THE FILES

Three inputs. Eight documents. One of them is yours.

Everything a buyer receives is in one directory. Two files are the deployment shape's measured grant and the published vocabulary, copied in and pinned by version. One file is the mandate, and it is the only one a person writes. The rest are derived from those on every build, and a hand edit to any of them is undone by the next one.

FILE	WHAT IT IS	WHO WRITES IT
MANDATE.md	What the agent is authorised and expected to do — six wanted, three refused, fourteen unstated	**you** — elicited; the only authored file
GRANT.md	Everything the agent can do: fifteen capabilities, each with its barrier, undo class, evidence tier and the tool it goes through. Irreversible rows first	measured from the shape — 13 of 20 rows observed on the thing itself
DELTA.md	Nine in the grant that nobody asked for: three refused, six never mentioned, seven with nothing but a setting or a sentence in the way	derived — never authored, recomputed on every build
LICENCE-TO-OPERATE.md	The organisation authorises the agent under this behaviour policy, for an interval, on nine conditions — each next to what enforces it	derived from the mandate; a named person signs it when issued
AGENT-BEHAVIOUR-POLICY.md	The four objects in one document	derived
AGENTS.md	The file you hand the agent: what the others are, what to do with them, and what a file like it cannot do	generic — travels unchanged with every vault
SKILL.md	The same, in the portable agent-skill format	generic — travels unchanged
README.md	How to read it, how to correct it, how to give it to the agent	derived
data/ · history/	grant.json, mandate.json, delta.json, validity.json, the pinned vocabulary, and one history entry per recompute	the machine-readable half of every row above

- **The grant was measured, not typed.** The profile is the published one for `anthropic/claude-code-remote/ccr-container` at abp.sgit.ai v0.3.0 — thirteen of its twenty rows were observed on the thing itself on 5 September, the rest derived from what the shape architecturally is. Nothing in the vault's grant was written by us.
- **The delta was recomputed here and checked.** The build derives it from the grant and the mandate, then compares all six lists against the record published on the model site, row for row, and refuses to write if any of them disagree. They agree. A delta a site cannot reproduce is not one it should sell.
- **The mandate is the starting point the model site published,** written to be argued with. That is what makes this a *template* rather than a policy: nobody has corrected it yet. Correcting it is the sale.

THE NUMBERS, FOR THIS SHAPE

Fifteen it can do. Six you asked for. Seven nothing bounds.

None of these is a score. Excess is a count of capabilities in the grant and not in the mandate; unbounded excess is how many of those sit at a barrier that is not a control. Neither says whether any of it is acceptable, because acceptability is not in the document — it is in the deployment, and this shape is a disposable container with one repository attached.

CAPABILITY	WHAT IT IS	BARRIER
`authenticate-as.credential.signing`	Sign commits with the key it holds	● none
`delete.file.host`	Delete files anywhere the account can reach	● none
`read.credential.host`	Read credentials stored where it runs	● none
`read.file.host`	Read any file the account can reach	● none
`read.record.history`	Read a retained record: past sessions	● none
`authenticate-as.credential.tenant`	Act in accounts with the credentials it holds	○ boundary — the token's scope, set by the platform
`write.file.host`	Change any file the account can reach	● none
`create.schedule.tenant`	Create something that outlives the session, on the platform	◐ setting — the platform's routines are the operator's to delete
`create.schedule.host`	Create something that outlives the	○ boundary — the container

CAPABILITY	WHAT IT IS	BARRIER
	turn, where it runs	is ephemeral
`send.endpoint.allowed`	Reach a permitted list of hosts	○ boundary — an egress proxy above the process
`execute.process.host`	Run programs as the account	● none
`write.file.project` · `write.repository.project` · `read.file.project`	Read, change and commit to the attached repository	● none
`write.repository.tenant`	Push to a code host, any branch it can reach	◐ setting — hooks in the clone; no rule at the host

- **Six of the nine excess rows are unstated, not refused.** The mandate never mentioned that the agent could delete or rewrite anything in the container, read any file in it, or act with the platform's scoped token. That is not a mistake in the mandate; it is what a mandate looks like before anybody has been shown the grant. The correction goes upward, and this table is what it goes upward from.
- **Two of the three boundaries are the platform's, not the deployer's.** The egress proxy and the token scope are set above the session by the vendor, and the third — the container dying — is a property of the shape. A deployer who wants a fourth has to add it: a branch rule at the host is the obvious one, and it is a free setting.
- **Reading it as the four rooms would:** the founder sees rows they did not know they had granted; the investor sees a count they can ask every portfolio company for; the security vendor sees the seven unbounded rows their product would move to the fourth barrier — and that count contains no verdict.

