AVENKIN · FIELD ASSIST

Building a Field Assist vault from your own manuals

APPLIES TO	builds with the manual tier (September 2026 or later)
LICENCE	a Field Assist subscription or an organisation licence
FORMATS	PDF (text layer or scanned), EPUB, Markdown, plain text

A vault is a folder. It holds a manifest that names things, a few short markdown files the assistant always has in front of it, and the OEM manuals it searches when a technician asks. You build the folder on a computer, move it to the phones that will use it, and import it once on each.

For an example with selectable procedures, see the aviation operations workflow discovery vault. It contains seven proposed draft workflows for organisational adaptation, not operator-approved aviation procedures or a live integration.

What you are building

Every vault has the same shape. The names in `manifest.json` have to match the files exactly, but the file names themselves are yours to choose.

```
rtu-service-vault/
├── manifest.json              names everything below
├── safety.md                  core: always in context
├── error_codes.md             core: fault codes by model
├── models.md                  core: nameplate quick reference
├── documents/                 reference: retrieved per question
│   ├── RTU-500-service-manual.pdf
│   ├── RTU-500-wiring.pdf
│   └── RTU-700-install-guide.pdf
├── procedures/                optional: step-by-step flows
│   └── low_pressure_diagnostic.json
└── sites/                     optional: how the units at one site work together
    └── hillside-house.json
```

The two kinds of content behave differently, and getting the split right is most of the work.

	CORE FILES	DOCUMENTS
When read	Into the assistant's context on every single question	Only the passages relevant to a question, retrieved at that moment
Size	Keep the total under about 32 KB; the importer warns above that	No practical limit; a 300-page manual is fine

	CORE FILES	DOCUMENTS
Right for	Safety rules, fault-code tables, model and refrigerant quick references	Full service manuals, wiring guides, installation and commissioning documents
Citations	Source file name	Manual title and page number
Editable on the phone	Yes; edits are kept separately from your originals	No; re-import to change

---

Before you start

1. Install the TestFlight build on the iPhone, and pair the glasses if you use them.
2. Open **Settings › Field Assist**. If your organisation issued a licence code, paste it and tap **Activate Licence** — the status card should show **Tier: Team**. A Field Assist subscription on the phone's own Apple ID does just as well for building vaults: vaults of your own and manual indexing come with a subscription or an organisation licence. The one-time unlock some early users hold covers the bundled vaults only; audited PDF export and organisation-issued configuration stay with a team licence.
3. Turn on **Enable Field Assist** on the same screen.
4. Have a way to get a folder onto the phone. iCloud Drive in the Files app is the simplest; AirDrop of the whole folder also works.

Step 1 · Prepare the manuals

The importer reads the text layer of a PDF. Open each manual on your computer and try to select a sentence with the cursor. If the text highlights, the page imports as-is. If nothing selects, the page is a scanned image. Scans still import; you have three ways to handle them, and you can mix them per manual.

Route A · Let the phone read it. Nothing to install. Import the scanned PDF like any other. The app reads scanned pages itself during import, page by page, on the device. A 300-page scan takes minutes; keep the app open, and if it is interrupted the import resumes where it stopped. The vault row afterwards shows how many pages were read this way and how many came out low confidence, so you know which manual to replace with a better copy. Any answer drawn from a recognised page carries a note to verify figures against the printed page, because numbers are where recognition fails quietly.

Route B · Add a text layer on any computer (recommended for anything you will reuse). Run the scan through an OCR tool that writes a searchable PDF, then import that PDF. The pages stay the same, so citations still match the printed book, and the phone does no recognition at all. Free, works on Windows, Mac and Linux:

```
ocrmypdf --language eng RTU-500-service-manual-scan.pdf RTU-500-service-manual.pdf
```

OCRmyPDF is at ocrmypdf.readthedocs.io; on Windows install it through the instructions there (it runs under Python, with Tesseract as the recognition engine). Adobe Acrobat's "Recognize Text" does the same job if you already have it, and so do most scanner drivers' "searchable PDF" settings if you are scanning the book yourself. Whatever tool you use, open the result and select a sentence to confirm the text layer is there before you copy it into `documents/`.

Route C · The repository's extractor, on a Mac. If you have a Mac, the app's repository includes a script that does the same per-page read and writes Markdown with a `Page N` line before each page, which lets you open the text and correct a misread figure before anyone relies on it. It is not in the app; download it from the public repository:

```
curl -O https://raw.githubusercontent.com/straff2002/OpenGlasses/main/Scripts/extract-manual-text.swift
swift extract-manual-text.swift RTU-500-service-manual.pdf
```

It needs Apple's command-line developer tools (`xcode-select --install`). It writes `RTU-500-service-manual.md` beside the PDF and prints how many pages were recognised and how many came out low confidence. Put the `.md` in `documents/` instead of the PDF and give it the same title.

The script also reads the manual's **structure from its type** and writes it into the Markdown: a line set in bold at the manual's body size becomes a `## Heading`, a bold `FIGURE 58` or `TABLE 16` caption becomes a `### Figure 58 — …` line, and a page that is mostly small labels with no sentences in it — a wiring diagram, a dimensions drawing, a table page — is tagged `<!-- page: diagram -->`. The closing summary says how many of each it found. That is what makes a citation read *page 62, §Turning Off Gas to Unit* rather than naming whatever line happened to be in capitals, and it is also how a wiring question reaches the wiring page. Everything the marks describe stays in the text, so the file is still yours to read and correct; if the extractor marked something wrongly, edit the line. Importing the PDF itself gets the same treatment inside the app, so the two are interchangeable — what Route C buys you is the chance to look first.

It also warns, per page, when a manual prints its own page number at the top of the page and that number disagrees with the page's position in the PDF — usually an unnumbered cover or roman-numeral front matter shifting everything by a page or two — and counts them in the closing summary. Citations always name the physical page, the one you reach by counting from the front, so either fix the PDF (delete or add the front matter until the numbers line up) or accept the offset and tell your technicians about it. Zero warnings means a citation reads exactly as the page is printed.

Import the PDF rather than the extracted text when you want the drawings. Both routes give the same citations and the same answers, but only the PDF keeps the *pages*: when a question reaches a wiring diagram, the app opens that page on the technician's phone — the real drawing, zoomable — and sends it to the assistant as a picture, so the answer is read off the diagram rather than off the list of terminal labels underneath it. A manual imported as extracted text still cites its figures correctly; it simply has no page to show, and the app says so when a technician asks to see one. Route C is for looking at what the extractor read and correcting it; if the drawings matter, import the PDF once you have.

Bundle the original beside the extracted text for anything a manufacturer requires as an SOP. If you take Route C — or receive a manual as text from anyone — put the manufacturer's PDF in `documents/`

beside it and name it in the manifest as that document's `source`. It is copied into the vault and hashed, and it is never indexed, so the text stays what gets searched and your corrections to it still count. What it buys is the page itself: a technician who taps a citation sees the manufacturer's own page, and the header says so and says whether the file is still byte-for-byte the one you imported. Without it the same citation opens the extracted text and says, in as many words, that the original is not in this vault — which is exactly what a compliance reviewer will ask.

Page numbers are preserved automatically either way, so a citation reads *RTU-500 Service Manual, page 42* and the technician can open the paper copy to it. That only works if the PDF's pages match the printed pages, which is true of nearly every OEM PDF.

A few habits that pay off:

- **One file per manual, one manual per model family.** A combined 900-page binder still works, but a technician hears the title in every citation, so shorter, specific titles read better.
- **Trim what does not help in the field.** Warranty terms, ordering forms and marketing pages add index time and add nothing to answers. Delete those pages before you copy the file in. Trim from the back of the book: deleting front pages shifts every page's position against its printed number. Leave a manufacturer's PDF that you bundle as a `source` untouched.
- **Check for the same document under two names.** Manufacturers re-publish a document under a new file name or link, so two downloads with different names, and even different checksums, can hold identical text. A matching page count is the quick tell; then compare the text (`pdftotext a.pdf - | diff - <(pdftotext b.pdf -)` on a Mac or Linux). Keep one copy. A duplicate brings every passage back twice under two titles, crowding out passages the technician needed.
- **Keep fault-code tables in the core too.** The manual's fault-code chapter is retrieved fine, but a code that also sits in `error_codes.md` is found instantly by the nameplate lookup without a search at all.
- **A code book or standard imports the same way.** Keep it as one file so a citation's page matches the printed book. Clause and rule numbers are looked up exactly — "clause 3.9.4.2", "rule 26-700", "section 110.26" — and the answer leads with the clause itself rather than the places that refer to it. Trim the index at the back, but leave the contents pages in if you want; they are recognised and never quoted as the clause.

Step 2 · Write the core files

Markdown, plain text, any editor. Two conventions matter because the app relies on them.

Headings define lookup sections

When a technician says "*look up fault T02*" or points the glasses at a display, the app splits each core file at `##` and `###` headings and returns the whole section that contains an exact match for the code. So group codes under a heading per controller or family, name the models it covers in the section's first line, and put each code in a table row or at the start of a line. A section that runs on for pages will be read back in full, so keep sections tight.

```
# Fault Codes

## Series 5 controller faults

Models: Acme RTU-500, RTU-550. Source: RTU-500 Service Manual.

| Code | Meaning                     | First check                                 |
|------|-----------------------------|---------------------------------------------|
| ZX9  | Low refrigerant charge      | Sight glass, then subcooling before adding |
| ZX3  | Condenser fan failure       | Fan capacitor, fan relay                    |
| ZX1  | Discharge line sensor fault | Sensor resistance vs. table on page 61      |

## Series 7 controller faults

Models: Acme RTU-700. Source: RTU-700 Service Manual.

| Code | Meaning                     | First check                                 |
|------|-----------------------------|---------------------------------------------|
| F12  | High pressure lockout       | Condenser coil, fan staging                 |
```

> **A model number goes in exactly one heading: the model's own heading in `models.md`.** The app reads every `##` and `###` heading in every core file as a possible list of models; that is how it knows which machine the technician means (see **A nameplate reference** below). If RTU-500 also headed a section in `error_codes.md`, saying "RTU-500" would match two sections, and the session could not settle on the machine. A manual's own number (`SM-5002`) in a heading reads as a model too. Put those numbers in the section's first line, as above: lookups still find them there, and they don't compete with the model's own heading. Topic headings like *Series 5 controller faults*, *Service Test 56* or *R-454B detection* are safe.

