



SHPBL MCP

API • SDK • MCP SERVER

CONNECT. EXTEND. BUILD WITHOUT LIMITS.

The Strategic Master Library, now accessible everywhere.



MCP SERVER

Plug into SHPBL's capability engine through the Model Context Protocol.



API ACCESS

Programmatic access to thousands of reusable software capabilities.



DEVELOPER SDK

Build faster with our SDKs in multiple languages and frameworks.



SECURE BY DESIGN

Enterprise-grade security, governance, and performance built in.



BUILT TO SCALE

From side projects to global teams. Scale with your ambition, not complexity.

ONE LIBRARY. INFINITE BUILDS.

Harvest. Compose. Ship.



COMING SOON TO



OpenAI
Marketplace



Claude
Marketplace



Lovable
Store



API & SDK
Documentation



OWN IT. INTEGRATE IT. BUILD WHAT'S NEXT.

Full ownership of your builds. Full control of your future.



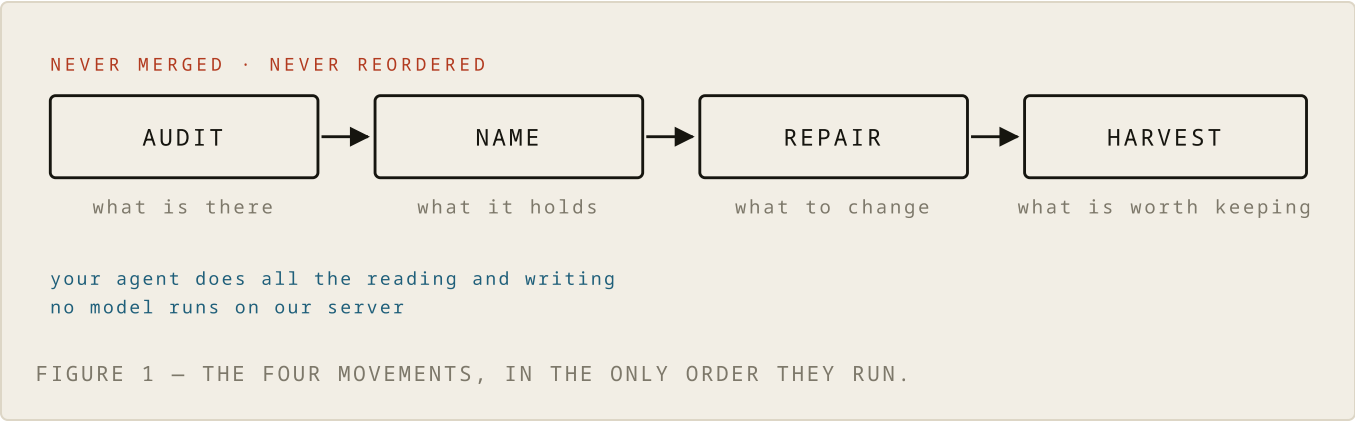
WE SHIP SOFTWARE.
THEN YOU SHIP SOFTWARE.

TOGETHER, WE SHAPE THE WORLD.

SHPBL.COM

User's Manual

SHPBL: Harvest Reusable Software Capability. Point it at a repository. It audits what is there, repairs what it finds, and harvests what is reusable into a library you can keep. This is the whole instrument, explained once, for people and for the agents they work with.



17

TOOLS

12

FREE, NO KEY

4

WAYS IN

0

MODELS ON OUR SERVER

WHAT IS IN THIS MANUAL

\$1-3	getting in	welcome · what it does · connecting it
\$4-6	using it	your first run · the seventeen tools · what to say
\$7-8	what to expect	how a run behaves · what it will never touch
\$9-10	what you keep	reusing your harvest · the three lanes
\$11-12	if you need us	troubleshooting · dev@KESJr.com

FIGURE 2 – HOW THIS MANUAL IS LAID OUT.

THE THREE LANES – same method, different keeping

Try	\$0	full evaluation, keep nothing (no key, no card)
Borrow	\$39/mo	cited rows + write-back into your own repository
Keep	\$499 once	the library as a tangible thing, mailed to your home, licensed forever, with a second copy given to a nonprofit

FIGURE 3 – THE THREE LANES: TRY, BORROW, KEEP.

