Technical Architecture & Screening Workflow

How the IRM Sanctions Screener downloads, parses, matches and reports

The IRM Sanctions Screener is a local Windows desktop application — a CustomTkinter GUI on Python 3.10+ — built as a config-driven, one-directional pipeline that screens a company or a natural person against the official lists of OFAC (US), the UN, the UK (OFSI/HMT) and the EU, then renders a professional PDF audit report. Lists are fetched straight from each issuing authority and cached on disk, so every screening after the first update runs fully offline. No third-party sanctions provider sits in the path: there are no API keys, no subscriptions and no vendor lock-in, and subject data never leaves the machine.

OVERVIEW

System at a glance

Each stage of the tool is an isolated module under `sanctions_screener/`, wired together purely through the `sources` map in `config.json`. The runtime stack is small and auditable:

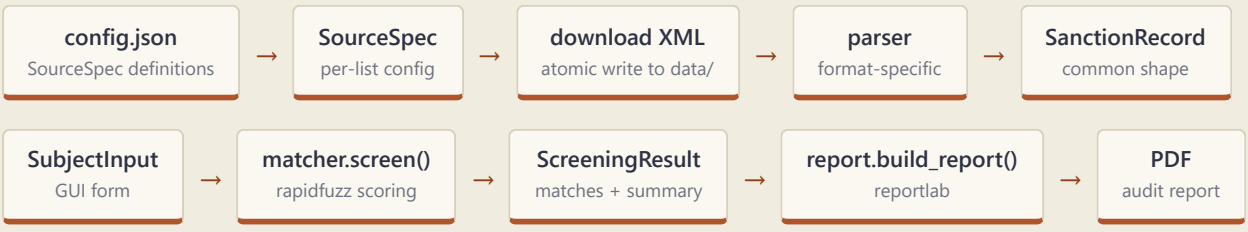
Python 3.10+ · CustomTkinter · ttk.Treeview · rapidfuzz · reportlab · requests · PyInstaller · Pillow · pytest

4	5	3	SHA-256
Issuing authorities	Official lists screened	Match tiers	Per-source integrity

ARCHITECTURE

Data-flow pipeline

Two pipelines run end to end: **A** turns configuration into cached, parsed records, and **B** turns a GUI form into a signed-off PDF.



RUNTIME MODEL

Concurrency & UI model

The GUI is a single `App` class in `app.py`, launched through `__main__.py` → `app.main()`, and presents two tabs: **Screen** and **Lists**. Every slow operation — downloads, screening and PDF generation — runs on **daemon threads** so the interface never blocks.

> **THREAD-SAFETY CONTRACT**
>
> Worker threads never touch Tk widgets directly. Results are marshalled back to the Tk main loop through a `queue.Queue` plus custom virtual events, posted with `self._post(...)` and bound in `_wire_events`: `EVENT_LOG`, `EVENT_DOWNLOAD_DONE`, `EVENT_SCREENING_DONE`, `EVENT_REPORT_DONE` and `EVENT_ERROR`. The UI uses a hand-built cream `PALETTE` derived from `branding.COLORS`; a reskinned `ttk.Treeview` backs both the result and source tables.

WORKFLOW

The screening workflow, stage by stage

Stage 1 — Update & download

`download_source` builds an ordered candidate-URL list via `_candidate_urls` and tries each in turn until one succeeds. Every download is **atomic**: bytes land in a `.part` temp file and are promoted with `os.replace` only after validation.

- ✓ **EU resolution.** RSS discovery reads the FSF feed each run to locate the current `xmlFullSanctionsList` URL, and falls back to the OpenSanctions mirror if the primary is unavailable.
- ✓ **Resilient fetch.** Transient failures (HTTP 5xx / connection drops) are retried with exponential backoff; an HTTP 5xx ultimately raises `SourceError` as a retryable outage.
- ✓ **Cache protection.** `_validate_xml_download` checks a minimum size and `iterparse` well-formedness **before** `os.replace`, so a truncated or HTML response can never poison the cache.
- ✓ **Auth handling.** HTTP 401 / 403 fail over immediately rather than retrying.
- ✓ **Bookkeeping.** On success a `data/manifest.json` entry records the timestamp, byte size, SHA-256, the resolved URL/label that actually worked, and any attribution.

Stage 2 — Parse

One parser exists per list **format**, registered in `PARSERS` keyed by the config `format` string; each exposes `parse_*(path, source_key)` returning `list[SanctionRecord]`. The four XML schemas differ materially:

`ofac.py`
Namespaced XML. SDN and Consolidated share one schema and are distinguished by `source_key`.

`un.py`
Separate `INDIVIDUALS/INDIVIDUAL` and `ENTITIES/ENTITY` sections.

■

`uk.py`

FCDO format. Concatenates `Name1..Name6`, detects the “Primary Name”, and treats non-Latin names as aliases.

■

`eu.py`

FSF format. Deliberately namespace-agnostic — matches by local tag name with attribute fallbacks, since field and namespace details vary between EU schema versions.

