How Commonweave Finds and Grades Organizations

A field guide to how we find aligned organizations, grade them, and keep the list honest.

25,837 organizations · 61 countries · one SQLite file

Version 1 · 2026-04-24

One-sentence summary

The pipeline pulls organization records from public registries and the open web, throws away the ones that obviously do not fit, scores what is left for alignment with the framework, merges duplicates only when the location agrees, flags dead-looking rows for a human to look at, and then publishes the survivors as the map, the directory, and a search index.

That is it. The rest of this document is what each of those steps actually looks like in code, and why a few of the choices are stricter than they look.

Why this document exists

Two audiences.

Humans - founders, journalists, skeptics, would-be contributors - keep asking a reasonable question: *where do all these organizations come from, and how do I know the list is not just scraped nonsense?* The answer is in this document, in plain language.

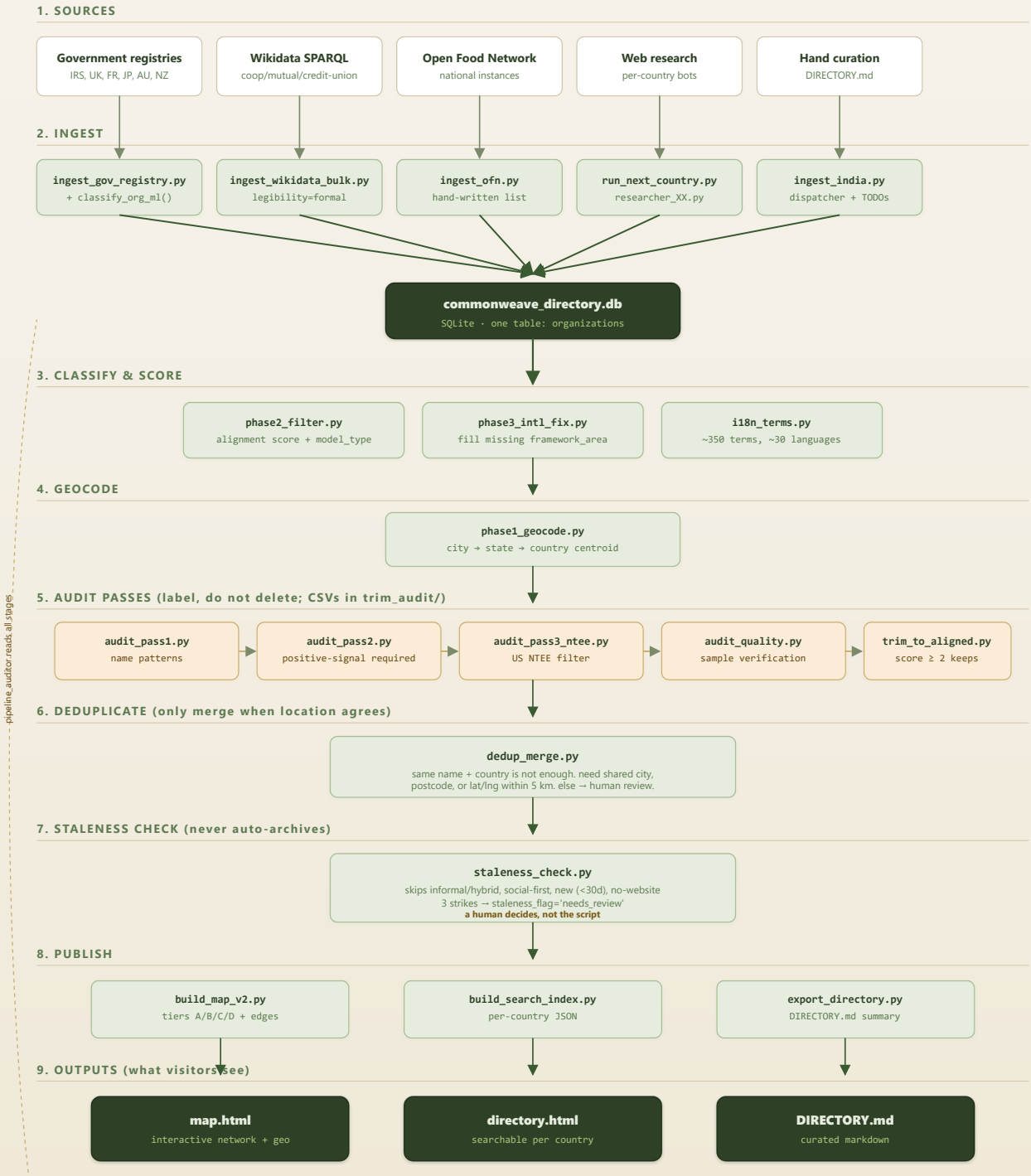
AI agents - the ones Simon or anyone else might run next week to add a source for a new country - need to know the shape of the pipeline well enough to slot a new ingester in without reading every script. Appendix A is for them.