CONTENTS

1. Welcome
2. What the server does
3. Connecting it
4. Your first run
5. The seventeen tools
6. Things you can just say
7. How a run behaves
8. What it will never touch
9. Reusing your own harvest
10. The TypeScript SDK
11. Keeping what a run produces
12. If something looks wrong
13. Support, ideas and suggestions

1 • Welcome

Thank you for bringing this in. SHPBL was built by one person who got tired of watching good software forget what it already knew — the same authentication guard written for the fifth time, the same retry loop, the same careful thing solved and then lost. Every caller who runs it is the reason it keeps getting sharper, and we hope you get something real out of it.

The free lane is a complete repository evaluation, not the full gauntlet. It returns the audit, ranked repair order and a sealed ledger of what it found, then stops before either library is searched. The full governed sequence — composition, verification and delivery — is the Practitioner step, \$39 a month with the first seven days free — shpbl.com/mcp-access. We would rather say that plainly than dress it up.

2 • What the server does

Four movements, always in this order:

WHO DOES WHAT

you ———▶ say what you want, in plain words
your agent ▶ reads, reasons, writes the diffs (your model, your keys)
SHPBL ———▶ the method, the mechanical work, the library
no model runs here · nothing of yours is stored

FIGURE 4 — WHO DOES THE READING, THE REASONING AND THE WRITING.

MOVEMENT	WHAT YOU GET BACK
Audit	The repository surveyed as it actually is — files, bytes, languages, the load-bearing files, risk signals — before any suggestion is made.
Name	Every capability it holds, listed with its signature, its file, its line and its stated contract.
Repair	An ordered remediation plan, with verbatim source handed to your agent so diffs land on real lines.
Harvest	What is reusable, folded into a sealed capability ledger you can keep in your own repository.

It reads public repositories that carry a proper open-source licence, your own repositories, and private repositories you have access to. **No model runs on this server.** The reading and the writing are always your agent's. We supply the method, the mechanical work and the library.

3 · Connecting it

The advertised door is `https://shpbl.com/mcp`. It speaks OAuth: your client sends you through a sign-in page, so no key is ever typed into the client.

```
{
  "mcpServers": {
    "shpbl": {
      "type": "http",
      "url": "https://shpbl.com/mcp"
    }
  }
}
```

Clients that can only carry a static key use the key-only door instead:

```
URL      https://shpbl.com/api/public/mcp
Transport Streamable HTTP
Auth      None
Header    Authorization: Bearer shpbl_mcp_YOUR_KEY
```

Either way, a key only matters when you want a run to persist. Everything else runs without one. The step-by-step per client — Claude, ChatGPT, Cursor, Codex — lives at shpbl.com/mcp-access.

THE SHORTEST WAY IN

Say to your assistant: “*Read <https://shpbl.com/llms.txt> and connect to the SHPBL MCP server described in it, then explain in plain language what it can do for me.*” Everything in this manual is published there in machine-readable form.

4 • Your first run

On the first message of a new conversation your agent calls `welcome` — the onboarding tool. It returns this manual in miniature: the greeting, the tool menu and the sentences you can say. Before work begins, checkpoint 0A asks which voice you want. The agent stops and waits; silence never selects a default. Then say something like “`audit owner/repo` with SHPBL”, and it begins.

If the repository you are after is not yours to change — an open-source project whose licence permits reuse, say — `harvest_repo` is the shortcut. It reads what that repository already does, fuses those affordances with the owned capability library, and names ranked proposals in plain language. Nothing is built until a person says which ones they want and what the upstream licence permits, and the target repository is never written to: no branch, no commit, no pull request. Approved proposals come back as seed modules into your own `.shpbl/`. If you only want the old whole-tree ledger, ask for `mode: "walk"`, and `estimate: true` on that mode still reads no source at all.

A FIRST RUN, END TO END

```
you say —▶ "audit owner/repo with SHPBL" —▶ 0A: choose voice
      |
step 1 – AUDIT   files, bytes, languages, load-bearing files —▶ one line back
step 2 – NAME    every capability, with file and line       —▶ one line back
step 3 – REPAIR  ordered plan, verbatim source for real diffs —▶ one line back
      |
      — check-in: "continue?" (at every required decision) —
      |
step 4 – HARVEST reusable work folded into a sealed ledger —▶ one line back
      |
free lane —▶ names exactly what it would have kept
borrow   —▶ writes .shpbl/ into your repository

cheaper pre-flight: harvest_repo with estimate:true reads no source at all
```

FIGURE 5 – A FIRST RUN, END TO END, WITH THE STOPS MARKED.