A safety file the assistant repeats before risky steps

The assistant is instructed by the manifest to remind the technician of safety steps before opening electrical panels or the refrigerant circuit. Put the reminders you want spoken in `safety.md`, in your own words and to your own site rules. Lockout, PPE, recovery before opening the circuit, gas isolation for furnaces, and anything specific to Manitoba code belong here.

A nameplate reference

`models.md` is where model and serial patterns live: what the numbers on the plate mean, refrigerant per model, nominal charge, controller type. When the camera reads a plate, the model number is matched against these sections the same way a fault code is.

Put every spelling of a model in that model's own heading. Manuals and nameplates rarely agree — `SLP99UH090XV60CK` on the plate, `SLP99UHXV-090-60C` in the parts list, `090XV60C` in a table. A lookup returns whole `##` sections in file order and stops at three, so a spelling that appears anywhere earlier in the file — an introduction, a summary table, a compatibility note — spends one of those three slots and can push the model's own section out of the answer. A heading that carries all the spellings costs nothing and cannot be crowded out:

```
## SLP99UH090XV60CK (SLP99UHXV-090-60C, 090XV60C, SLP99UH090V60CK)
```

Put the alternative spellings in brackets rather than separating them with slashes. The part before the bracket is the model's name: what the assistant says back and what the **Active Session** card shows. With slashes there is no bracket, so the whole line becomes the name.

Many models, many manuals: a model-to-manual map

A vault for one product line can let the manuals speak for themselves. A vault that spans several families (dozens or hundreds of models, a few service manuals, many owner guides) needs a map, or the assistant will happily answer a question about one model from a passage written for another. Make `models.md` that map: one section per model, listing the documents that apply to it and nothing else.

```
## RTU-500A (RTU-500A01, RTU-500A02)
Service: D01 (engineering versions 01 and 02 only). Owner: D04. Other: D07.

## RTU-550A
Owner: D04. No service manual loaded.
```

- **Give each document a short ID** (`D01`, `D04`) and start its manifest `title` with it, so the map, the citations and your own records all point at the same thing. A `document_index.md` core file with one line per ID (`D01: RTU-500 Service Manual SM-5002 Rev B`) gives the assistant the lookup.
- **Confirm applicability from evidence:** the model list on the manual's cover or inside it, or the manufacturer's own catalogue entry for the model. A model number that merely starts the same way is not evidence. When a manual lists engineering versions (suffixes), record exactly the ones it lists.
- **Say the difference between kinds of document.** An owner guide or quick-start sheet does not make a model serviceable from the vault; the map should show which models have a service manual and which do not, so the assistant can say so.
- **Add a rule that holds the assistant to the map**, for example: "*Identify the exact model first, then use only the documents listed for it in models.md. A passage written for another model is not evidence.*"
- **Naming a document assigns it.** When a model's section names a document, that document is assigned to the model. With the model on the job, passages from its assigned documents come first, every other document ranks down (it can still be found, so the assistant can say a figure belongs to another model), and the assistant is told which documents apply. Two things count as naming a document: its ID, which is the first word of its `title` when that word holds a digit (`D01`), and a word of its title that looks like a document number, at least six characters of which five are digits (`SM-50021`, not `SM-5002`). Nothing else does, so a refrigerant, a voltage or a size in a section's prose assigns nothing. List every document a model has: a section that names one document by accident ranks all the others down. A section that names none changes nothing.
- **Write for one supported check.** On a symptom, the assistant gives one check the passages support, says what to observe, and uses the answer to pick the next. It does not call a check the manufacturer's first step unless a page says so, and with nothing supported it names the gap. A manual with no symptom section can only support a check, not an order of checks, so say so in your

rules rather than asking for one. Do not write "state the gap and escalate": a gap is said to the technician, and an expert is offered, not called.

- **Keep rules proportionate.** A rule demanding the serial range before every question slows down a filter-light question for no gain. Ask for the details only where a manual makes the answer depend on them.
- **Mind the budget.** One short section per model is about 40 bytes; three hundred models is roughly 12 KB of the 32 KB core.
- **Sibling brands.** One service manual often covers several of a manufacturer's brands. Decide deliberately whether those models belong in the vault: a model with no section is refused as out of scope, even when a loaded manual covers it.

Symptoms, in the manual's words

A technician says what the machine is doing. A service manual is arranged by part: it has a test for the condenser fan, and no page headed *not cooling*. So a symptom often misses the page that would help, however good the search, and even a page that is found says how to test a part, not which part to test first. Where the manuals give no order, a person who knows the machines can give one in a core file.

Add a `symptoms.md` core file, list it in the manifest's `files`, and write one section per symptom:

```
# Symptoms

## Not cooling, compressor running
Said as: not cooling, warm cabinet, runs all the time, won't pull down.
Applies to: the RTU-500 family. Not the RTU-550A, which has no service manual loaded.
The manual's words: condenser fan, condenser coil, suction pressure.
Checks, in this order:
1. Condenser fan turns with the compressor. Fan test. D01 page 41 (printed 3-7).
2. Condenser coil clean. Coil cleaning. D01 page 58 (printed 4-12).
3. Suction pressure against the chart. Charging chart. D01 page 63 (printed 4-17).
Reviewed: 12 November, against D01 Rev B.
```

- **Symptom words.** Write what your technicians actually say, several ways. These are what the assistant recognises the symptom by.
- **The families it applies to.** A check for one family is wrong on another. Name the families in the section's text, not its heading: a model number in a heading reads as a model (see above). Say which models it does not cover and why.
- **The manual's words.** Copy the part's name as the manual prints it, spelling and all, so the assistant can search for it. Technicians say *fan*, and the manual may say *condenser fan motor*.
- **Ordered checks, each with a document and a page.** Give the PDF page, and the printed page when it differs. A check with no page is not a check: leave it out. Use the document IDs from your model map.
- **A person must check it against the pages.** Open every page cited and read it. The order is yours, not the manufacturer's, unless a page says otherwise: a wrong first check costs a call-out, and an unchecked one looks just as sure as a checked one. Record who reviewed it and against which revision, and review it again when a manual is replaced.

- **Add a rule that says what the file is**, for example: "*For a symptom in symptoms.md that applies to the unit, give its checks one at a time, in that order, each with its document and page. Say the order is this vault's, not the manufacturer's. Read the cited page before relying on it.*"
- **Mind the budget.** A core file goes to the model on every turn, so a section should stay short. Start with the five symptoms your crew hears most.

A symptom the file does not list changes nothing: the assistant searches the manuals as before.

Parts, so a number the assistant says can be checked

A part number is the one thing on a job report that somebody else acts on: base orders against it. So the app never writes a number down without looking it up first, and a number it cannot find is recorded and spoken as **unverified** rather than quietly trusted.

Give it something to look up. Add a `parts.md` core file — or a `## Parts` section in any core file — with a table whose first two columns are `Part` and `Description`. `Fits` and `Supersedes` are optional, and `Supersedes` is what lets a technician read the old number off a component and get the one to order:

```
## Conversion kits

| Part  | Description                                | Fits                | Supersedes |
|-------|--------------------------------------------|---------------------|------------|
| 65W77 | LP/propane changeover kit, 0–10,000 ft      | 070, 110, 135       |            |
| 20A26 | LP/propane changeover kit, 0–4,500 ft       | 090 models          | 20A11      |
```

Only `parts.md` is read as parts throughout; in any other core file, only the tables under a `## Parts` heading count — so a specifications table that happens to have a "Part" column cannot leak in. Put the file in `files` in the manifest like any other core file, and keep the core inside its 32 KB budget: a full price book belongs in `documents`, not here.

A number that is not in the table is still checked against the manuals as a whole word, and the page it is found on is what the record cites. A vault with no parts table is not broken — every number is simply verified against the manuals alone, and a vault with neither records them unverified, which is the honest answer.

Step 3 · Write the manifest

This is the whole file for the folder above.

```
{
  "id": "acme_rtu",
  "name": "Acme RTU Service",
  "version": "1.0.0",
  "files": ["safety.md", "error_codes.md", "models.md"],
  "documents_dir": "documents",
  "documents": [
    { "file": "RTU-500-service-manual.pdf", "title": "RTU-500 Service Manual", "kind":
"service_manual" },
    { "file": "RTU-500-wiring.pdf",         "title": "RTU-500 Wiring",         "kind": "wiring" },
    { "file": "RTU-700-install-guide.pdf",  "title": "RTU-700 Install Guide",  "kind":
"install_guide" },
    { "file": "RTU-900-service-manual.md",  "title": "RTU-900 Service Manual",  "kind":
"service_manual",
      "source": "RTU-900-service-manual.pdf",
      "source_url": "https://example.com/manuals/RTU-900-service-manual.pdf" }
  ],
  "procedures_dir": "procedures",
  "gating": { "iap": "enterprise" },
  "prompt_rules": [
    "Never fabricate equipment data, fault codes, refrigerant properties, or procedures.",
    "Use only the vault contents, the retrieved manual passages, and the technician's stated
observations.",
    "Cite the source file or manual page on every factual claim.",
    "If the manuals do not cover the question, say so and recommend escalation rather than
guessing.",
    "Remind the technician of the relevant safety steps before any action that opens an electrical
panel, the gas train, or the refrigerant circuit."
  ],
  "source_attribution_format": "Source: {files}",
  "source_attribution_required": true,
  "trades": ["refrigeration_hvac"]
}
```

KEY	WHAT IT DOES
`id`	Lowercase, no spaces. Never change it after the first import; updates match on it.
`name`	What technicians hear: "*start a session in Acme RTU Service*".
`version`	Bump it when you re-import.
`files`	The core files, in the order you want them read.
`documents_dir` / `documents`	The manuals. `title` is what the technician hears in a citation; `kind` is a free-text label.
`source` (per document)	Optional. The manufacturer's own PDF, sitting in the same folder, for a document you imported as extracted text. Copied and hashed at import, never indexed. Tapping a citation can then show the manufacturer's page rather than the transcription of it.
`source_url` (per document)	Optional. Where the manufacturer publishes this manual, as an `https://` link. Offered as a button that opens outside the app; it is never a substitute for bundling the original.