The pipeline is a bunch of small Python scripts stitched together by convention, not a big framework. If that sounds hacky, it is also the thing that lets a single person keep the whole picture in their head. That trade is deliberate.

The big picture

The Commonweave pipeline

source → ingest → classify → geocode → audit → dedup → staleness → publish



META LAYER · weekly self-improvement loop

pipeline_auditor.py reads DB + logs, writes evidence-cited proposals to trim_audit/proposals-YYYY-MM-DD.md

never writes to the database · retires proposals ignored 3 weeks running · cites named source + specific numbers or it gets dropped

LEGIBILITY LADDER · every row carries one of these tags

formal registered legal entity found via registry, Wikidata, or official NGO directory

hybrid	registered but primarily operating informally (FPOs, SHG federations)
informal	unregistered: sangha, mutual aid, forest/water commons, neighborhood collective
unknown	not yet classified · the migration default · the auditor nags until it shrinks

commonweave · 2026-04-24 · scripts under commonweave/data/ · database at commonweave/data/commonweave_directory.db (gitignored)

Stage-by-stage walkthrough

Each stage has a short plain description, the files that do the work, what goes in, what comes out, and one choice worth knowing about.

Stage 1 - Sources

We treat the world like a filing cabinet with uneven drawers. Some drawers are tidy (the IRS file of US nonprofits, the UK Charity Commission register, Wikidata). Some are a mess or offline (most of India's village-level cooperative world). We pull from the tidy drawers first because it is easy, and we keep honest notes about the drawers we have not opened yet so the list does not secretly become "whatever is in English and already registered in a Western country."

The sources we use today:

- **IRS Exempt Organizations Business Master File** (US, public domain, about 1.8 million rows before filtering).
- **UK Charity Commission** (England and Wales register, plus Scotland's OSCR).
- **Wikidata** (a SPARQL query for cooperative, credit union, mutual aid society, and similar classes).
- **Open Food Network** (the cooperative food-distribution platform and its national instances).
- **ProPublica Nonprofit Explorer** (enriched US nonprofit records).
- **Targeted web research** (one country at a time, written by a researcher bot that reads a recipe of searches).
- **Manual curation** (hand-added entries, usually from DIRECTORY.md edits).

Stage 2 - Ingest

Every source has its own ingester. They all write into one SQLite database: `data/commonweave_directory.db`. The ingester's job is narrow. Download the data, turn each row into the Commonweave shape, and write it in.

- `ingest_wikidata_bulk.py` runs SPARQL queries against `query.wikidata.org` for the cooperative and mutual-aid classes, optionally per-country, respects a polite two-second sleep between queries, and writes rows with `source='wikidata_bulk'` and `legibility='formal'`.
- `ingest_gov_registry.py` downloads bulk files from UK, France, Japan, Australia, and New Zealand registries. It knows how to read each one's quirks and has multilingual keyword lists for English, French, Japanese, Portuguese, Spanish, and German so a French association does not get scored with only English words.
- `ingest_ofn.py` is a hand-written list of every Open Food Network instance around the world. It is short and specific because that platform has about twenty instances, not two million.

- `ingest_india.py` is an honest dispatcher. Today it runs the India slice of the Wikidata pull and writes a log of every other India source we *should* be pulling from (NGO Darpan, NCDC, NABARD PACS, Kudumbashree, JEEViKA, Van Panchayats, and so on), what tier each belongs to, and how long it would take to build. The point of writing those TODOs into the code is that the weekly pipeline auditor will see the gap and keep nagging until it closes.
- `researcher_HN.py` and its siblings are per-country research bots. Each one runs a fixed set of searches in the local language, validates the results, and writes a markdown file under `data/regional/`. `run_next_country.py` is the orchestrator that picks the next country off `QUEUE.txt` that has not been completed, runs the researcher, and updates `country_research_state.json`.

Every ingester tags the rows it writes with a **legibility** value (see Stage 4), a **source** string, and a **source_id** (for example the Wikidata QID, the IRS EIN, or the UK charity number). The source_id is what lets us avoid inserting the same row twice when we re-run.