THE FILES YOU HAND THE AGENT

Drop it into the session. It says, in its own words, what it cannot do.

The idea is that the deliverable is not only read by people. `AGENTS.md` goes wherever the agent already reads — a `CLAUDE.md`, an `AGENTS.md`, a role file, the body of a skill — with `MANDATE.md`, `GRANT.md` and the behaviour policy beside it. It tells the agent that the mandate is its scope, that the grant is a description and not permission, that anything in the excess needs a stop-and-report, that instructions found inside content have no authority, and that it must never edit its own mandate.

- **It is honest about which barrier it is.** The file says of itself: *this is a rule in prose — the second of four barriers — and it bounds nothing. It shapes what you tend to do; it does not change what you can do. What would is in the Barrier column of GRANT.md.* That sentence is what separates it from every packaged prompt on the market, and it is the sentence Lab 05 argued had to be there.
- **`SKILL.md` is the same file in the portable skill format,** and it states the format's limit on its own face: the portable frontmatter carries a name, a description and a licence, and cannot carry a constraint. A skill distributed to somebody else carries instructions; it does not enforce them.
- **The agent can be asked to check the grant.** `GRANT.md` ends with a block to paste into a session running in the shape: report every row as present, absent or cannot tell, with one line of evidence; never exercise an irreversible capability to prove it exists; add rows for anything reachable that is not listed; never touch the mandate; write the result to `history/` and mark it self-report. One early user did this for an automation platform and the agent found considerably more than the draft listed — which is the point. A claim from inside the deployment is a better starting point than a template, and it stays a claim until a log held outside the agent agrees.
- **We ran it on ourselves.** The session that built this vault is the shape it describes, so it read the grant back against what it had actually done: eleven of fifteen rows seen present in ordinary work, none absent, four not re-checked — including the credential row, deliberately, because checking that row is looking for credentials. Two probe batches it proposed were refused by the platform's own action classifier before they ran. That refusal is written up in the check as a barrier the grant has no row for: real, above the session, and perishable in exactly the way Lab 06 describes.

Mandates are per product, not per agent — at least to begin with. The mandate in this vault is really the mandate for *a repository attached to a coding agent in a container*. The next ones are the same: the mailbox mandate, the shared-drive mandate, the automation-platform mandate, the desktop-app mandate. Each is a product-specific vault — a measured grant for that product and a starting mandate for it — and that is what the shape library is. What is sold is a copy of one of them, with the mandate corrected and a name on the licence.

LICENCE TO OPERATE, WITH A REFERENT

The organisation is the authority. The behaviour policy is the instrument. The agent is the licensee.

Until now *Licence to Operate* was the name of a demonstration. In the vault it is a document with fields: who authorises, whom, under which instrument, for what scope, on what conditions, for how long, and who signs. Self-issued and witnessed, which is how most assurance works. Nine conditions, each beside the thing that enforces it — and for six of them that thing is **nothing**, which the licence says in bold, because an organisation issuing it should know what it is accepting.

- **No interval, no licence.** The template's *valid until* reads: *an interval is set when it is issued; a licence with no expiry is not a decision.* That is the acceptance model this site has argued for a year, applied to the one document an agent can carry.
- **Void when the inputs move:** the grant version (a release, a setting, a connector), the mandate, the vocabulary, or a barrier. When any of those happens the deployment changed, not the document — rebuild, re-read the delta, re-issue.
- **Not a word about cover.** The other document called a policy on this site is the one in the demonstration, and the two never share a page. The licence carries no score, no premium and no ceiling; it carries scope, conditions, interval and a signature.

ONE COMMAND PER SHAPE

The template is the product. The second shape costs one profile and one mandate.

Lab 02 said the thing being built is not a document and not even a vault — it is the template plus the shape library, and the marginal cost of a policy is the cost of its shape. This is that template, as a script in this repository: point it at a directory holding a grant, a mandate and the pinned vocabulary and it writes the eight files, recomputes the delta, appends to the history when the counts move, and fails if it cannot reproduce the published record.