KEY	WHAT IT DOES
`edition` (per document)	Optional. Which revision or edition this is (`"Rev C"`, `"2024"`, `"3rd edition"`) when the vault holds more than one of the same work. Documents of the same `kind` whose titles are identical once the edition is taken out are treated as editions: the newest answers by default, and an older one only when the technician asks for it by name or asks what changed.
`procedures_dir`	Omit it if you have no procedures.
`gating`	Always `{ "iap": "enterprise" }` for a custom vault.
`prompt_rules`	The rules the assistant follows while this vault is active.
`source_attribution_*`	Forces a source line on every factual claim drawn from the core files.
`sites`	Optional. The site documents, one JSON file per site under `sites/` — see **Describing a site system** below. Omit it if you have none; a vault without it imports exactly as before.
`vault_only`	Optional. `true` seals a job on this vault to the vault's own content: no web search and no other outside source — see **Sealing a vault to its own content** below. Omit it to leave things as they are; a vault without it imports exactly as before.
`trades`	Optional. The trades this vault is for: `refrigeration_hvac`, `electrical`. Each trade adds its own tiles to the Field Assist grid on phones whose default vault is this one — `refrigeration_hvac` adds **Read Gauges** and **P-T Lookup**; `electrical` adds no tiles yet. A trade also decides which calculations the assistant offers: the refrigerant ones are refrigeration's and voltage drop is electrical's, so a phone set to one trade only does not offer the other's. The technician's own choice in Settings › Field Assist › Trade, then the organisation's, comes first; the vault's is used when neither has chosen. A value the app does not know is ignored, not refused, so you can list a trade a newer version will understand. Omit it to leave the choice to the technician.

> **The importer refuses a manifest whose rules do not mention fabrication and citation.** Your `prompt_rules` must include a rule containing the word *fabricate* and one containing *cite*. The rules above satisfy that; reword them freely as long as those two words survive.
>
> **The title is the citation.** Whatever you put in `title` is what the technician hears: "*Source: RTU-500 Service Manual, page 42*". Name it the way your people already refer to the book.
>
> **Write titles to be heard, and make every one different.** Say what the document is and which one: product type, document number, revision, as in *RTU-500 Rooftop Unit Service Manual SM-5002 Rev B*. Twelve titles that all read *Owner's Manual* tell the technician nothing. Leave out separators like `|`, which only get in the way when a title is read aloud.
>
> **Two revisions of one document.** If the newer one replaces the older for every model, keep only the newer. If you need both, decide which case you are in. When the newer one supersedes the older, give both the same title and an `edition`, so the newest answers by default. When each revision covers different models (common for kits and accessories), leave `edition` out, keep the revision in each title, and map each model to its own revision in `models.md`. A year in the title counts as an edition label too: two same-kind titles that differ only by year are treated as editions, and the older one drops out of answers unless asked for.

Step 4 · Procedures (optional)

A procedure is a branching checklist the assistant walks a technician through by voice: "*next*", "*go back*", "*repeat that*", and branch choices from what the technician reports. Skip this for your first import; add one once the question-and-answer loop works.

Each procedure is one JSON file in `procedures/`. The importer checks the graph before install: the entry step must exist, every branch must point at a real step, every non-terminal step needs a way forward, and at least one terminal step must be reachable.

```
{
  "id": "rtu500_low_charge",
  "title": "RTU-500 Low Charge Diagnostic",
  "version": "1.0.0",
  "description": "Confirm and correct a ZX9 low-charge fault on the RTU-500.",
  "safety_notes": [
    "Lock out power before opening the control panel.",
    "Recover refrigerant before opening the circuit."
  ],
  "entry_step": "confirm_fault",
  "steps": [
    {
      "id": "confirm_fault",
      "title": "Confirm the fault",
      "instruction": "Read the controller display. Is ZX9 active now, or only in the history
log?",
      "expected_input": "active or history",
      "citations": ["error_codes.md"],
      "branches": [
        { "id": "active",  "condition": "ZX9 is active now",     "next": "check_sight_glass" },
        { "id": "history", "condition": "ZX9 is only in the log", "next": "intermittent" }
      ],
      "default_next": "check_sight_glass"
    },
    {
      "id": "check_sight_glass",
      "title": "Check the sight glass",
      "instruction": "With the unit running for ten minutes, look at the liquid line sight glass.
Is it clear, or bubbling?",
      "expected_input": "clear or bubbling",
      "safety_note": "Keep hands clear of the condenser fan while the panel is open.",
      "citations": ["RTU-500 Service Manual"],
      "branches": [
        { "id": "bubbling", "condition": "Bubbling or foaming", "next": "measure_subcooling" },
        { "id": "clear",    "condition": "Clear",               "next": "sensor_check" }
      ],
      "default_next": "measure_subcooling"
    },
    {
      "id": "measure_subcooling",
      "title": "Measure subcooling",
      "instruction": "Read liquid line pressure and temperature and tell me both. The manual's
target is on page 58.",
      "expected_input": "liquid pressure and temperature",
      "citations": ["RTU-500 Service Manual"],
      "default_next": "done_charge"
    },
    { "id": "sensor_check", "title": "Sensor check", "instruction": "The charge looks right. Check
the low-pressure transducer per page 61 before adding refrigerant.", "terminal": true, "outcome":
"sensor_suspected" },
    { "id": "intermittent", "title": "Intermittent fault", "instruction": "Clear the log, run the
unit for a full cycle, and re-run this check if ZX9 returns.", "terminal": true, "outcome":
"deferred" },
    { "id": "done_charge", "title": "Charge corrected", "instruction": "Adjust charge to the
target subcooling, record before and after readings, and clear the fault.", "terminal": true,
"outcome": "resolved" }
  ]
}
```

Tables and chains (optional)

A table in a manual or a code book you load is read into rows and columns at import, on the phone, from your own copy, and the technician can ask for one value from it: "*what's the rating in Table 99 for QXRT, three conductors, rating group 2?*" The answer comes back with its row, its column, its notes and the page it is printed on, or as the two rows a number falls between. Nothing is interpolated, and the app never picks "the next size up" on its own.

Some answers take several tables and a rule or two together: size a conductor for a load once it is corrected for the temperature and the grouping, or check whether one already installed carries it. The app does not know which rules your code applies, so it does not guess. You write the steps down once, as a **rule chain**, against the table and rule numbers of your own code and edition, and the assistant runs them for the technician one step at a time: it asks for each value in its own words, does every lookup and every sum itself, never goes past a step it could not complete, and reads the whole chain back with the page of every value it used. The read-back goes on the job record like a reading.

Write the chains

Add a core file called `chains.md` and list it in `files` like any other core file. It is not read into every question the way the other core files are; the assistant offers its chains by name when a question is one of them. Each chain is a `##` heading, its name, then an `Answers:` line saying what it answers, then its steps, one per line, in a fenced block (or as a list, one step per item).

STEP	WRITTEN AS	WHAT IT DOES
`ask`	`ask load (A) range 1..400: What load will the circuit carry, in amps?`	Asks the technician for a value, with its unit and the range you accept. An answer outside the range, or one that is not a number, is asked again; it is never taken. A temperature or a length said in another unit the app knows (°F for °C, feet for metres) is converted, and the read-back says what was said. `(text)` asks for words instead, such as a size as the table prints it, and takes no range.
`lookup` … `row`	`lookup factor = Table 98 row {ambient} column Rating group {group} › {band} rule at_least`	Reads one cell: the table by its number as printed, the row by its heading or a number, and the column by its heading, or a path of headings joined with `›`.
`lookup` … `cell`	`lookup size = Table 99 column Rating group {group} cell {corrected} rule at_least`	Searches one column for a number and takes the row whose cell meets it, which is how a table of ratings gives a size.
`calc`	`calc corrected (A) = load / factor`	Arithmetic over earlier values: `+ − ∗ /`, brackets and `sqrt( )`, with the unit of the result.
`convert`	`convert metres = length to m`	Converts an earlier value, which has to carry its unit, to another unit.

STEP	WRITTEN AS	WHAT IT DOES
`check`	`check capacity >= load ? "carries it" : "does not carry it"`	Compares two values (`<` , `<=` , `>` , `>=` , `==` , `!=`) and says one of two sentences. The verdict goes in the read-back.
`cite`	`cite Rule QR-14`	The rule the step above follows, written as your book prints it. The app does not read or check it. It carries the text into the read-back.
`say`	`say "Use {size.row}, rated {size} A."`	The result, as a sentence.

Which row. Every lookup ends with its rule: `exact` takes the row whose heading is the number, or a printed range that holds it (5 is in "4-6"); `at_least` takes that row, else the first row above the number; `at_most` takes that row, else the last row below it. In the `cell` form, `at_least` is the smallest cell at or above the number and `at_most` the largest at or below it. A number between two rows under `exact` , a row or column the table does not have, a cell with nothing printed, or a table whose page could not be read as a grid stops the chain at that step. The technician is told why and offered the page.