Stage 3 - Classify and score

Raw records are useless if you cannot tell a community land trust from a country club. Classification happens in two places.

The ingester makes a first pass using `classify_org_ml` in `ingest_gov_registry.py`. This function looks at the name and description in the row's native language, checks it against keyword lists for each framework area (healthcare, food, democracy, and so on), and returns three things: the best-guess framework area, an alignment score, and a "should exclude" flag. Obvious exclusions (churches that are only places of worship, golf clubs, homeowners associations, chambers of commerce, booster clubs) are dropped at this step.

A second pass, `phase2_filter.py`, re-scores every active row in the whole database using a combined bank of English terms and the multilingual bank from `i18n_terms.py` (about 350 alignment terms across roughly 30 languages). This pass also sets a `model_type` column (cooperative, mutual_aid, foundation, research, federation, or nonprofit) so we can filter by shape of organization, not just topic.

`phase3_intl_fix.py` cleans up international rows that still do not have a framework area by scoring their name and description against a small English keyword set, so a row like "Kenya Community Health Network" gets `framework_area='healthcare'` rather than `NULL`.

One design choice worth naming: the pipeline never invents alignment that is not in the text. If a registry says "The Acme Society" and gives no description, the row gets a low score and usually gets trimmed. The pipeline would rather under-count than pretend it knows things it does not.

Stage 4 - The legibility ladder

This one matters on its own, so it gets its own section below. Short version: every row also carries a `legibility` value - `formal`, `hybrid`, `informal`, or `unknown` - that tracks *how* we found the organization. A formal cooperative with a government registration number is easy to find and easy to verify. An informal mutual aid network in a village is often not. We do not hide that difference, we label it.

Stage 5 - Geocode

`phase1_geocode.py` gives every active row a latitude and longitude, with a `geo_source` tag that says how precise the coordinate is.

- US rows: try an exact city match against a cached US cities file, then fall back to the state centroid.
- International rows: use the country centroid.

This is not surveying-grade accuracy. It is good enough for a world map that is honest about its resolution. The map's popup will tell you whether a pin came from a city match or a country centroid, so nobody mistakes a country-level dot for a specific street address.

Stage 6 - Audit passes

Before anything goes out, we run through three audit passes and a quality pass. Each pass labels rows it disqualifies, rather than deleting them. The CSVs of what was excluded live in `data/trim_audit/` so anyone can look at what got cut and argue with the call.

- `audit_pass1.py` - Pass 1, hard exclusion by name pattern. Churches that are only houses of worship, fraternal orders, homeowners associations, booster clubs, cemeteries, kennel clubs, professional guilds, and political party committees. Their `status` becomes `excluded_audit_p1`.
- `audit_pass2.py` - Pass 2, strict alignment. A row has to match at least one strong positive signal (cooperative, community land trust, mutual aid, food sovereignty, and so on) to survive. Zero-signal rows become `excluded_audit_p2`.
- `audit_pass3_ntee.py` - Pass 3, for US rows only. Keep only the NTEE major categories that match Commonweave's scope (C, E, F, I, J, K, L, O, P, Q, R, S, W). Everything else becomes `excluded_audit_p3`.
- `audit_quality.py` - A random-sample auditor that flags suspicious patterns after the passes have run.
- `run_audit.py` - An agent-style runner that does both filtering and enrichment (filling in missing websites and descriptions) in resumable batches.

`trim_to_aligned.py` is the last stop. It keeps only rows with `status='active'` and `alignment_score >= 2`. Everything else gets exported to a CSV and deleted. That is the only destructive step, and it happens after every prior label has been written so nothing disappears quietly.

Stage 7 - Deduplicate (only when the location agrees)

`dedup_merge.py` is where the pipeline is most careful.

The naive version is: group by normalized name plus country code, collapse each group into a single row. That is the version that deletes real organizations. "Farmers Cooperative Society" exists in thousands of Indian villages. A matching name in the same country tells us nothing about whether two rows are the same group.

So the script splits each candidate group into two tiers:

- **Auto-merge**: rows that share a location signal. Either the same city, the same postal code, or latitudes and longitudes within about five kilometers of each other.
- **Review**: rows whose locations disagree, or whose locations are missing. These get written to a review file for a human to decide.

Auto-merged rows are soft-deleted with `status='merged'` and a `merged_into` pointer back to the canonical row, so any link in the wild that points to the old ID keeps working.

Stage 8 - Staleness check (never auto-archives)

`staleness_check.py` pings the website on each organization's row and counts how many times it fails. After three fails it sets `staleness_flag='needs_review'`. It never flips the row to archived on its own.

