IndyKnow — Best Practices, Tips & Tricks

A short field guide for using IndyKnow from Claude Desktop. If you're new to Claude itself, the first two sections are the must-reads.

---

1. Quickstart

You've installed `indyknow.mcpb` and pasted your **Environment Id** into the extension settings. Here's what happens next:

1. Open a new chat in Claude Desktop.
2. Ask your first IndyKnow-flavored question (see §2). Claude will reach for an IndyKnow tool.
3. **The very first tool call triggers Keycloak sign-in.** Your browser opens to a sign-in page with a short code already filled in. Approve it.
4. The browser will say "Sign-in successful." Return to Claude — your answer is on its way.
5. Subsequent calls reuse a cached token silently. You won't sign in again until that token expires (typically several days, controlled by your realm's refresh-token policy).

How do I know Claude is actually using IndyKnow? Claude announces tool calls inline: "*I'll use the `whoami` tool to confirm your identity…*" If Claude answers without mentioning a tool, it's guessing from training data, not your codebase.

Picking up new capabilities. IndyKnow's tools live on the server, so new features (like bug reporting) arrive without a reinstall or a new `.mcpb`. When an IndyKnow update is announced, **fully quit and reopen Claude Desktop** to pick them up — the extension reads the available tools when it starts, so opening a new chat alone won't surface them. If something's still missing, toggle the IndyKnow extension off and back on in **Settings** → **Extensions**.

---

2. Talking to Claude so it actually uses IndyKnow

First — know which role you have

Two role tiers control the **cadence** of IndyKnow's answers — not how much it can see. Both tiers investigate with the same full access to source, schema, and live data:

- **`indyknow:viewer`** — the support tier. Answers arrive support-mentor style: plain English first, jargon defined the first time it appears, customer impact up front. When source, SQL, or line numbers genuinely help, IndyKnow shows them and walks through what they mean.
- **`indyknow:developer`** — the developer tier. Same access, terser delivery: source, DDL, file paths, and `file.pas:line` references come back directly, in the form a developer wants them.

Ask Claude to run `whoami` if you're not sure which you have — the **Roles** line tells you. Either tier gets the exact, correct answer; they differ only in how it's explained.

Tips that apply to everyone

The single biggest factor in answer quality is your prompt.

- **Mention "IndyKnow" or "the codebase" in your prompt.** Without that hint, Claude often skips the tool and improvises. "*Use IndyKnow to find how the splash screen checks permissions*" works better than "*how does the splash screen check permissions?*" on its own.
- **Ask for whatever you need — behavior *or* source.** Every tier can get source, file paths, DDL, and live query results. Viewers get them explained in plain words; if you'd rather see the raw code or a terser answer, just

say so.

- **Be specific.** Name a module, form, table, or stored procedure when you can. "How does the work-order save logic work?" beats "how does saving work?"
- **One question at a time.** Multi-turn chats keep context, so follow-ups like *"what about when the network drops mid-save?"* work great. Start a fresh chat when you change topics — long histories cost more per turn.

Examples that work for everyone

- ✓ "Use IndyKnow to walk me through what happens when a work order is saved."
- ✓ "In the codebase, what columns are on `WO_Charges` and what's the primary key?"
- ✓ "How does the splash screen check user permissions on startup?"
- ✓ (follow-up) "What happens if there's a network failure mid-save?"

Asking for source directly — any tier

These all work now, whatever your role — a viewer just gets the answer walked through in plain words:

- ✓ "Show me how the splash screen checks permissions, and explain what the code is doing."
- ✓ "What columns and keys does `WO_Charges` have?"
- ✓ "Walk me through the save logic and point me at where it starts."

With `indyknow:developer` you get the same facts back more tersely — raw source, full DDL, and `file.pas:line` references in developer shorthand.

Searching deeply — and asking for more

When you ask a "find **everything**" question — *"every place that calls the licensing check"*, *"all the rules that touch this table"* — IndyKnow returns matches a page at a time, and Claude keeps pulling the next page on its own until it has enough to answer you. You don't have to manage that.

- **If Claude stops sooner than you'd like, just nudge it.** Say **"keep going"** or **"show the rest"** and it fetches the next page of matches. Claude decides how deep to go based on your question, so a quick nudge is how you ask for more.
- **For a single example or definition, one page is plenty** — Claude won't page through hundreds of results when a handful already answers you. That restraint is deliberate: focused results keep the answer sharp (and cheaper), so the depth matches the question.

---

3. The local tools: `whoami`, `signin`, `signout`

Three tools live inside the extension itself — they never round-trip to the IndyKnow backend. Ask Claude to run them by name.

- `whoami` — Shows the Keycloak username/email currently signed in, the roles your token grants (look for `indyknow:viewer` and/or `indyknow:developer` — see §2), your target environment, and how long until the access token refreshes. Use this first when something feels off.
- `signin` — Forces a fresh sign-in by discarding the cached token and re-running the browser device flow. Use this after an admin changes your roles, or when you want to switch Keycloak accounts.
- `signout` — Clears the local access token and the persisted refresh token. The next tool call will require signing in again.