Values. Every step that makes a value names it, and later steps use it by that name, in braces inside headings and sentences: `{load}` . A value from a lookup also has `{size.row}` , `{size.column}` , `{size.table}` , `{size.page}` and `{size.next}` , the row printed after it, for "the next size up is …". Any value has `{name.unit}` . A name is used only after the step that sets it, and set only once. Name the table itself as printed (`Table 98`), not from a value, so the import can check it is there.

At import. Every step is checked. A step that does not read stops the import with its line, as a procedure that does not decode does. A chain that names a table none of the vault's manuals prints is kept and reported as a warning. `rule_chain list` marks it as needing that table, and it stops at that step until a manual that prints it is loaded. A table whose caption was found but whose grid could not be read counts as present: the chain stops on it and shows its page.

A worked example

These three chains are the ones the app's own tests run, over invented tables: Table 90 says which band a number of conductors falls in, Table 98 gives correction factors by ambient band and rating group, Table 99 gives conductor ratings by rating group, and Table 97 gives a drop per size. Every table, rule number and value in them is made up. Write yours against your own book.

````
# Rule chains

Invented for the example: every table, rule and value here is made up.

## Size a conductor for a load
Answers: which conductor carries a load once it is corrected for the ambient temperature and the
conductors grouped in the run.

```chain
ask load (A) range 1..400: What load will the circuit carry, in amps?
ask ambient (°C) range 0..40: What is the ambient temperature along the run?
ask conductors (conductors) range 1..9: How many current-carrying conductors share the run?
ask group range 1..3: Which rating group is the installation method in?
lookup band = Table 90 row {conductors} column Band rule exact
cite Rule QR-12
lookup factor = Table 98 row {ambient} column Rating group {group} › {band} rule at_least
cite Rule QR-14
calc corrected (A) = load / factor
lookup size = Table 99 column Rating group {group} cell {corrected} rule at_least
cite Rule QR-17
say "Use {size.row}: rated {size} A in rating group {group}, against {corrected} A once the load
is divided by the factor of {factor} ({factor.table}, {factor.page})."
```

## Check an installed conductor
Answers: whether a conductor already installed carries a load, after the same correction.

```chain
ask size (text): Which conductor is installed, by its designation and number of conductors?
ask load (A) range 1..400: What load does the circuit carry, in amps?
ask ambient (°C) range 0..40: What is the ambient temperature along the run?
ask conductors (conductors) range 1..9: How many current-carrying conductors share the run?
ask group range 1..3: Which rating group is the installation method in?
lookup band = Table 90 row {conductors} column Band rule exact
cite Rule QR-12
lookup factor = Table 98 row {ambient} column Rating group {group} › {band} rule at_least
cite Rule QR-14
lookup rating = Table 99 row {size} column Rating group {group} rule exact
cite Rule QR-17
calc capacity (A) = rating * factor
check capacity >= load ? "{rating.row} carries it: {capacity} A once corrected, against a load of
{load} A." : "{rating.row} does not carry it: {capacity} A once corrected, against a load of
{load} A. The next row is {rating.next}."
```

## Voltage drop on a run
Answers: the voltage drop along a run as a share of the supply, against the limit this example
sets.

```chain
ask size (text): Which size is the run, as the table prints it?
ask current (A) range 1..400: What current does the run carry, in amps?
ask length (ft) range 1..2000: How long is the run, one way, in feet?
convert metres = length to m
lookup drop = Table 97 row {size} column Drop rule exact
cite Rule QR-30
calc volts (V) = drop * current * metres / 100
ask supply (V) range 50..1000: What is the supply voltage?
calc percent (%) = volts / supply * 100
check percent <= 4.5 ? "The drop is {volts} V, {percent} % of {supply} V: within the 4.5 % this
````
````

```
example allows." : "The drop is {volts} V, {percent} % of {supply} V: over the 4.5 % this example
allows, so a larger size or a shorter run is needed."
```

Asked to size a conductor for 32 A at 14 °C with five conductors in the run and rating group 2, the first chain reads back:

```
Chain: Size a conductor for a load
1. load: 32 A, as the technician said
2. ambient: 14 °C, as the technician said
3. conductors: 5 conductors, as the technician said
4. group: 2, as the technician said
5. band: Table 90 (Grouping bands), row 4–6, column Band: Band B — page 5 of Example Book (row
asked for: 5; rule exact) — follows Rule QR-12
6. factor: Table 98 (Correction factors), row 13–15, column Rating group 2 › Band B: 0.86 — page 3
of Example Book (row asked for: 14; rule at_least) — follows Rule QR-14
7. corrected = load / factor = 32 / 0.86 = 37.21 A
8. size: Table 99 (Conductor ratings), row QMX90 › two conductors, column Rating group 2: 45 —
page 12 of Example Book (the row whose cell is at least 37.209302; rule at_least) — follows Rule
QR-17
9. said: Use QMX90 › two conductors: rated 45 A in rating group 2, against 37.21 A …
Result: Use QMX90 › two conductors: rated 45 A in rating group 2, …
```

Each page in it opens from the answer's source chips, with the row marked where the page lets it be found.

Voltage drop, in a chain and on its own

Voltage drop is arithmetic, so a chain writes it out in a `calc` step over named values: the current and the one-way length as the technician says them, and the conductor's resistance per unit length, or a factor and the conductor's area, as your own book's table prints them. Nothing in the app supplies a resistance, a factor or a limit; the chain looks it up or asks for it.

```
single-phase run         calc volts (V) = 2 * current * length * resistance
three-phase run          calc volts (V) = sqrt(3) * current * length * resistance
a factor over the area   calc volts (V) = 2 * factor * current * length / area
as a share of the supply calc percent (%) = volts / supply * 100
```

Keep the length and the resistance in the same length unit: with the length in metres and a resistance per km, divide by 1000 (`2 * current * length * resistance / 1000`); the same for feet and a resistance per 1000 ft. `sqrt(3)` is for a three-phase run measured line to line. A factor and its area must be in matching units too: Ω·mm²/m over mm², or Ω·cmil/ft over circular mils. The resistance usually comes from a lookup just before:

```
lookup resistance = Table 96 row {size} column Resistance rule exact
cite Rule QR-30
calc volts (V) = 2 * current * length * resistance / 1000
```

The same calculation without a chain is the assistant's voltage-drop calculator: "*what's the drop on a 30 m single-phase run at 20 A?*" It needs the phase, the current, the one-way length in metres or feet, and the

conductor's resistance (per km, m, 1000 ft or ft) or a factor with the conductor's area, and it asks for whichever is missing rather than assuming one. It answers with the drop in volts and, given the supply voltage, as a percentage of it, says every input back, and says that it counts resistance only, not reactance or power factor. It is the same arithmetic as the `calc` lines above, so a chain and the calculator give the same number for the same inputs.

The electrical starter

The app ships an **Electrical Service** vault, offered beside the refrigeration and IT vaults in Settings › Field Assist. It holds a `safety.md` (isolate, lock off, prove dead, never guess a rating), an `error_codes.md` and a `parts.md` with nothing in them yet, and a `chains.md` holding the step vocabulary and the worked example above, renamed "Example: …" and commented, so the first chain file an electrician opens already parses. It declares the `electrical` trade. It ships no standard's text, no table and no rule, and it lists its example chains as needing Tables 90, 98, 99 and 97 until a book that prints tables with those labels is loaded.

To make it your organisation's vault with your own code books, start from its folder in the app's public repository, which also holds a `manifest.json` and a `documents/` folder with a note:

```
git clone --depth 1 https://github.com/straff2002/OpenGlasses.git
cp -R OpenGlasses/OpenGlasses/Sources/Resources/Vaults/electrical my-electrical-vault
```

Give the manifest your own `id` and `name`, put your licensed code PDFs in `documents/` and list each under `documents` with the title your electricians use for it, replace the example chains with your own, and import the folder the usual way. Every table a chain reads is extracted on the phone from your copy, and none of it leaves the phone.

Check the chains before the vault travels. On a Mac, `Scripts/chains-validate/run.sh my-electrical-vault` reads your PDFs' tables as the import does and prints, by line, every step that does not read and every lookup whose table, row or column is not in the tables it found, and whether the manifest lists `chains.md`. Its README explains each line. It prints names only from your chains file, never from the book, so its output can go to whoever writes the chains.

When no chain answers: a proposed chain

The technician does not need to know which rules apply. For a question about a table or a rule that none of your chains answers, the assistant first looks up the clauses in your loaded books, then proposes a chain in the same syntax, built only from what those passages say: a `cite` line on every lookup and every check, and the passage each lookup came from. The app checks the proposal before the technician hears it, and refuses one that cites nothing, a lookup or check with no `cite`, a lookup whose passage was not retrieved on the job, or a rule the retrieved passages do not contain, which is a rule the assistant brought from memory. A lookup naming a table your manuals do not print is kept and marked "needs Table N", as an authored chain's is.

The technician hears the proposal read back step by step, each step with its rule and its passage, marked "proposed, unreviewed". Then they say go, change a step ("ask me the ambient instead"), or stop.

Nothing runs before go. A proposed chain runs through the same runner as yours, and its read-back goes on the job record marked "proposed, unreviewed", with the passages it was built from.

Keeping one, on the phone. A proposed chain that ran to a result can be kept: "keep that chain". It is saved on that phone in the vault's own folder, beside your `chains.md`, marked "unreviewed — kept on this phone". It is listed on that phone only and runs like any other chain. It is not one of the vault's files, so it never leaves in an export, a pack or a link, and a new version of the vault leaves it alone.