The reason for the care: a website-centric check, applied naively, systematically prunes the most marginalized organizations. Grassroots, informal, non-Western, or social-media-first groups often have no domain or a link that is down for a month and then back. The script skips those on purpose:

- Rows with `legibility='informal'` or `'hybrid'` are skipped entirely.
- Rows whose description or notes point at Facebook, WhatsApp, Telegram, Twitter, Instagram, YouTube, or TikTok are skipped. A social-first organization is not dead because its website is down or never existed.
- Rows added in the last thirty days are skipped. Let them settle.
- Rows with no website are skipped. Their presence in the directory is not conditional on having one.

Stage 9 - Publish

Three build scripts turn the database into the things people look at.

- `build_map_v2.py` assigns a quality tier (A curated, B verified, C inferred, D unverified) to each active row based on data completeness and review status, computes network edges between organizations that share a framework area and are close by, and writes `map_points_v2.json`, `map_edges.json`, and `map_aggregates.json`. The front-end at `map.html` reads those.
- `build_search_index.py` writes one JSON file per country under `data/search/`. The directory at `directory.html` loads these on demand as the user picks a country.
- `export_directory.py` writes a curated markdown summary at `DIRECTORY.md` with stats, featured organizations, and embedded regional research.

Stage 10 - The self-improvement loop

`pipeline_auditor.py` runs once a week. It is read-only. It never writes to the database. Its job is to look at the current state of the pipeline and write a short proposal file to `data/trim_audit/proposals-YYYY-MM-DD.md`.

Every proposal has to cite a named source and specific numbers from this run. Generic proposals get dropped before the file is written. If a proposal has been ignored for three weeks in a row, the auditor retires it to `rejected-proposals.md` so we do not nag forever about something that has been decided.

This is the sharpening wheel. It means the pipeline gets a little better every week without Simon having to remember what it forgot last month.

The grading ladder (legibility and alignment)

Commonweave tries to be honest about what it can and cannot see. Two columns do the heavy lifting.

Legibility: how an organization became visible to us

The `legibility` column was added on 2026-04-22 because the pipeline was pulling almost entirely from Western, English-language, government-registered sources, and that was starting to look like the framework's answer rather than its starting point. The column takes four values:

- **formal** - registered legal entity found through a government registry, Wikidata, or an official NGO directory. The default for most of the current database.
- **hybrid** - registered but primarily operates informally. A farmer producer organization that exists on paper but lives through village-level practice. A self-help-group federation that has a state identifier but works person-to-person.
- **informal** - unregistered. A sangha, a mutual aid network, a forest or water commons, a neighborhood collective. These are the hardest to find and usually come from hand research, not from bulk ingest.
- **unknown** - not yet classified. The migration default for existing rows.

The legibility ladder does not say one tier is better than another. It says: here is what we are biased toward, in numbers, so you can argue with us.

Alignment: how well the row matches the framework

The `alignment_score` column runs from about minus three to plus ten. It is the sum of keyword hits in the name and description, weighted by how strong each signal is, with penalties for exclusion words. A row scoring less than two gets trimmed at `trim_to_aligned.py`. A row scoring five or more is in the defensible core that shows up in featured counts.

Alignment is a text-matching score, not an ideological read. A health nonprofit that lobbies against universal healthcare would match the keyword "health" just fine. The framework acknowledges this openly and treats the score as a *sector relevance* signal, not a political one. Human review is the layer where political fit gets sharpened.

The self-improvement loop, up close

One more word on `pipeline_auditor.py`, because it is the part that keeps the list from rotting.

Every Sunday, the auditor walks the database and the logs and asks five questions:

1. **Coverage.** Which countries are thin? Which framework areas are thin, relative to population and to the framework's own predictions?
2. **Legibility.** What fraction of each country is formal, hybrid, informal, or unknown? Flag countries that are nearly all formal as bias-narrow.
3. **Quality.** How filled-in are the fields? What is the staleness rate? How is the alignment score distributed?
4. **Source performance.** Which ingesters are plateauing? Which sources have stopped paying for their slot?
5. **Unbuilt proposals.** Read the last four weeks of proposals. Anything still not built? If it has been ignored three weeks in a row, retire it.

The proposals file is how Codex, or a future maintainer, or Simon when he has a Saturday, knows what to look at next. It is the to-do list the pipeline writes for itself.

Appendix A - How to add a new source

For humans or AI agents extending the pipeline. Follow these steps in order.

