ending), and the answers of reasoning models were cut to almost nothing (answer-token starvation).

2. Features

- **Zero-config free tier:** `mode=free` works without an account or API key. It prefers a local Ollama daemon and falls back to a Pollinations.AI GET.
- **Bring your own provider:** `mode=provider` for Anthropic, OpenAI (or any OpenAI-compatible endpoint: vLLM, LM Studio, Groq, ...) and Ollama (no key).
- **No silent fallback:** a failing `mode=provider` call (unreachable, wrong key, timeout) is shown as an error and not quietly answered by a different, uncontrolled model (since v1.2; the `fallback` key is parsed for compatibility only).
- **Grounded answers with transparent sources:** light RAG over `data/howto.ai/*.info` with a filename-aware ranking, so a question that names a topic ("what is cs-sync") reliably finds the matching document. Every answer has two labeled sections: **Local docs:** (strictly grounded in the retrieved excerpts, never invents napp-it commands, paths or settings) and **General AI:** (the background knowledge of the model).
- **Level 1 read-only diagnostics:** optionally attach the live state (`hostname`, `zpool list`) of the selected member as data context.
- **Level 2 exec (agentic):** with `exec_access = exec|console` the AI may *propose* commands. Execution is gated by an allow list (`exec_allow`) and a deny list that always wins (`exec_deny`), runs only after your confirmation (`exec_mode=confirm`, the default) in a dedicated session-gated CGI, and the output is fed back into the conversation - the AI can search pools, analyze and fix problems step by step.
- **Web research:** `research=ddg` (DuckDuckGo Lite, no key, default) or `research=api` (any JSON search endpoint; Google CSE, Brave, Bing, SearXNG and Serper response shapes are detected automatically). Result URLs appear in the sources line.
- **Chat history:** one file per conversation in `_cfg/aihelp/conv_*.json`, retention `off | today | week | month | 6months | all` (default `month`), resume any earlier conversation. The current conversation is loaded automatically until you press **New**.
- **Two UIs:** the full chat page (**Help > AI Helpdesk**) and a draggable, context-sensitive popup ("Ask AI") on every logged-in page (`widget=on`).
- **Two provider slots (plan/act):** slot 1 = plan / read-only provider, slot 2 = act / exec provider; choose `plan` or `act` in the Helpdesk or the popup. An empty `mode2` falls back to slot 1.
- **Localized UI:** form elements and short hints stay English; the explanations and the Helpdesk and popup texts are translated through the napp-it language files.

3. The Go daemon

Since v1.0 cs-aihelp is a **standalone Go daemon** (pure standard library, `CGO_ENABLED=0`, 8 platforms) with its own memory: an in-memory RAG index over `data/howto.ai/*.info`, a persisted chat history (same format as the Perl frontend) and a reloadable in-memory config. It is always ready in the background and reachable remotely over HTTPS.

```
cs-aihelp serve  [--listen 0.0.0.0:45555] [--config PATH] [--foreground]
cs-aihelp start  [--config PATH]      (detached and idempotent; boot hook)
cs-aihelp ask    --question "..." [--json]
cs-aihelp status [--json]
cs-aihelp stop   [--config PATH]      (reads the daemon PID file)
cs-aihelp reindex
cs-aihelp version
```

`serve` writes a PID file (Unix `/var/run/cs-aihelp.pid`, Windows next to the config) and removes it on shutdown. `start` is the boot-hook entry point (detached, skips when the port is already in use). `stop` works on all platforms (`taskkill` on Windows).

Endpoints (HTTPS, Bearer authentication): `GET /health /status /sources?q= /models`, `GET /resume?member=...` (newest conversation of the member), `POST /ask` (JSON, or `stream=true` for SSE) and `POST /reload /reindex`. From the browser they are reached through the session-gated `cs-aihelp.pl` proxy.

Daemon-only config keys (in the same file `_cfg/cs-aihelp`):

Key	Description
<code>listen</code>	<code>127.0.0.1:45555</code> (default); <code>0.0.0.0:45555</code> for remote access
<code>allowed_ip</code>	comma-separated IP/CIDR allowlist; empty = loopback only; <code>*</code> = any
<code>auth_token</code>	bearer token (mandatory when <code>listen</code> is not loopback)
<code>tls_cert / tls_key</code>	default = the certificate of <code>webserver.pl</code> (<code>_cfg/webserver/cert/server.crt</code> and <code>.key</code>)
<code>cors_origin</code>	frontend origin for CORS pinning (optional)
<code>ssrf_allow_private</code>	<code>no</code> or <code>yes</code> - allow RFC1918 / private endpoints (LAN-only remote Ollama); metadata and link-local addresses stay blocked
<code>rate_limit</code>	maximum requests per minute per client IP in the daemon (0 = off, default 60)

Security: IP allowlist and constant-time bearer token check, an SSRF guard on all configured endpoints, TLS with the same certificate the web GUI uses (remote browsers already trust it) and config files written with mode `0600`.