Getting it to your organisation. "Share the kept chains" puts them on the share sheet as a `chains.md` file. Each chain carries a line saying when and on which job it was kept, and the passage each lookup was read from. Check every step against your own book, paste the chains you want into your vault's `chains.md`, and push the vault the usual way. None of this needs the office app. If your organisation has the office app, keeping a chain also files the same file for review with the next team-learning bundle.

```
## Size a conductor, ambient assumed
Answers: which conductor carries a load at an assumed ambient, once corrected for grouping.
Unreviewed — kept on a phone on 2026-10-20, worked out on job 1042.
Read from: band from Example Book, page 6, §QR-12; factor from Example Book, page 7, §QR-14; size
from Example Book, page 8, §QR-17.
```

The lines under the heading are prose to `chains.md`, so the file imports as it is. **A reviewed chain replaces the kept one:** once the vault arrives with a chain of the same name in `chains.md`, the phone offers yours and no longer offers its kept copy.

A table printed as a picture: checking it on the phone

Some books print a table as a picture: the caption is text and the grid under it is not. The phone reads such a table from the picture at import, on the phone, and every value it gives from one says so: "read from a picture of the table on page 31 of …". A cell the picture did not show clearly is never given; the technician is offered the page instead. Until someone checks the table against the page, each value is also read back for the technician to check.

Checking a table. Open **Settings › Custom Vaults**. Under each manual, a table read from a picture shows **Read from a picture: not checked**, **checked** or **checks disagree**. Tap it to see the picture of the page above the grid the phone read. A cell it could not read clearly, or one checked to two different values, is marked with a triangle. Tap a cell and type what the page prints, or leave what was read and save it to confirm it. A row's name or a column's heading the phone was unsure of can be confirmed, not corrected: if the phone misread it, its values are not given and the page is shown instead. **Mark table checked** confirms every cell that was read clearly, and leaves the marked ones for you to look at one by one. Checking is for whoever can import vaults on that phone.

A checked value is still said to be read from a picture, with who checked it and when, and is no longer handed back to the technician to check. A rule chain step that read a checked cell is marked the same way. Every check names who made it (the name in **Settings › Field Assist**, or the phone's name) and when.

Where the checks are kept. In a vault you imported on the phone, the checks are written into the vault's own `table-checks.json` beside `chains.md`. In a vault that came from your organisation's office, a link

or a signed pack, the phone never writes into the vault: its checks are kept on the phone beside it and used together with the vault's own file.

Getting them to another phone. In the check view, **Share checks** puts the checks of that table's manual on the share sheet as `table-checks.json`. Put the file in the vault's folder (at the top, beside `manifest.json`) on the other phone and import the folder again: its tables are checked there too. No office is needed. An organisation with the office can write the same file into the vault it sends, and the phone reads it the same way.

When two checks disagree. A cell checked on one phone to one value and in a file to another is not given at all: it shows **checks disagree** with both values and who gave each, until someone checks it again on the phone to the right value, or the file is corrected. Neither value wins on its own.

A new copy of the book starts unchecked. Checks are kept against the PDF they were made on. Import a different PDF and its tables start unchecked; the old checks stay in the file and no longer apply. Removing a manual removes its checks. Removing the vault removes all of them.

Requiring checks. An organisation profile can require every table read from a picture to be checked before any of its values are given (`organizationRequiresTableChecks`). Until a table is checked, a lookup on it gives no value and says it has not been checked yet, and a chain stops on it. Without the profile, checking is optional.

`table-checks.json`

```
{
  "version": 1,
  "checks": [
    {
      "document_sha256": "<the SHA-256 of the PDF, 64 lower-case hex characters>",
      "table": "Table Q7",
      "row": ["QMX90"],
      "column": ["Spacing (mm)"],
      "kind": "corrected",
      "value": "35",
      "checked_by": "the vault manager",
      "checked_at": "2026-10-21T09:30:00Z",
      "origin": "phone"
    }
  ]
}
```

FIELD	WHAT IT HOLDS
`version`	`1`. A file with another version is not read.
`document_sha256`	The SHA-256 of the PDF the table was read from: the manual itself when it is a PDF, otherwise the original PDF it was made from. For a table read from a photographed manual's pages it is the SHA-256 of that capture revision's manifest (a retake starts unchecked), and those checks are kept on the phone, never in a vault's file.
`table`	The table's label as printed.

FIELD	WHAT IT HOLDS
`row`, `column`	The row's and the column's headings as the phone read them, outermost first, as a list. A cell's check has both; a check of a row's name has `row` only; a check of a column's heading has `column` only. A cell is never named by its position.
`kind`	`confirmed` (the page prints what was read) or `corrected` (the page prints `value`). A row's name or a column's heading can only be `confirmed`.
`value`	What the page prints in the cell, footnote mark included. Empty when the page prints nothing there.
`checked_by`, `checked_at`	Who checked, and when (ISO 8601).
`origin`	`phone` for a check made on a phone; left out by anyone else.

An entry that does not read is skipped with a warning, and the rest of the file is kept. A check for a PDF the vault does not hold is kept, with a warning, and applies if that PDF is loaded.

Describing a site system (optional)

A manual describes one unit on its own. Nobody writes down how the units at one site work together, and that is where the hard faults live: the heat pump shows a communication fault because the furnace board dropped off the bus, and a technician who reads only the heat pump's manual replaces the wrong board. A **site document** is the overlay that joins them: which units are at the site, how they are connected, which controller commands which, the systems outside the site they talk to, the settings someone set on site, and the faults your crew has seen come from the link between two of them.

It is for any trade. The same shape describes a furnace, heat pump and communicating thermostat; a plant line with its PLC, drives and fieldbus; a comms room with its switches, UPS and management network; a building with its chiller, air handlers and building management system; a dairy shed or a solar installation. The kinds of relationship are a small shared vocabulary, and your trade's own words go in the free text beside them.

When to write one. When units share controls, a bus, a line set, a pipe run, a network or an outside system, and a fault on one can come from another. Write one for a customer's own site, with its serials, or for a reusable system type ("our standard dual-fuel install", "the standard three-rack comms room") with no serials at all.

How it is used. A site document is never loaded into every question, and never searched like a manual. When a job is at a documented site, the phone gives the assistant that one site's connections and controls, tells it to consider the link when a fault crosses units, and has it cite the site document as your vault's, never as a manual page. The brief says "Part of the Hillside house system, with 2 other units", and recognising one of its units says so once. A job is matched to a site by:

- **a serial** on the job, or one the technician reads out on site — this confirms the site;

- **the other system's identifier** for a unit or for the location, when the job file came from your job-management system — this confirms it too;
- **the models alone**, when exactly one site lists every model the job names — this is said as unconfirmed, "matched by model only; the serial will confirm it". When several sites fit, nothing is assumed and the brief says how many.

Each unit stays itself. The site document refers to units by the model their own section already uses; their specifications stay in that section and in their manuals, and the job's record keeps each unit apart as always.

Write it

One file per site, in `sites/`, named for the site's `id`, and listed in the manifest:

```
"sites": ["sites/hillside-house.json"]
```

This is the whole of `sites/hillside-house.json` for a three-unit dual-fuel system. Every model, serial, place and identifier in it is invented.

```
{
  "format": "avenkin.site-system",
  "version": 1,
  "id": "hillside-house",
  "name": "Hillside house, whole-home system",
  "location_label": "Hillside house",
  "external_location_ids": [ { "system": "example-fsm", "id": "loc-88412" } ],
  "units": [
    { "id": "furnace", "model": "NV80F060", "serial": "AF80-2201", "label": "basement furnace" },
    { "id": "heatpump", "model": "NV36HP2", "serial": "AH36-7713", "label": "side-yard heat pump",
      "external_ids": [ { "system": "example-fsm", "id": "asset-5521" } ] },
    { "id": "tstat", "model": "NT9C", "label": "hall thermostat", "covered": false }
  ],
  "connections": [
    { "from": "furnace", "to": "heatpump", "kind": "fluid", "medium": "refrigerant",
      "detail": "3/8 and 3/4 line set, 9 m" },
    { "from": "tstat", "to": "furnace", "kind": "communication", "protocol": "4-wire serial bus",
      "detail": "terminals 1-2-R-C" },
    { "from": "furnace", "to": "heatpump", "kind": "communication", "protocol": "4-wire serial
bus",
      "detail": "daisy-chained from the furnace board" }
  ],
  "controls": [
    { "controller": "tstat", "commands": ["furnace", "heatpump"], "mode": "communicating",
      "detail": "Dual fuel; balance point at the thermostat" }
  ],
  "integrations": [
    { "id": "portal", "name": "Remote-monitoring portal", "kind": "cloud_portal",
      "units": ["tstat"], "direction": "outbound", "protocol": "Wi-Fi",
      "detail": "Alerts reach the dealer account; no control from it." }
  ],
  "settings": [
    { "unit": "furnace", "name": "Blower profile", "value": "B", "note": "Set at install" }
  ],
  "interaction_faults": [
    { "seen_on": "heatpump", "code": "E77", "link": "communication",
      "text": "E77 on the heat pump when the furnace board drops off the bus; check the bus at the
furnace first." }
  ],
  "notes": "Installed 2024; disconnect behind the side gate."
}
```

KEY	WHAT IT SAYS
`format`, `version`	Always `"avenkin.site-system"` and `1`.
`id`	Letters, digits, `.`, `-` or `_`; unique in the vault; the file is `sites/<id>.json`.
`name`	What the assistant cites: "*Source: Hillside house, whole-home system*".
`location_label`	Optional. What the technician hears the site called: "*Part of the Hillside house system*".

