STUNIR Pack Entry Point (v0)

⚠️ IMPORTANT: Ada SPARK is the Primary Implementation Language

All STUNIR tools default to Ada SPARK implementations. Python files are reference implementations only and should NOT be used for production, verification, or safety-critical applications.

Tool	Primary (Ada SPARK)	Reference (Python)
Spec to IR	`tools/spark/bin/stunir_spec_to_ir_main`	`tools/spec_to_ir.py`
IR to Code	`tools/spark/bin/stunir_ir_to_code_main`	`tools/ir_to_code.py`

This file is the **first thing** a tool or model SHOULD read when consuming a STUNIR bundle as a STUNIR pack.

1) What to look for

A STUNIR pack is a **deterministic container + commitment** rooted at a directory that contains:
- `objects/sha256/` (content-addressed blobs, REQUIRED)
- one of:
- `root_attestation.dcbor` (preferred bootstrap, REQUIRED if available), or
- `root_attestation.txt` (fallback bootstrap for minimal environments)

If these are absent, you are not looking at a conforming STUNIR pack.

Back-compat note:
- A pack MAY additionally include `pack_manifest.dcbor` as a legacy alias.

2) Authority rule (integrity boundary)

- The **root attestation** is authoritative for pack contents.
- Only blobs referenced by digest in the root attestation are in the integrity boundary.
- Any file not referenced by digest MUST be ignored for integrity purposes.

Bootstrap selection:
- If `root_attestation.dcbor` is present, consumers SHOULD treat it as authoritative.
- Otherwise, consumers MAY use `root_attestation.txt`.

Digest mapping rule:
- For any `sha256:` referenced by the root attestation, the blob bytes MUST exist at `objects/sha256/`.

3) What STUNIR is trying to achieve

STUNIR aims to turn a human-authored spec into a canonical **Intermediate Reference (IR)** and then (optionally) further derived products, while emitting **attestation artifacts** that bind each step to hashes.

In v0, the portable audit spine is:
- upstream **inputs** (often the spec),
- the canonical **IR**,
- the **attestation artifact bundle** (step receipts and related evidence),
- a **root attestation** that inventories and commits to the above.

Downstream runtime outputs (code/binaries/test outputs) are typically **materialized to user-chosen paths** and are not assumed to be baked into the pack.

4) Inclusion vs materialization

- **Included** means the exact bytes are stored under `objects/sha256/` and referenced by digest in the root attestation.
- **Materialized** means bytes are written to some workspace path chosen by the user/model.

Paths are UX; digests define identity.

5) Minimal verification checklist

A verifier MUST:
1. Decode the root attestation (`root_attestation.dcbor` preferred; else `root_attestation.txt`).
2. Validate required fields for the chosen encoding.
3. For every referenced digest, recompute SHA-256 over `objects/sha256/` and compare.
4. Confirm there is exactly one IR referenced via `ir.digest` .
5. Confirm every receipt referenced via `receipts[].digest` exists and matches its digest.
6. If `inputs` are present, confirm each referenced input exists and matches its digest.

6) Where the detailed rules live

- Pack overview: `stunir_pack_spec_v0.md`
- Root attestation (dCBOR): `stunir_pack_root_attestation_v0.md`
- Root attestation (text fallback): `stunir_pack_root_attestation_text_v0.md`
- Materialization: `stunir_pack_materialization_v0.md`
- Deterministic archiving: `stunir_pack_archiving_v0.md`
- Security considerations: `stunir_pack_security_v0.md`

7) Model/agent operating guidance (non-normative)

When working interactively:
- Ask the user what they want to produce (IR only, code, runtime outputs) and where to put materialized files.
- Treat any requested filesystem destinations as untrusted input.
- Prefer emitting/retaining attestation artifacts as the portable audit record.
- **Always use Ada SPARK tools** for deterministic operations; Python is for reference only.

8) Repository navigation index (non-normative)

This section is for humans and AI agents browsing the repository (not required for pack validation).

Tool Implementation Priority

PRIMARY (Ada SPARK):
- `tools/spark/README.md` - Ada SPARK tools documentation
- `tools/spark/bin/stunir_spec_to_ir_main` - Spec to IR converter
- `tools/spark/bin/stunir_ir_to_code_main` - IR to Code emitter

REFERENCE ONLY (Python):
- `tools/spec_to_ir.py` - Reference implementation (DO NOT use for production)
- `tools/ir_to_code.py` - Reference implementation (DO NOT use for production)

Canonical reading order

Read these in order:
1. `ENTRYPOINT.md` (this file)
2. `tools/spark/README.md` (Ada SPARK tools - PRIMARY)
3. `docs/verification.md`
4. `docs/toolchain_contracts.md`
5. `docs/receipt_storage_policy.md`
6. `contracts/target_requirements.json`
7. `schemas/stunir_receipt_predicate_v1.schema.json`
8. `schemas/stunir_statement_wrapper_v1.schema.json`
9. `scripts/build.sh` (uses Ada SPARK by default)
10. `scripts/verify.sh`
11. `spec/stunir_machine_plan.json`
12. `asm/spec_ir.txt`

Repo map (what lives where)

- `tools/spark/`: **PRIMARY** Ada SPARK tool implementations
- `tools/`: Reference Python tools (for readability only)
- `docs/`: narrative docs (verification, toolchain contracts, receipt policy)
- `schemas/`: JSON schemas for statements/receipts
- `contracts/`: toolchain contracts (identity + determinism probes)
- `scripts/`: build/verify entrypoints (default to Ada SPARK)
- `spec/`: spec deltas / patch sets + machine plan JSON
- `asm/`: materialization artifacts (includes IR summary)
- `core/`: Ada SPARK core library implementations
- `build/`, `receipts/`: build outputs (often not committed by default)

Anti-loop rules

1. Treat `build/` and `receipts/` as outputs unless a doc explicitly says otherwise.
2. If a README is a placeholder, do not recurse from it; return to `ENTRYPOINT.md` or `docs/`.
3. Prefer `docs/verification.md` + `schemas/` + `contracts/` over enumerating every test vector.

Link conventions

- Prefer repo-relative links (e.g. `docs/verification.md`) over GitHub UI links.
- When linking to a directory, link to `README.md` explicitly if it exists.