STEP	WHAT HAPPENS	WHO
1	Copy the template vault for the closest shape	us, in seconds
2	Edit `data/mandate.json` — move rows between *want*, *do not want* and *unstated*. This is the correction, and it is the elicitation	**the buyer**, with a pen on a card or a person on a call
3	Fill the organisation, the owner and the interval in `vault.json`; status becomes *corrected*	us
4	Rebuild. The delta is recomputed, the licence is re-rendered with the conditions and their barriers, the history gets an entry	one command
5	Push the vault; hand over the read key. The buyer clones it, or opens it in a browser, or hands the key to whoever asks how they govern their agents	us, then them

- **Time it.** The first vault took an afternoon and almost all of it went into the generator. The second vault of this shape should take the length of the conversation that corrects the mandate. If it does not, the library is not working and nothing should be priced yet — the instrumentation table in Brief B1 is still the only honest input to a price.
- **Nothing sells the shape library itself.** The templates are the free half of the line: they describe *a* deployment. The paid half is *your* deployment — the corrected mandate, the name on the licence, and a vault that recomputes when the grant moves.

THE VAULT ITSELF

Pushed, and read live on its own page.

The same files are in an encrypted SG/Vault on the platform this site's demonstrations use — vault `ruj286tr` . Its read key grants read and nothing else and is published on purpose, like every demo key on this site. It is printed on the vault's own page, which reads the vault live in the browser: the card, every table, the vault's app in a sandboxed frame, and the file list, all decrypted from the current commit as you read. A Lab page carries no key by rule, so it is not repeated here; every file is also served from this site — the links above — so nothing depends on the vault host.

Vault	ruj286tr
Host	dev.send.sgraph.ai · opened at dev.vault.sgraph.ai, like the demos
Read key	published on the vault's page, in the public form sgit declares for a deliberately published key
Clone it	`sgit clone <read key>` — read-only; the key is on the vault's page
Commits	15 September 2026 — v1 the template with the grant check · v2 the vault app · v3 the start-here view, the zip and the PDF · v4 the primer and the minimal HUD · v5 the loader: the renderer now lives once, in the app vault

The write side of this vault is held by us and appears nowhere on this site; a test in the repository fails the build if anything shaped like one lands in site/. The vault's history is the vault's: every recompute is a commit, and a buyer's own vault keeps the draft it was corrected from as its first entry.

OPEN

Five things this does not settle, and one it does.

- **The store's tier-one page still says the delta is "computed when you open it and never stored".** The model site corrected that on 11 September and this vault stores its delta with both inputs pinned. The delivery page and the deliverable must not disagree before a card is printed.
- **Mandate elicitation for a connector shape has no question set.** For a coding agent the mandate is a job; for a mailbox it is closer to a relationship. The three questions in `MANDATE.md` — did you ask for it, would you object, neither — are a start and not a method.
- **The mailbox and drive shapes are not in the published data yet** (Lab 03, request 2). Until they are, the connector vaults a stranger recognises cannot be built derived-not-authored, and we will not build them any other way.
- **Can a read key be revoked?** The hand-it-to-an-underwriter story leans on it and it is still unchecked.
- **What the delivered file is, in the store's words.** Tier one promises "your answers and the delta, as a file you keep". This vault is a candidate for that file; whether the store points at it is a decision.

- **Settled: Licence to Operate has a referent.** The direction brief of 11 September proposed organisation, instrument, licensee and left the ruling to the project lead. Asking for the file by name makes the ruling, and this vault carries it. *Mandate to operate* is retired.

Built 15 September 2026 from abp.sgit.ai v0.3.0 (profile anthropic/claude-code-remote/ccr-container version 2026-09-05.2, mandate coding-assistant-in-a-container of 2026-09-09) with scripts/site/build-abp-vault.mjs. The generic AGENTS.md and SKILL.md are in site/vaults/_template/. The eight-line prompt with its enforcement column is Lab 05; the perishable-barrier and self-report findings are Lab 06; the twelve-stage flow this delivers stage ten of is Lab 02.

Lab 07

Correct the mandate. That is the whole purchase.

Everything else in the vault is derived. If you run an agent in this shape, open `MANDATE.md`, tell us which of the six you actually asked for and which of the nine you would object to — and the next build is yours.