KEY	WHAT IT SAYS
`external_location_ids`	Optional. The location's id in another system, each as `{ "system": …, "id": … }`. A job file from that system with the same `location_id` confirms the site.
`units`	One to twelve. `id` is a short lowercase handle used only inside this file. `model` is the spelling the unit's own vault section uses. `serial` and `label` (where it is at the site) are optional. A unit your vault has no section for — a thermostat, a third-party controller — says `"covered": false`, so the importer can tell it from a typo. `external_ids` is optional: the unit's id in another system, as `{ "system": …, "id": … }`, one per system.
`connections`	Optional, up to thirty. `from` and `to` name units or integrations; `kind` is one of `power`, `control`, `communication`, `data`, `fluid`, `air`, `mechanical`, `other`; `medium`, `protocol` and `detail` are your trade's words; `direction` is `one_way` (from → to) or `two_way` (the default).
`controls`	Optional. Which unit (`controller`) commands which (`commands`), with `mode` `communicating`, `conventional` or `other`.
`integrations`	Optional, up to eight. The systems outside the site the units talk to: `kind` is `bms`, `scada`, `cloud_portal`, `network_management`, `business_application`, `metering` or `other`; `units` it touches; `direction` `inbound`, `outbound` or `both`; and in `detail`, what it reads, what it may command, and who holds the login (a role, never the credentials).
`settings`	Optional, up to thirty. A value someone set on site, per unit.
`interaction_faults`	Optional, up to twenty. A fault `seen_on` one unit that comes from somewhere else: `link` names the unit, the integration or the kind of connection it crosses, and `text` says it in plain words.
`notes`	Optional. Anything else a technician arriving should know.

What to write: the units by the model spelling their section already uses; serials when the vault is for one customer's sites, none for a reusable system type; the connections a technician would trace, typed generically with the trade's words in the detail; the controller; the outside systems and who holds their logins; the settings someone set on site; and the faults the crew has actually seen come from the link.

What not to write: specifications (they belong to the model's section), customer contact details, passwords, links, or a diagnosis nobody has seen.

The importer checks it. Every text is plain — no markup, no line breaks, no links — and within its length. It refuses a unit whose model the vault has no section for (unless it says `"covered": false`), a unit that names another vault (a site document names only units in its own vault or on the job), repeated unit ids, a serial given to two units in the vault, a location or a unit identifier given to two sites, a connection, control, setting or fault that names a unit nobody declared, an unknown kind or mode, a member it does not know, a file over 8 KB or more than a hundred sites. A site whose description is longer than the assistant's site block (1,600 characters) still imports, with a warning naming what the block will leave out; the assistant can still ask for the whole document.

Packs and exports. A pack describes products, not customers' sites, so a pack that carries `sites` is refused. A vault you export from the phone carries its site documents, which can identify a customer's premises — keep that in mind before you send one on. Office-built vaults carry site documents only once the office's vault check accepts them.

Sealing a vault to its own content (optional)

Some content must not be mixed with anything else: a licence forbids it, or a customer wants answers from their documents and nowhere else. A rule in `prompt_rules` asks the assistant not to look elsewhere. `vault_only` makes the app stop it. It goes in the manifest:

```
"vault_only": true
```

What it shuts, while a job on this vault is open:

- web search, news and loading a page from a link or a QR code. They are not offered to the assistant, not listed in its instructions, and refused if anything asks for them: a name it remembers, a goal step, a skill pack;
- every tool on a connected MCP server, and the OpenClaw gateway;
- the assistant's automatic web re-check of an answer it was unsure of;
- custom tools and skill-pack tools, and any tool not on the app's list of what a job may keep.

What a job keeps: the job's own tools (manuals, tables, procedures, readings, photos, the record, an expert) and the phone's own that do not answer a technical question (timers, weather, directions, calls, notes).

What it does not do. It stops outside *sources*, not the model provider. Every turn still goes to the cloud model the phone has chosen, with the passages it was given, as it does on any other job. To keep turns on the phone, choose an on-device model. And it cannot unteach a model what it already knows, so keep your rules about citing and about saying "the manuals do not cover this".

It needs a recent build. The app reads `vault_only` from build 611. An older build ignores the key and imports the rest of the vault, so the job runs unsealed and nothing says so. A sealed vault must state its minimum build, wherever you hand the vault over, and you should check the build on each phone before relying on it. It travels with the vault: an export keeps it, and so does removing a manual on the phone. Set it to something other than `true` or `false` and the importer refuses the manifest, because a seal the author asked for must not quietly be missing.

Check it. The job's support report says `Vault-only: on` on the line under the job's heading. A sealed job has no web search in its trace; if something did reach for a shut tool, the trace shows that tool as `(refused)` and nothing ran.

Step 5 · Import on the iPhone

1. Copy the whole folder into iCloud Drive (or AirDrop it to the phone and save it in Files). Do not zip it; the importer takes a folder.
2. Open **Settings › Custom Vaults** and tap **Import Vault Folder…**, then pick the folder.
3. The app validates first. If anything is wrong it names the file and the problem, and nothing is installed. Fix it on the computer and try again.
4. On success the vault appears in the list and indexing starts immediately. Each manual shows a progress bar and then a section count, for example *RTU-500 Service Manual · 412 sections*. Leave the app in the foreground until every manual has a count.
5. Warnings are advisory. The one you are most likely to see says the core files exceed the budget; move long material into `documents` and re-import.

Step 6 · Test the loop

1. In **Settings › Field Assist**, choose your vault as the default.
2. Say "*Start a session in Acme RTU Service.*"
3. **Read the nameplate first.** With the plate in view say "*Look up this nameplate*", or simply say the model. The app reads the plate on the phone, matches the model against your core files and then against the manuals, and — when it resolves to exactly one of the models named in your `##` headings — remembers it for the rest of the session. The assistant says which machine it settled on, and the **Active Session** card on the Field Assist screen shows the model; tap it for the heading it matched, whether it came from the plate, your voice or the work order, and the time. If the read was wrong, correct it by voice — "*no, it's the 070*" — or tap **Change** on that card and pick from your own model list; **Clear** forgets it. When the glasses have a display, the model appears there too, on one line, each time it changes.

From then on the session answers for that machine: a table row printed for a different model of the same family ranks below the row that covers yours. This is also what makes the next test work, so do this one first.

Say each model that has a service manual in the vault, one at a time. Each should settle straight away and show on the **Active Session** card. If one doesn't, or the assistant asks which one you mean when the model has only one section, its number is in a second heading somewhere in your core files (see **Headings define lookup sections**); move it into that section's first line. 4. Ask about a code you know is in the manual: "*What does fault ZX9 mean on the RTU-500?*" The answer should end with a source line naming the manual and page. 5. Ask about a machine you do not have — "*how do I replace the heat exchanger on a Carrier 58MVB*". The answer should name the models your vault does cover and refuse: "*The loaded manuals cover Acme RTU Service models RTU-500, RTU-750 and 2 more; 58MVB is not one of them. Confirm the nameplate or load its manual.*" This works only because your model headings list every spelling of every model (Step 1); a vault whose headings name no models never refuses on equipment, and behaves exactly as it did before. 6. Ask something the manuals do not cover, for example

a torque spec that is not in the book. The correct answer is that the loaded manuals do not cover it and to escalate.

The check that produces that answer is measured, not assumed, and it is worth knowing what it can and cannot do. Against a real pair of furnace manuals it refused three out of four out-of-scope questions — a torque figure the book never gives, another manufacturer's efficiency rating, a price, a warranty procedure. What word overlap alone does **not** refuse is a question about a subject your manuals genuinely cover but a machine they do not: ask about the heat exchanger on someone else's furnace and passages about heat exchangers come back, because they are about heat exchangers. That is what the equipment check above is for, and with model headings in place the same pair refused nine out of ten. A question that names no machine at all — "*how do I charge the refrigerant on the outdoor unit*" — is still outside its reach; there the vault's prompt rules and the page citation on every passage are what keep the answer honest, which is why the rules about fabricating and citing are required. The same check occasionally refuses a fair question whose wording shares little with the book's; rephrase it in the manual's own words and ask again.

If you get a confident answer where the book is silent, tell us; that is the behaviour the vault rules exist to prevent. 7. **Tap the citation under the answer.** Every source an answer names becomes a chip; tapping one opens that page. The header tells you which document you are looking at — *Manufacturer's document · page 42 of 118 · unmodified since import*, or *Extracted text · page 42* with a line saying whether the original is bundled — and you can swipe through the manual and tap once to come back to the cited page. A chip naming a core file opens that file at the section it cites, ready to correct. If a chip opens a page that does not answer the question, the manual's page numbering is off by a page or two; see Step 1. 8. **Ask for the job so far.** Say "*read back the job*" — or tap **Read back the job** on the Active Session card — and the assistant speaks the work record: the machine, each task with its status and evidence, the parts requested, the pages verified against the manufacturer's document, and the time on site. It is assembled from what was recorded and nothing else; no model writes it, so two identical visits read identically. This is what the technician confirms before anything is sent. 9. End the session. The audit log records every question, answer, photo and citation — including which citations were opened and which document each was read in — and the machine the session identified, printed at the top of the exported work order (*Equipment: RTU-500 — from the nameplate at 14:02*). A team licence can export the record as PDF.

> **Everything stays on the phone.** Manuals are indexed and searched on the device and never uploaded to us. The passages relevant to a question are sent to whichever AI model you configured, together with the question, in the same way the rest of the app works. If you use an on-device model, nothing leaves the phone at all.

Step 7 · Send the job report

At the end of a visit there is one record: what was recommended, what the technician decided, what was done and on what evidence, and what base is being asked for. It leaves in three shapes off the same data — a PDF a person reads, a JSON file a job system parses, and a short summary that fits in a message — so they cannot disagree with each other.

What your organisation receives from the phones, what it never receives, and what it owes when staff use the assistant are set out in the deployer information sheet.