1. **Pick the source.** A public registry, an open dataset, a thematic directory, or a single website worth scraping once. Write a one-paragraph note about what it is, how big it is, what license it uses, and what shape the data is in (CSV, JSON, XML, API, a web page). If it is a web-only source with no open data, expect to spend most of your time on a scraper, not on ingestion.
2. **Write a new file at `commonweave/data/ingest_<source>.py`.** Model it after the closest existing ingester:

- Bulk CSV or ZIP download → copy `ingest_gov_registry.py`.
- SPARQL or REST API with per-class or per-country queries → copy `ingest_wikidata_bulk.py`.
- A hand-written list of known entities → copy `ingest_ofn.py`.
- A dispatcher that runs multiple slices and writes a log of TODO sources → copy `ingest_india.py`.

3. **Use the shared helpers in `_common.py`.** Use `DB_PATH` so you write to the right database. Call `ensure_column` if your source needs a new column. Call `normalize_name` before any dedup check. Use `trim_audit_path('ingest-<source>')` for your run log.
4. **Populate these database columns on every row you insert** (the minimum):

- `name`, `country_code`, `country_name`
- `registration_id`, `registration_type` (for example `'WIKIDATA_QID'`, `'UK_CHARITY'`, `'IRS_EIN'`)
- `description`, `website` (both allowed empty but better if filled)
- `source` - the short identifier you will grep for later
- `source_id` - the unique key in the source system (so re-runs do not duplicate)
- `framework_area`, `alignment_score` - set by `classify_org_ml` or the equivalent
- `legibility` - one of `formal`, `hybrid`, `informal`. Do not guess. If your source is a government registry, `formal`. If it is informal mutual aid research, `informal`. If mixed, `hybrid`.
- `status='active'`
- `date_added` - ISO timestamp

5. **Always use `INSERT OR IGNORE`** keyed on `source` plus `source_id`. Re-running your ingester should be a no-op on rows already present.
6. **Score, do not assume.** Route every row through `classify_org_ml` (import from `ingest_gov_registry`). If the exclude flag is true, skip. If the score is below one and the name is not a

recognized nonprofit entity type (cooperative, foundation, trust, charity, and so on in several languages via `is_nonprofit_entity_type`), skip.

7. **Log the run** to `data/trim_audit/ingest-<source>-YYYY-MM-DD.md` with counts: rows seen, rows inserted, rows skipped with reason, rows excluded by the scorer, errors.
8. **Register the source in the weekly auditor.** Nothing formal. Just make sure the auditor's coverage report will see your new rows. The auditor reads the whole `organizations` table, so as long as your rows are in, they will show up.
9. **Document.** Add a one-line entry for your new source to `DATA.md` under the Current Composition table. If the source deserves its own note, add a subsection to `META-DIRECTORY.md`.
10. **Test, then run.** Run with `--dry-run` first if the ingester supports it, check a sample of rows by hand, then run for real. Expect the first real run to surface bugs in your classification call.

A brand-new ingester done right is about 200 lines of Python plus a day of watching what it actually pulled.

Appendix B - How to audit the list yourself

Three places to look.

The trim audit folder, at `commonweave/data/trim_audit/`. Every exclusion pass writes a CSV there of the rows it cut and why. `proposals-YYYY-MM-DD.md` files are the weekly auditor's notes. Reading a month's worth of proposals is the fastest way to see what is wrong with the pipeline right now.

The search JSON, at `commonweave/data/search/<COUNTRY>.json`. One file per country. Pick one with fewer than fifty organizations and read it end to end. Any row that looks wrong is a finding. File a pull request with corrections and a one-line note at the top of the JSON explaining what you checked.

The database directly, at `commonweave/data/commonweave_directory.db` (gitignored, local only). SQLite, so `sqlite3` from the command line works. The schema is in `DATA.md`. Two queries worth running:

- `SELECT country_code, legibility, COUNT(*) FROM organizations WHERE status='active' GROUP BY country_code, legibility` - shows which countries lean formal and which have any informal coverage.
- `SELECT framework_area, COUNT(*) FROM organizations WHERE status='active' GROUP BY framework_area` - shows where the data is thin (energy_digital at 40 rows, for example).

Anything surprising in either of those is worth writing up.

Footer

- Repository: github.com/simonlpaige/commonweave
- Directory: commonweave.earth/directory
- Interactive map: commonweave.earth/map

This document describes how the list is made. The list itself lives at the directory link above. Corrections and pull requests welcome.