THE TOOLBOX AT A GLANCE – you never call these yourself

free, no key		paid, \$39/mo
welcome	onboarding	run_gauntlet
list_repos	what it may read	compose_capability
method_protocol	the discipline	library_search
run_contract	purpose + twelve steps	library_document
pin_source	which bytes, exactly	write_to_repo
evaluate_repo	audit + capabilities	
fix_repo	verbatim source	twelve free tools diagnose
harvest_repo	whole tree, batched	at full depth and stop at
build_intent	the build gate	the composition boundary;
subscription_status	who you are	the five paid ones compose,
library_index	what exists	cite and keep.
selfcheck_mcp	is it healthy	

FIGURE 6 – THE TOOLBOX: TWELVE FREE, FIVE SUBSCRIBED.

5 • The seventeen tools

In the order a run uses them. Twelve are free with no key; five are what a \$39/month subscription opens.

TOOL	LANE	WHAT IT DOES
welcome	FREE	This orientation: the tool menu, the example commands, and where the manual lives.
list_repos	FREE	Real repository names — yours, an owner's public ones, or everything a token can see — and whether a pull request may be opened there.
method_protocol	FREE	The whole discipline in words: how an audit, a repair and a harvest are conducted. Fetched once, cached, so later calls carry source instead of instructions.
run_contract	FREE	The canonical twelve steps as machine-readable authority: what each step must produce, the precedence, the two ways a run may land — and, given a run's files, the same pass/fail verdict the offline gate returns. Read it instead of inferring a step.
pin_source	FREE	Step one, pinned: one source digest over a GitHub tree, a directory or a repository ZIP, computed the same way every time, so a connected run and a fully offline run agree on exactly which bytes were audited.
evaluate_repo	FREE	The audit: a repeatable survey, every capability found with its file and line, a benchmark against the audited corpus, and the library rows that already cover what the repo does.
fix_repo	FREE	Verbatim source of the files you name, plus the ordered remediation protocol, so your agent writes real diffs against real lines.
harvest_repo	FREE	The shortcut to the capabilities in a repository you are licensed to reuse: it reads what that repository already does, ranks named proposals against the owned capability library, stops for your decision on each one, then hands back the seed modules for the ones you approved. It never writes to the target. A free call reports the reading and how much is offerable, then stops at the composition boundary.
compose_capability	\$39/MO	The harvest lane, and the point of the instrument: what your software already does, fused with owned SHPBL primitive capabilities — DREAM, EVOLUTION, MEMORY, DEFENSE, BRAIN and the rest of the canonical forty — returned as Capability Grants with the boundaries, their class and ports, a seed module, the wiring, the verification and the limits. Your agent writes the repository-specific code.
build_intent	FREE	The gate between finding something and building it: register what the harvest proposes and the server returns the licence verdict, the invariants it passed, and — where you are entitled — the order to write the software and its tests.
run_gauntlet	\$39/MO	The full conductor: audit, repair, harvest, library comparison, composition, verification and write-back from one tool, one step per turn. Requires Practitioner.
library_index	FREE	What SHPBL publishes: editions and prices, the seven strategy volumes, the public downloads, the case studies of real runs.

selfcheck_mcp	FREE	SHPBL's own audit, run against this live server, one pass or fail per verification axis. We hold ourselves to the method we sell.
subscription_status	FREE	The lanes, the prices, which tools meter, and — if you hold a key — this month's usage.
library_search	\$39/MO	The capability library as cited rows: the engineered catalog and the capability units, searched in plain words before you write anything new.
library_document	\$39/MO	One long document read paged: a strategy volume in full, the catalog outline, the report template, the standing order.
write_to_repo	\$39/MO	The result kept: repairs, ledgers and reports landed on a branch as a pull request for a human to merge. Never a push to the default branch.

6 • Things you can just say

You never call a tool yourself. You talk to the agent you already use, and it picks.