Set the destinations once

In **Settings › Field Assist › Job Reports**:

- **Email / Messages recipients** — where a report goes when nobody names anybody. Separate several with commas. A technician can always override by naming a contact out loud.
- **Office endpoint** (optional) — an HTTPS URL your own system answers on. With one set, job records and parts requests are posted there through the offline queue with no tap at all, retried until they arrive. The optional bearer token is kept in the device Keychain and never in preferences, never in an export.
- **Allowed channels** — which routes a report may leave by: Email, Messages, WhatsApp, Telegram, Share…, Endpoint. **What may carry site data is the organisation's call, not each technician's.** Switch a channel off and the assistant refuses it out loud and names the ones that are allowed. The endpoint becomes selectable once one is configured.

Sending it

Say "*send the job report to base*", "*email this to the office*", "*message the job to Dave*" or "*share the report*" — or tap **Send report…** on the Active Session card. The composer opens on the phone with everything filled in, and **the technician taps Send**. iOS does not let an app send mail or a message on someone's behalf, and that requirement is the point rather than an obstacle: nothing leaves the device without either a person's tap or the endpoint the organisation configured.

What each channel carries:

CHANNEL	BODY	ATTACHMENTS
Email	The full work record	Work order PDF + JSON record
Messages	Short summary and the job reference	The PDF, when the device can attach one; the body says so when it cannot
WhatsApp / Telegram	Short summary and the job reference	None — these open by URL scheme and cannot carry a file
Share…	The full record	Both files, to anywhere the share sheet reaches
Endpoint	The JSON record, posted	—

Recipients need an address the channel can use: a contact name resolves to a phone number for the message channels, and an email needs an actual address — the assistant says so rather than guessing one, because the wrong inbox is the one mistake here that nobody would notice.

Nothing is silently lost

A composer you dismiss sends nothing, and nothing pretends otherwise: the record stays in the queue, any parts requests stay *requested*, and the Active Session card says **Unsent**. Open **Settings › Field**

Assist › Field Sync to see every job report and parts request the office has not received, listed by job reference with what is in it. Each row has **Retry**, which puts it back through the queue, and a work record also offers **Send by email instead**, which opens the composer with the record in the body. WhatsApp and Telegram cannot tell the app whether you tapped Send in them, so a report handed to one of those is recorded as not confirmed and stays queued — deliberately.

Sending the technician their next job

An office can send a job before the visit as a small file — `.ogjob` — attached to an email. On the phone the technician taps the attachment, chooses **Open with Avenkin**, reads what it says on a review sheet, and taps **Add to upcoming jobs**. Nothing is added without that tap, and a job file cannot start a job, change equipment, create a task or send anything: it only proposes a job ahead, which then appears under **Upcoming** on the Job tab and on the car screen, where it can be briefed, navigated to, and started on site.

The format, in one line: a JSON object with `"format": "openglasses.job"`, `"format_version": 1`, and any of `job_reference`, `site` { `customer`, `address`, `contact` }, `fault_report` (the words the customer gave, verbatim), `equipment` [{ `model`, `serial` }], `scheduled_for` (ISO 8601), `notes`, `attachments` [{ `name`, `reference` }] (named, never embedded) and `issued_by`, plus an optional `signature`. Plain text only, 64 KB at most; a field the app does not know is refused rather than ignored.

Make one with `swift Scripts/make-job-file.swift make job.json 1007.ogjob --key-file office.key`, which checks the app's limits before writing. Leave out `--key-file` to write it unsigned — the phone will say *Not signed — check it came from your office*.

Signing is your organisation's, not ours. Mint a key once with `swift Scripts/make-job-file.swift keygen office.key`; it prints the public half, which goes into your organisation's profile on the phone. A job file signed with it shows *Signed by* your organisation's name; one signed with any other key, or altered after signing, is refused. In medical mode, or when your profile requires it, an unsigned job file is refused outright.

Receive only. The app never emails a job file out, and there is no link form — a link in an email that opens a job is the shape of a phishing message.

Updating a vault

Edit the folder on the computer, bump `version`, and import again with the same `id`. Manuals whose content has not changed are skipped; a changed manual is re-indexed and its old index removed; a manual you dropped from the manifest is removed. Technicians' in-app edits to the core files are kept and layered over your new originals. To pull a vault off the phone with those edits included, swipe the vault row and choose **Export**.

An export does not contain the manuals. It carries the manifest, the core markdown with the technicians' edits, and the procedures — never the manual text, never the manufacturer's PDFs. The

manuals stay on the phone they were indexed on. The exported manifest still *lists* every manual the vault needs and marks them as not included, so importing that folder on another phone stops and names the files to supply: put the manuals back into the folder's `documents/` directory, from your own copy on the computer, and the import goes through. What an export is for is the vault's own content and the technicians' edits; what carries the manuals is the folder you built, which is why you keep it.

Removing one manual on the phone

A manual gets superseded, or it turns out to be the wrong revision, and the technician holding the phone is not the person with the vault folder on a computer. Open **Settings › Custom Vaults**, find the manual under its vault, and tap **Remove manual…**.

The confirmation names the manual and the vault and says what it does: *Remove this manual from this device's vault and search index? Existing conversations and core reference files are unchanged. Importing a vault containing this manual again can restore it.* Tap **Remove manual** and the row shows **Removing…** until it is done; the rest of the vault keeps working throughout, and the vault itself stays installed even if you remove the last manual in it.

Three things are worth being precise about, because they are three different kinds of memory:

- **Manual retrieval** — the passages the assistant searches to answer a question. This is what the removal takes away, permanently and on this device: after it finishes nothing can retrieve that manual, including after a restart and after tapping **Re-index manuals**.
- **Core reference files** — your markdown (`error_codes.md`, `models.md`, safety rules). These are loaded whole into every prompt and are **untouched**. If a fact was copied out of the manual into a core file, it is still there; edit the core file if you want it gone.
- **Conversation history** — the questions and answers already on screen, and the session records already written. These are **not rewritten**, and neither is what was already sent to an AI model. An older answer can still quote the manual you removed, and tapping its citation now says *Manual removed from this vault* instead of opening the page.

So **start a fresh conversation for a clean test.** Ask the same question again in a new conversation: the assistant should answer from the manuals that are left, or say the loaded manuals do not cover it. Do not treat an old answer still quoting the removed manual as a failure — that answer was given while the manual was loaded.

A job in progress is not interrupted. The session keeps its job reference, the machine it identified, its tasks and parts and its audit log; only the manual's own material drops out from that moment, and the record notes that a manual was removed part-way through the visit.

If the removal reports a failure, the manual stays unavailable and the row offers **Retry removal** — a removal that stopped part-way is finished by trying again, or at the next launch. If it says the vault's manual index needs repairing, import the vault's folder again; that rebuilds the index and the removal can then proceed.

Restoring one. There is no undo, and nothing keeps a list of what was removed. Importing a vault folder that contains the manual puts it back — an explicit import is always authoritative. What an export cannot do is bring it back: an export of the reduced vault no longer lists the manual at all, and it does not carry any manual's file either. Keep the original folder on a computer.

Removing a manual is only offered for vaults you imported yourself. A signed pack's manuals belong to its publisher; remove the whole pack instead. Removing installed content does **not** need a current subscription or licence — importing manuals does, but taking a superseded one off a phone should not stop working when a subscription lapses. The same line holds for the vaults already on the phone: a lapsed subscription keeps them readable and removable, and only stops new manuals going in.

If the import refuses

MESSAGE	FIX
`manifest.json is missing or unreadable`	The file must sit at the top of the folder with that exact name.
`listed file missing: models.md`	A name in `files` does not match a file. Case and extension matter.
`listed document missing: documents/…`	Same for a manual; check `documents_dir` and the file name.
`manual not included in this export: … — add documents/… to the folder before importing`	The folder came off a phone, and exports never carry manuals. Copy the named files into it from your own copy and import again.
`… has no text layer on N of N pages; it will be read by on-device recognition at import`	A warning, not a refusal. Either let the phone read it (Route A), or add a text layer on your computer first (Route B or C).
`prompt_rules should address 'fabricate'`	Add a rule containing the word fabricate; likewise for cite.
`document has no title`	Every entry in `documents` needs a non-empty `title`.
`… → unknown step`	A procedure branch points at a step id that does not exist.
`no terminal step is reachable`	Give the procedure at least one step with `"terminal": true` that the entry step can reach.
`core files total … characters (budget 32768)`	Warning only. Move the long material into a document.
`chains.md line N: … unknown step '…'` (or another message after a line number)	A rule chain step does not read. Fix that line; the message says what the step should look like.
`chains.md: the chain '…' needs Table N, which no manual in this vault prints as a table the app could find`	Warning only. The chain is kept and stops at that step. Check the table's number as the book prints it, or load the book that prints it.

MESSAGE	FIX
`table-checks.json entry N: …; skipped`	Warning only. That check does not read and is left out; the message says why. The rest of the file is kept.
`table-checks.json: N checks are for a PDF this vault does not hold (…)`	Warning only. The checks were made on another copy of the book. They are kept and apply if that copy is loaded.
`table-checks.json has version N; this app reads version 1, so it was not read`	Warning only. The file was written for another version of the app; no check in it is used.
`sites/…: unit '…' model '…' is not a model this vault has a section for`	Spell the model as its section's heading does, or mark a unit the vault does not cover `"covered": false`.
`sites/…: … names '…', which is not a declared unit`	A connection, control, setting or fault names a unit id the file does not declare.
`sites/…: serial '…' is also listed in sites/…`	One serial is one unit in one place; a reusable system type has no serials.
`sites/…: a site document's file must be named for its id`	Name the file `sites/<id>.json`, one file per site.
`sites/…: … contains a link`	Site documents are plain text; say who holds a login, not where it is.
`sites/…: the session's site block is held to 1600 characters …`	Warning only. Shorten the details, or accept that the assistant asks for the rest when it needs it.

Receiving a vault from a link or a QR code

Not everybody builds a vault. If a publisher — a distributor, a dealer, a manufacturer's own support arm, whoever may lawfully supply the manuals — sells you a set, they can hand it over as a link or a code on the page you land on after buying. The phone does the rest, and the manuals arrive already read for search, so there is no afternoon of on-device text recognition.