> ***Note on `signout`:*** *Your browser's Keycloak SSO cookie is **not** touched. The next sign-in may skip the password prompt and recognize you instantly. If you want a full re-prompt — say, you're handing the machine to a colleague — clear cookies for the Keycloak host in your browser too.*

---

4. Switching environments

If you need to point IndyKnow at a different environment (a different customer database, dev vs. prod, etc.):

1. Open **Claude Desktop → Settings → Extensions → IndyKnow → settings**.
2. Paste the new **Environment Id**.
3. Restart the extension (toggle it off then on, or restart Claude Desktop).
4. Ask Claude to run `whoami` — the `Environment:` line should show the new id.

You do *not* need to sign out and back in. Your Keycloak session is independent of the environment id.

Gotcha: if a tool call returns a 401 or 403 right after switching, it usually means you don't have access to that environment — not that you're signed out. Run `whoami` to confirm your identity is intact, then check with whoever provisioned the env id.

---

5. Common errors & what they mean

"Server disconnected" / "Unable to connect to extension server" The extension crashed during boot or can't reach the IndyKnow backend. Check the in-app extension log for lines starting with `[indyknow]` — the boot wrapper writes a beacon on startup and a stack trace on any unhandled error.

A tool call returns 401 or 403 Three causes worth checking in order:

1. Your access token was revoked server-side (admin pulled it). Run `signin` — re-auth pulls fresh claims.
2. The environment id you're pointed at doesn't grant you access (a licensing problem, not an auth one).
3. The tool requires a role you don't have — every IndyKnow tool needs at least `indyknow:viewer`. Check `whoami`.

"Cannot reach IndyKnow at https://..." Network problem: VPN dropped, backend is down, DNS isn't resolving. Try opening the same URL in your browser; if `/api/health` doesn't respond, the backend is the issue.

The sign-in browser window didn't open The extension printed a sign-in URL to its stderr log. Find it in the in-app log (look for `Sign in:` from `[indyknow-auth]`) and open it manually.

"Environment selection timed out" Only applies to the in-flow environment picker (currently dormant — env id is supplied via extension settings). If you see this, just re-run the trigger.

---

6. Cost & speed

IndyKnow runs against **your own Claude Pro or Max quota** — every question you ask consumes tokens from your Anthropic subscription, not from a shared IndyKnow pool.

- **Narrower questions are cheaper.** "How does `WO_Charges` get totaled at invoice time?" is a fraction of the cost of "explain billing."
- **Start a fresh chat when topics change.** Multi-turn keeps prior turns in context — useful for follow-ups, expensive for unrelated questions.
- **Follow-ups cost slightly more than the first turn.** Prior context is re-sent on every reply. If a thread has gotten long and you're hopping topics, a new chat saves money.

---

7. Privacy: what goes where

Three systems see your data, each with a different slice:

- **Anthropic (Claude Desktop)** — receives the text you type, any screenshots in the chat, and the tool results IndyKnow returns. This is normal Claude usage; your subscription's privacy terms apply.
- **The IndyKnow backend** — sees only your Keycloak access token, your environment id, the name of the tool being invoked, and its arguments. It never sees the rest of your chat or anything you've typed before the tool call.
- **Your local machine** — your Keycloak refresh token is persisted to an AES-256-GCM encrypted file under your platform's appdata directory:
  - **Windows:** `%APPDATA%\IndyKnow\`
  - **macOS:** `~/Library/Application Support/IndyKnow/`
  - **Linux:** `~/.config/indyknow/`
  - *(Microsoft-Store installs of Claude Desktop: Windows keeps these files under `%LOCALAPPDATA%\Packages\Claude_...\LocalCache\Roaming\IndyKnow\` — the `%APPDATA%` folder will look empty in Explorer. IndyKnow always reports the real path; say "open my connections file" instead of browsing.)*

No screenshots, no clipboard contents, and no chat text outside of explicit tool arguments are ever sent to IndyKnow.

---

8. Reporting a bug

Found a real bug while exploring the code? Ask Claude to file it — IndyKnow can open a Defect in IndySoft's internal tracker for you, without leaving the chat.

- **Just say it in plain English:** "*That looks like a bug — report it.*" Claude drafts the Defect (summary, severity, what's wrong, what should happen, steps to reproduce, suggested fix), **shows you the draft in the chat first**, and files it only after you give the go-ahead.
- **Only the issue link comes back.** The reply is just the new issue key and a link — Claude never sees the filed ticket's contents. Your email and environment are stamped onto the ticket server-side; you don't supply them.
- **Verified code evidence (optional).** If a specific file range or stored procedure is relevant, IndyKnow verifies it against the *authoritative* source server-side and attaches the relevant snippet to the ticket as supplemental info (placed last). The source is captured at filing time and lands only in the internal Defect — it's never shown back to you in the chat. A reference it can't verify is dropped, not guessed.
- **Avoid duplicates.** Claude can search existing open Defects before drafting ("*has this already been reported?*"). If a similar ticket exists for the same finding, IndyKnow surfaces it and lets you choose: **file a new one**, or **add your context to the existing ticket**.
- **Fair-use limit.** Filing is rate-limited per user so a runaway loop can't flood the tracker; if you hit the cap you'll get a clear "try again in a few minutes" message.

> *Reporting uses the same sign-in as every other tool — any signed-in user can file. The feature ships with the IndyKnow backend, so there's nothing extra to install or update in your extension.*

This guide is versioned with the extension. Open `/tips` on the IndyKnow install page for the latest copy.