- “Welcome me to SHPBL and show me what it can do.”
- “List my repositories with SHPBL.”
- “Show me the SHPBL method before we start.”
- “Show me the SHPBL run contract, then gate my run against it.”
- “Pin the source digest for my repo before we start.”
- “Audit facebook/react with SHPBL and tell me what it can do.”
- “Use SHPBL to fix the auth handling in my repo.”
- “Harvest the capabilities out of this repo — I hold a licence to reuse it.”
- “Compose new capability for my repo from the SHPBL library.”
- “Register a build intent for that composition and tell me if I can build it.”
- “Run the full SHPBL gauntlet on owner/repo with my Practitioner key.”
- “What does SHPBL publish?”
- “Audit the SHPBL server itself.”
- “What does my SHPBL key cover?”
- “Search the SHPBL library for rate limiting before we build one.”
- “Read me SHPBL volume three.”
- “Open a pull request with everything SHPBL found.”
- “Audit / with SHPBL and tell me plainly what it can do.”
- “With my Practitioner key, run the full SHPBL gauntlet on my repo and check in at every step.”
- “Estimate a SHPBL harvest first — I want to know the cost before we start.”
- “Use SHPBL to write the fix for the thing it flagged, and show me the diff.”

- “Ask SHPBL whether this problem is already solved in its library.”
- “Have SHPBL audit itself so I can see whether the method is real.”

7 • How a run behaves

The procedure is law, not advice — and it is enforced by the server, not requested of the model. In practice you should see:

- **One step per turn.** Call, do, report a single readable line, then the next. Never the whole run in one breath.
- **Proof of the last step.** The conductor refuses step two and beyond without proof of the step before it, so no step can be skipped quietly.
- **A real stop at every decision.** The run enters `AWAITING_YOU`, asks one plain question, and waits. Checkpoint 0A selects the voice; 9A records a decision on each proposal; 9B records the separate authority to build or change the repository; 12A confirms final placement. Silence never supplies an answer.
- **No filling in gaps.** If the supplied material does not answer something, it writes “not present” rather than guessing.
- **A named deviation.** There is exactly one legal reason to depart from a step — it would break a stated rule — and when that happens the run stops and asks you.
- **Proposals you can actually read.** Before anything is built, the run shows you each proposed piece of software by name, what new ability it gives *this* repository, and why this repository in particular. If it cannot explain it to you in plain language, it is not allowed to propose it.
- **You approve one at a time.** Your 9A decision is recorded per proposal as `APPROVED`, `DECLINED`, or `DEFERRED/NEEDS_EXPLANATION`, so “this one, not that one” is a normal answer. Approved work can continue even when other proposals are declined. General enthusiasm and silence are never read as blanket approval.
- **Permission to build is not permission to change your repo.** Approving a build never authorises touching your repository; that is a second, separate yes.
- **An interrupted run restarts cleanly.** If a run breaks partway, it resumes from the beginning of the procedure and reconciles anything held back, rather than pretending the missing part happened.

A run that ends in “change nothing” is a good run. The instrument is built to reject; most of a catalogue is wrong for any given repository, and saying so is the work.

An offline run has its own honest finish. If no write tool exists, or you choose to place the files yourself, it can complete as `PACKAGED_FOR_REVIEW`. The offline edition does not require a network connection, subscription, hosted tool, or online write-back to finish.

8 • What it will never touch

Some files are read by the tools you build with, so a change that looks obviously right in a diff can break your running app the moment it merges. A run reads them, writes about them, and hands the decision to you and to whatever agent already owns them:

- environment files of any kind
- every lockfile — bun, npm, pnpm, yarn, Cargo, Go, Composer
- generated code — route trees, schema types, *.gen.*, __generated__/
- agent instruction files — AGENTS.md, CLAUDE.md, .cursor*, .lovable/
- backend wiring, migration history and generated database clients
- build, CI and deploy configuration — workflows, Vercel, Netlify, Wrangler, Docker
- version-control internals, vendored output, and anything holding a credential or signing material

The write-back tool refuses those paths outright, at every tier, with no override. When a real finding lives behind one, your agent hands it to you as advice instead: the file, the finding, the change it would make, and what it would affect.