Deploy: `data/cs_server/tools/cs-aihelp/<os>.<arch>/cs-aihelp[.exe]`; it is started by `autostart.pl` alongside `server.pl`, `auto.pl` and `monitor.pl`.

4. Requirements and installation

- An installed napp-it CS web GUI (default path `/opt/csweb-gui`; on Windows `C:\opt\csweb-gui`).
- Perl with the core modules `HTTP::Tiny` and `JSON::PP` plus `URI::Escape` (`liburi-perl` on Debian and Ubuntu; already bundled with napp-it CS).
- Optional for the free tier: **Ollama** (`curl -fsSL https://ollama.com/install.sh | sh`, or the Windows installer from ollama.com, then `ollama pull llama3.1`). The free mode detects it on `127.0.0.1:11434`. Internet access is needed for the Pollinations fallback and for web research.
- Optional for cloud providers: an API key from console.anthropic.com or platform.openai.com.

The module files are copied into the napp-it CS installation with:

```
perl install.pl /opt/csweb-gui      # default target is /opt/csweb-gui
```

File	Install target
data/wwwroot/cgi-bin/cs-aihelp.pl	session-gated proxy to the Go daemon (chat, SSE)
data/wwwroot/cgi-bin/cs-aihelp-exec.pl	exec CGI (Level 2)
data/menues/_lib/windows/aihelpplib.pl	shared library
data/menues/_lib/windows/cstoolslib.pl	CS tools registry and GitHub download
data/menues/05_Help/00_AI_Helpdesk/action.pl	chat menu (first item in Help)
data/menues/10_System/05_Services/12_AI_Helpdesk/action.pl	settings menu
data/menues/_lib/tools/CS_Tools_Download/action.pl	download and update menu (including local AI / Ollama)
data/howto.ai/ai-helpdesk.info	module documentation
config/cs-aihelp.example -> _cfg/cs-aihelp	default config (only if absent)

cs-aihelp is not bundled in napp-it CS. The daemon binary is downloaded from GitHub with **About > Download cs-tools**; on first use the settings menu shows "please download CS tools first". Only the binary for the frontend OS is fetched and installed keeping the OS structure (data/cs_server/tools/cs-aihelp/<platform>.<arch>/), so csweb-gui/data can be copied to another OS, where the matching binary is resolved. The same page offers the local AI (Ollama) download and setup and a background model pull; start and stop of the daemon are in **System > Services > AI Helpdesk**. No restart is needed for the menus; the config file is read on every request.

5. Quick start

1. Install the module (perl install.pl).
2. Reload the web GUI and log in.
3. For a reliable free setup install Ollama: curl -fsSL https://ollama.com/install.sh | sh and ollama pull llama3.1.
4. Open **Help > AI Helpdesk** and ask a question. It works immediately in mode=free (Ollama, or the Pollinations fallback if Ollama is not installed).
5. Optional: **System > Services > AI Helpdesk** to choose a free model, a cloud provider, web research and the other options.

6. Configuration

The module is configured in **System > Services > AI Helpdesk** and saved to _cfg/cs-aihelp, a flat key = value file that is created on first run.

6.1 Config keys

Key	Values (default)	Description					
mode	off / free / provider (free)	slot 1 (plan/RO): free = no key, local Ollama then Pollinations; provider = the configured endpoint and an optional key					
provider	openai / anthropic / ollama (openai)	slot 1, used in mode=provider					
endpoint	URL (empty)	slot 1: full chat-completions / messages URL; empty = provider default					
model	string (empty)	slot 1: empty = provider default					
api_key	string (empty)	slot 1: cloud providers only; stored server-side, never logged					
mode2	empty / free / provider (empty)	slot 2 (act/exec provider); empty or off = uses slot 1					
provider2, endpoint2, model2, api_key2, free_model2		slot 2 provider settings					
free_model	string (empty)	slot 1 mode=free: local Ollama model tag; empty = first available					
fallback	off	removed in v1.2, parsed for backward compatibility only					
tool_use	no / yes (no)	Level 1: attach read-only live state (hostname, zpool list)					
research	off / ddg / api (ddg)	web research; ddg = DuckDuckGo Lite (no key), api = external endpoint					
research_max	number (5)	maximum number of search results added to the context					