`base.py` defines the frozen `SanctionRecord` dataclass — the common shape every parser must emit — and `normalize_subject_type`, which canonicalises source type strings to `individual`, `entity`, `vessel`, `aircraft` or `unknown`.

Stage 3 — Match

`screen()` normalises every name with NFKD diacritic stripping, lowercasing and punctuation removal; for company subjects it additionally strips the legal-form suffixes listed in `config.json`. It then scores each record's candidate names with rapidfuzz as the **maximum** of token-set ratio, token-sort ratio and a conditional partial ratio. Obvious individual-vs-entity type mismatches are filtered out up front.

Tier thresholds applied to the rapidfuzz score

Tier	Score threshold	Meaning
STRONG	≥ 90	High-confidence name match
POSSIBLE	≥ 80	Likely match for reviewer attention
WEAK	≥ 70	Low-confidence candidate

Disambiguation rules applied after scoring

Signal	Effect
Exact identifier match	Promote to STRONG
Same-kind, different-value identifier	Evict the record entirely
Year-of-birth / incorporation mismatch	Downgrade one tier
Country mismatch	Downgrade one tier

AUDIT DEFENSIBILITY

Per-session caches `_RECORD_CACHE` and `_PREPARED_CACHE` are keyed by the file's `(path, size, mtime_ns)`; a re-download changes `mtime` and auto-invalidates them. The prepared normalization produced by `matcher.prepare_records()` is **bit-identical** to the inline normalization — a fact guarded by a parity test — and there are deliberately **no** recall-reducing pre-filters, so a true hit can never be silently dropped and results stay reproducible.

Stage 4 — Report

`build_report(result, output_dir)` renders the PDF with reportlab using a `BaseDocTemplate`, A4 page size and a per-page footer. Sections are emitted in a fixed order:

- Cover
- Subject details
- Methodology
- Sources screened
- Summary
- Per-match detail blocks
- Reviewer sign-off
- Attributions — emitted only when a fallback mirror actually served data
- Disclaimer

The output filename follows the pattern `{sanitised-case-id}_{YYYYMMDD-HHMMSS}.pdf`.

COVERAGE

Sources screened

Five official lists across four authorities, each routed to its parser by the `format` field:

Authority	List	Format	Endpoint / resolution
OFAC (US)	Specially Designated Nationals (SDN)	`ofac`	`treasury.gov · ofac/downloads/sdn.xml`
OFAC (US)	Consolidated (non-SDN)	`ofac`	`treasury.gov · ofac/downloads/consolidated/cons_prim.xml`
UN	Consolidated List	`un`	`scsanctions.un.org · resources/xml/en/consolidated.xml`
UK (OFSI/HMT)	OFSI/HMT Consolidated	`uk`	`ofsistorage.blob.core.windows.net · …/ConList.xml`
EU	Financial Sanctions Files (FSF)	`eu`	RSS discovery on the FSF feed (`webgate.ec.europa.eu`, matches `xmlFullSanctionsList`); falls back to the OpenSanctions mirror `data.opensanctions.org · …/eu_fsf/source.xml`

STATE

Configuration & state

`config.json`

Committed and safe to hand-edit. Holds `sources` (urls, format, discovery, fallback_urls), `matching` (thresholds, entity_suffixes) and `ui`. A copy placed next to the `.exe` overrides the bundled one without rebuilding.

`config.local.json`

Per-user, created at runtime (currently `screener_name`). Merged under the `"local"` config key.

`data/manifest.json`

Runtime source of truth for download state — per-source timestamp, byte size, SHA-256, record count, resolved URL/label and attribution. The report reads it; it is never hand-edited.

FROZEN-VS-SOURCE DUALITY

`paths.py` centralises path resolution. As a PyInstaller executable (`sys.frozen`), `ROOT` is the directory containing the `.exe` — so `data/`, `reports/` and `config.local.json` persist beside it — while `BUNDLE` / `_MEIPASS` holds read-only bundled resources; from source, both point at the project root. Writes are atomic, and `load_manifest` tolerates a corrupt file by backing it up to `manifest.corrupt.json` and returning an empty mapping.

OPERATIONS

Build & run

```
# Run from source (first run creates .venv, installs requirements.txt, launches the GUI)
run.bat
.venv\Scripts\python.exe -m sanctions_screener

# Build the standalone single-file executable
build.bat            # produces dist\IRM-SanctionsScreener.exe

# Run the automated test suite
.venv\Scripts\python.exe -m pytest
```

EXTENSIBILITY

Extending the tool

Adding a new list in a new format is a three-step change. Because the GUI, downloader, screener and report are all config-driven off the `sources` map, no other wiring is required.

1. **Declare the source.** Add a `sources` entry in `config.json` and set its `format` (plus any `discovery` or `fallback_urls`).
2. **Write the parser.** Create `parsers/<name>.py` exposing `parse_*(path, source_key)` that returns `list[SanctionRecord]`.
3. **Register it.** Add the parser to `PARSERS` in `parsers/__init__.py`, keyed by the `format` string used in the config.