Your keys never come back out

Your subscription key, and any repository token you pass with it, are used to answer the call and nothing else. They are never written into anything a run produces — not a report, not a ledger, not a harvest, not a commit — and they are never read back to you. If a run needs to refer to a credential, it names where the credential came from, never its value. Anything shaped like a key is withheld from what the server serves, whether or not your call was the one that carried it.

If a run finds a credential committed inside a repository it read, that is a finding you are told about: the file and the name of the key, with the value withheld and a plain instruction to rotate it and stop tracking the file. Withholding a value is never withholding the finding.

What this does not do: it keeps credentials out of what a run writes and says. It cannot clean a key out of a repository's history and it cannot rotate anything for you. A key that has been pasted somewhere it should not have been needs rotating.

9 • Reusing your own harvest

A kept run writes a .shpbl/ folder into your repository, with a README.md listing what was harvested. On the next run, ask your agent to read those lines and pass them back as the run's own library. It then checks each concern against what you already solved before citing anything new — your library compounds, and it stays yours.

Those entries are held for that one call and never stored by SHPBL. That folder is your private SHPBL capability library: it is yours outright, it grows with every kept run, and a later run can build on it.

10 • The TypeScript SDK

There are four doors onto the same method: the offline zip, the MCP server, the plain HTTP API at shpbl.com/api-access, and — for TypeScript and JavaScript — a published client.

```
npm install @shpbl/sdk
```

Node 20 or newer, or any runtime with `fetch`. MIT licensed, zero runtime dependencies, no model inside it. Source: [packages/sdk of github.com/SweetKenneth/shpbl-master](https://github.com/SweetKenneth/shpbl-master).

```
import { ShpblClient, text } from "@shpbl/sdk";

const shpbl = new ShpblClient({ key: process.env.SHPBL_KEY }); // key optional
const audit = await shpbl.evaluateRepo({ repo: "owner/repo", brief: true });
console.log(text(audit));
```

Every tool in the table above has a method, and each tool's tier travels inside the package — so `canCall()`, `tierOf()` and `ask()` answer “may I call this?” with no request. `gauntlet()` holds the conducted run's bookkeeping: the fold token, the server-dictated step order and the pause for a person. The ledger stays yours; your model writes it. `compatibility()` reports whether the release you pinned still matches the live server.

The client reference — options, error classes, the drift report — is at shpbl.com/sdk. The method it runs is this manual.

11 • Keeping what a run produces

LANE	PRICE	WHAT IT IS
Try	Free	A complete repository evaluation, no key and no card. It is not the full gauntlet: it stops before either library is searched, with no composition, candidates or retained result.
Borrow	\$39 / month	The run continues past the composition boundary: both libraries searched, new capability composed and verified, the library read as cited rows, and write-back so a run persists into your own repository. First seven days free, cancels any time.
Keep	\$499 once	The Complete Master Library as a tangible thing: a hard copy mailed to your home, licensed to you in perpetuity, and a second copy contributed to a nonprofit you choose.

Whatever a run produces is yours outright — no licence back to us, no contribution loop, no claim. The library and the technique stay ours. That line never moves.

12 • If something looks wrong

WHAT YOU SEE	WHAT IT USUALLY MEANS
The client says the tool needs a subscription	You asked for a lane that keeps something. Ask for <code>subscription_status</code> — it reports which identity the server resolved for you.
A private repository reads as empty	No access. Pass a <code>github_token</code> , or connect the SHPBL GitHub App to your key on the access page.
The agent tried to run the whole gauntlet at once	The server will refuse out-of-order steps. Say “continue” at each check-in and let it walk.
You want to know the server itself is healthy	Say “run the SHPBL self-check.” It audits the live server against its own axes and reports pass or fail.

13 • Support, ideas and suggestions

Write to dev@KESJr.com. A real person reads it, and good ideas from callers have changed this server before. If a run found something surprising in your repository, we would like to hear about that too.

When you are done evaluating the free lane, shpbl.com is where you borrow it by the month or order the hard copy.

Kenneth E. Sweet Jr. • Solo Founder • Infrastructure Architect
SHPBL — Abilene, Texas • shpbl.com • dev@KESJr.com
User's Manual for MCP server v1.44.0 • 2026-09-08 • made by humans who care.