The app only ever receives. There is no "share this vault" anywhere in it: no link to send, no code to show somebody, no upload. A vault you built stays on the phones you put it on, and the export you can make of it carries no manuals at all. That is deliberate — an in-app share would make every subscriber a redistributor of manufacturers' manuals.

You need an active subscription (or a team licence) to add a vault of your own, by link or by folder.

1. Open **Settings › Custom Vaults** and tap **Add from Link or QR…**. Paste the address from the publisher's page, or tap **Scan a QR code** and point the phone at theirs. Scanning the same code with the phone's own Camera app works too — it opens the app on this screen.
2. The app shows you the **site** the vault would come from and asks before it downloads anything. It shows the site and nothing else of the address: a post-purchase link is often a one-time token, so the app treats it as a secret and never puts it on screen, in a log, in an export or in a job record.

3. When the download finishes you get a **review**: the vault's name and version, the site, how big it is, the manuals by title, and either - **Signed by <publisher>** — the archive is signed with a key the app knows, and nothing in it has changed since it was signed. One tap installs it; or - a highlighted **unverified source** warning — the archive is not signed, or is signed by somebody nobody has listed. It can still be installed, at your own risk, after you tick a second box saying you know where it came from. It is then marked **Unverified source** wherever it is listed, and the job record of any session that uses it says so.
4. Two things are refused outright, with no way to override them: an archive that has been **altered** since it was signed, and one signed by a publisher who has been **revoked**. A vault already installed from a publisher who is revoked later is left alone — you may be standing in a plant room depending on it — and is flagged in the list instead.
5. On a phone in **medical mode**, and on a phone carrying an organisation configuration that says so, unsigned vaults are refused as well. Ask the publisher to get their key listed.
6. Installing over a vault with the same id replaces the publisher's files and keeps your own edits to the core files. Indexing starts as it does after a folder import.

If anything goes wrong — the download fails, the archive is not readable, you change your mind — nothing is installed and nothing is left behind.

Publishing a vault

This section is for the other side: you hold the right to supply a set of manuals and you want a customer to be able to get the whole vault onto a phone from the page they land on after buying.

> You must hold the right to supply the manuals you put in a vault you publish. What that requires is between you and the manufacturer.

You host it, not us. A vault archive lives on your own site. The vendor does not host, mirror or proxy vaults, and the app's own catalog stays free of manuals.

Build the archive

Start from a vault folder that already imports cleanly (Steps 1–5 above), with the manuals extracted to text in `documents/`. Then, on a Mac:

```
./Scripts/make-vault-archive.swift path/to/acme_rtu \
    --publisher-id acme --publisher-name "Acme Manuals" \
    --key-file secrets/acme-signing-key.txt
```

That writes `acme_rtu.vaultarchive` — a zip of the folder plus a small header listing every file with its SHA-256 and size, and a detached signature over the lot. It prints the header so you can check what you are about to publish.

Get a signing key, and get it listed

```
swift Scripts/skillpack-sign.swift keygen secrets/acme-signing-key.txt
```

That writes the private half to a file only you can read and prints the **public** half. Send us the public half, the id you want to use and the name technicians should see. It goes into the signed catalog, and a phone that has seen the catalog will show *Signed by Acme Manuals* on your archives. A key can be revoked there without an app release, which is also what happens if one is lost.

Never pass a private key as a command-line argument; the tools refuse it. Arguments end up in `ps`, in shell history, in CI logs and in crash reports, and a key that has been through any of those has to be replaced.

Publishing without a key

You can build an unsigned archive by leaving the key flags off. It will install — behind a warning block and a second acknowledgement — and stay marked *Unverified source* on the customer's phone for as long as it is there. It is fine for handing a vault to your own crew. It is not fine as a product.

On your site

- Link to the `.vaultarchive` file directly, over https. That link is all the app needs.
- A QR code of the same address works, and so does `avenkin://vault?src=<the https address>`, which opens the app straight from the phone's Camera app. Builds from before October 2026 only answer to the older `openglasses://vault?src=…` form, which current builds still accept.
- **One-time or expiring links are worth using.** The app treats the address as a secret, but the publisher is the only one who can stop it being passed around.
- Keep archives under 250 MB. That is the ceiling the app will download and unpack; a vault of extracted text is a long way under it, and the size only becomes interesting when you include the manufacturers' original PDFs.
- To publish an update, rebuild the archive from the folder with a higher `version` in `manifest.json` and replace the file. A customer who installs it keeps their own edits.

Sharing a vault: three situations

Most of this guide assumes you are building a vault for your own technicians. Two other situations come up, and they work differently. Decide which one you are in before you start.

1 · Your own crew

You build one vault, your manuals included, and put it on the phones your technicians carry. Nothing changes; this is the path the rest of the guide describes.

2 · Building a vault for a customer

Here you assemble the whole vault for one customer: their core files, their manuals, their procedures. You test it, then you hand it over finished. Their technicians do not each repeat the work; one person prepares the manuals once and everyone else uses the result.

The manuals in such a vault are normally the customer's own. They already hold them through their OEM portal access, their dealer relationship, or the equipment itself. What you are providing is the assembly, the text extraction and the testing. If you intend to reuse one master set of manuals across several customers, that is redistribution and it needs the manufacturer's permission, so check your agreements first.

Before you hand it over:

- **Keep a maintainer's record outside the vault folder.** For each manual: where it came from, its checksum, its revision, and the evidence that it applies to each model you mapped it to. Keep your test results alongside. The folder that goes on the phone holds only what the manifest lists; the record is what lets you, or whoever maintains the vault after you, update it next year without redoing the research.
- **Write the test questions from your own map.** For each question, look the model up in `models.md` and confirm the document you expect the answer to cite is listed for it. A test that asks about a model the map doesn't link to the document fails for the wrong reason.
- **Test on the build the customer will run.** Import the exact folder you will deliver on a phone with that build, run your questions, and record the build, the AI model, the answer and the page it cited. Write the customer's instructions against that build, not against older release notes.
- **Check the folder you deliver, not the one you worked in.** If you use a script to check the vault, run it against the final folder, as the last step, and make sure it actually runs. A report is only as good as the copy it was made from.

Delivery is by hand, or by link. By hand: you install the finished folder on the phones you are already setting up, importing once per phone, a few minutes each. By link: you build a vault archive and put it on your own site, and their technicians add it from the link or the code — see **Publishing a vault** above. One scan can also set the phones up: ask us for an enrolment profile for the customer's organisation, and scanning its code (or entering its licence key) applies the organisation's configuration and installs its pack — the core files and procedures, published as in situation 3. A pack never carries manuals, so those still go by hand or by link; delivering them in the same step is planned.

3 · Publishing a pack

The parts you authored, the core files and the procedures, can be published as a **pack** that other Field Assist users install from the app's Packs list, and that a team licence can include for a whole organisation. What you are publishing is your fault-code tables, nameplate decoding, safety rules and diagnostics, which is where your field knowledge lives.

> **A pack must never contain manufacturer manuals.** It goes into a catalog anyone can install from, so everything inside it is copied to people you have never met. Keep the manuals out. A customer who installs your pack pairs it with manuals from their own vault, or from a vault you built for them as in situation 2.
>
> **Nor site documents.** A site document describes a customer's site, not a product, and the importer refuses a pack that carries `sites`.

To prepare one:

1. Copy the vault folder and delete `documents/` and `sites/`. Remove `documents`, `documents_dir` and `sites` from the manifest, and set `gating` to `{ "iap": "com.openglasses.vault.<id>" }`, where `<id>` is the vault's `id`.
2. Add a `pack.json` beside the manifest:

```
{
  "id": "com.openglasses.vault.acme_rtu",
  "vaultId": "acme_rtu",
  "version": "1.0.0",
  "name": "Acme RTU Service",
  "summary": "Fault codes, nameplate decoding, and diagnostics for Acme rooftop units.",
  "author": "Your company name, as technicians should see it",
  "licensePack": "acme_rtu"
}
```

3. Send the folder to us. We sign it, host it, and list it in the catalog; nothing unsigned can be installed. If the pack includes extracted manual text, it needs the manufacturer's written permission to redistribute, recorded in the pack, and we will ask for it.
4. Decide how it is sold. A pack is its own App Store product for solo technicians, and a team licence includes it when the code is issued with the pack's key, so your own customers get it with their licence rather than buying it separately.

Updating a published pack is the same folder with a higher `version`. Technicians who install the update keep their in-app edits and their own manuals.

Checklist before you send a vault to a customer

- [ ] Every PDF selects text on the computer, or you have decided to let the phone read it.
- [ ] No two documents hold the same text under different names.
- [ ] Each manual's `title` is the name the crew already uses, says what the document is, and is different from every other title.
- [ ] Fault codes sit under a heading per controller or family, with the models named in the section's first line, one code per row.
- [ ] No model number or manual number appears in a heading outside `models.md`, and each model's spellings sit in its own heading, in brackets.

- [ ] For a vault spanning several families, `models.md` lists the documents that apply to each model, confirmed from the manual or the manufacturer's catalogue.
- [ ] Every check in a `symptoms.md` has a document and page, and a person has read each page against it.
- [ ] `safety.md` says what you want spoken, in your words.
- [ ] Core files total well under 32 KB.
- [ ] Any site document has no contact details, passwords or links, and its serials are the customer's.
- [ ] If the vault is sealed with `vault_only`, its minimum build is stated, and a test job's support report says `Vault-only: on`.
- [ ] Import shows a section count for every manual.
- [ ] Saying each serviceable model settles the session on it without asking which one.
- [ ] A known fault code answers with a page citation.
- [ ] An out-of-scope question answers with "the manuals do not cover this".
- [ ] The test questions were run on the build the customer will use, and the results are in your maintainer's record.