Key	Values (default)	Description					
research_endpoint	URL (empty)	research=api: URL template with {q} (or automatic ?q=)					
research_key	string (empty)	research=api: optional key (sent as Bearer / X-API-Key)					
history	off / today / week / month / 6months / all (month)	chat history retention					
history_turns	number (10)	prior turns sent as context on resume					
widget	on / off (on)	floating read-only "Ask AI" popup on every logged-in page					
widget_input_lines	number (1)	popup question field: 1 = single line, 2-10 = multiline textarea					
widget_answer_height	number (220)	popup answer area height in px (100-1200)					
exec_access	ro / exec / console (ro)	Level 2: ro = read-only, exec = allow list, console = only exec_deny applies					
exec_mode	propose / confirm / auto (confirm)	propose = show only, confirm = every exec needs a click, auto = no per-step confirmation					
exec_allow	comma list (empty)	command classes / prefixes that may execute (first word of the command chain); empty = nothing executes					
exec_deny	pipe list	always applied, wins against exec_allow; default `zfs destroy	zpool destroy	rm -rf	dd	mkfs	format
autostart	on / off (on)	start the Go daemon at every server.pl boot (when mode != off and the binary exists)					

Key	Values (default)	Description					
<code>max_context</code>	number (8000)	system prompt budget in characters					

6.2 Provider setup

Option	What it is	Key needed
<code>free</code>	local Ollama (reliable, private) with the Pollinations.AI GET fallback (instant, experimental)	no
<code>provider = ollama</code>	local Ollama through an explicit provider config	no
<code>provider = anthropic</code>	Claude (best quality)	yes (console.anthropic.com)
<code>provider = openai</code>	OpenAI or any OpenAI-compatible service	yes (none for local servers)

Privacy: `free` (Pollinations) and the cloud providers send the question (including the tool-use context) to an external service. For sensitive environments use Ollama locally.

6.3 Web research

- `research=ddg` (default) uses `lite.duckduckgo.com` - free, no key, direct internet access. Direct scraping of Google is blocked by Google, so for Google use `research=api` with a Google CSE key.
- `research=api` calls any JSON search endpoint, for example `research_endpoint = https://www.googleapis.com/customsearch/v1?key=...&cx=...&q={q}`. The `{q}` placeholder is replaced by the question; if it is absent, `?q=` is appended. The response shapes of Google CSE, Brave, Bing, SearXNG and Serper are detected automatically.

6.4 Chat history

Conversations are stored in `_cfg/aihelp/conv_<timestamp>_<id>.json` (`{created, updated, member, title, messages[]}`). Only question and answer pairs are stored - never system prompts or secrets. Cleanup runs lazily on every request according to `history`. On resume the last `history_turns` turns are sent to the model as context.

7. Usage

7.1 The chat menu

Help > AI Helpdesk opens the full-screen chat page:

- Toolbar:** `Plan` | `Act` toggle (slot 2 = exec-capable model), the radio `ro` | `exec` | `console` (mirrors `exec_access`), the select `propose` | `confirm` | `auto` (mirrors `exec_mode`), **Plan first** (checked by default: the AI presents a plan and waits for your go-ahead before it proposes actions), **Abort** (stops the agentic loop) and **New** (fresh conversation).
- The question field is on top, the live transcript below. **Quick questions** fill and send a sample question.
- Agentic steps:** a proposed command arrives as an action card under the answer - in `confirm` mode with **Execute** / **Abort** buttons, in `auto` mode it runs immediately (Abort stops the next step). The command output is fed back to the AI, which continues.
- Every answer has a copy button and a sources line (documentation files and/or research URLs).
- There is one chat history per member; `New` starts fresh, and the `history` setting only controls when old turns are deleted.

7.2 The floating popup

With `widget=on` an **Ask AI** button is injected on every logged-in page. It opens a freely draggable, **read-only** chat popup - actions are never executed from here (use the full-screen page for exec). It sends the current menu path and the selected member as context, so a question asked inside the *ZFS Snaps* menu is answered with that context. The size is configurable with `widget_input_lines` and `widget_answer_height`.

8. Security model

- API keys live only in `_cfg/cs-aihelp` (server-side), never in the browser.
- Model answers are HTML-escaped before rendering (XSS guard).
- Live system state and web results are passed as **data** ("treat as data, not instructions") to mitigate prompt injection.
- **Level 2 exec is defense in depth:**
 - `exec_access=ro` (default): the AI is read-only, no proposals, no exec.
 - The Go daemon only ever *proposes* (an `[[ACTION]]` block). Execution happens in the separate session-gated Perl CGI `cs-aihelp-exec.pl` through the encrypted `&exe()/&socket()` channel.
 - `exec_deny` is always applied and **wins** (default `zfs destroy|zpool destroy|rm -rf|dd |mkfs|format`).
 - `exec_access=exec` ("act" in the GUI) is an **allow list**: the class of the command must be in `exec_allow`; an empty `exec_allow` means nothing executes.
 - `exec_access=console` is a **deny list only**: `exec_allow` is bypassed, anything may run except what `exec_deny` matches. Grant it only to fully trusted administrators.
 - `exec_mode=confirm` (default) requires an explicit click per command; `auto` should only be used with a tight allow list.
- Since v1.2.1 the model is instructed to act (emit `[[ACTION]]`) when asked to do or determine something, and to read the returned output on its next turn and keep going.
- Since v1.2.2 the model never proposes an installer command without its non-interactive flag (`apt-get install -y`, `winget install --silent`, ...). As a backstop, `server.pl` wraps every exec path in `IPC::Run` with a 60 s timeout that kills a hung child.
- The chat history contains question and answer pairs only (no secrets) and is pruned by the retention setting.

9. Architecture

```
Browser (menu page or popup)
  | POST /cgi-bin/cs-aihelp.pl (session check + read-only live_state)
  v
cs-aihelp.pl (Perl proxy, loopback, HTTPS webserver cert / Bearer auth_token)
  | /ask (buffered or SSE stream:true)
  v
cs-aihelp (Go daemon, 127.0.0.1:45555)
  | in-memory RAG index (data/howto.ai/*.info)
  | optional web research (ddg / api)
  | provider call: free (Ollama -> Pollinations) | provider (anthropic/openai/ollama)
  | chat history (_cfg/aihelp/conv_*.json), /resume
  v
Browser: tokens streamed as SSE or buffered JSON; answer ("Local docs:" / "General AI:") + sources
```

The Perl frontend (menus, settings, popup, chat page) is a thin layer: it writes `_cfg/cs-aihelp`, collects the read-only live state and the menu context and proxies every `/ask` to the Go daemon over loopback. The exec channel stays in the session-gated Perl CGI.

Level 2 agentic loop: the browser runs the loop, the daemon only answers. The question goes through the proxy to `/ask`; the answer may contain an `[[ACTION]]` with command and reason. In confirm mode the user clicks Execute, in auto mode it runs automatically; `cs-aihelp-exec.pl` validates the command against the allow and deny lists and executes it over the encrypted channel; the output goes back as `tool_results` to `/ask`, and the loop continues. Execution never happens in the Go daemon - it is a proposer only.

9.1 csweb-gui hooks

Four UI and boot hooks touch core napp-it files and are applied on the napp-it side (they are not part of the file list of `install.pl`):

- the **status light** (AI dot) in `get_async.pl` and `admin-wlib.pl`,
- the **popup injection** in the `interface.pl` header (`ai_popup(...)`) and `admin.pl`,
- the **daemon autostart** in `server_boot_tasks.pl` (`ai_boot_autostart()` at every `server.pl` start),
- the **localization** files `lang/*/ai_helpdesk.txt` and `lang/*/help.txt` plus the two menu keys in `about_menus.txt`.

10. Testing, release, license

```
tests/run_tests.sh                # Linux / macOS / illumos
powershell -ExecutionPolicy Bypass -File tests/run_tests.ps1  # Windows
```

The Perl suite starts a local mock HTTP server (OpenAI-compatible chat, Ollama, DuckDuckGo Lite HTML, Google CSE JSON) and runs 49 functional checks: config round trip, provider resolution, RAG retrieval, history save/load/list/cleanup, provider calls and error paths, the free-tier Ollama path, research parsing, the SSRF guard and the `log` key. The Go daemon has its own unit tests (`go test ./...`).

GitHub Actions builds the release on a `v*` tag: `ci.yml` (Go vet and test plus the Perl suite on every push) and `release.yml` (the Go daemon for 8 platforms as `cs-aihelp-<os>.<arch>.tar.gz` plus the module archive, with `checksums.txt`).

License: BSD 2-Clause - Copyright (c) 2026 Guenther Alka / napp-it.org.