ARCHITECTURE REFERENCE

ODT-MIM

USAREUR-AF

State of the MIM Project

Executive Summary

HEADQUARTERS
UNITED STATES ARMY EUROPE AND AFRICA
(USAREUR-AF)
Wiesbaden, Germany

20 MARCH 2026

DRAFT — UNOFFICIAL — NOT FOR OPERATIONAL USE

STATE OF THE MIM PROJECT

Snapshot date: 2026-02-28

EXECUTIVE SUMMARY

The project has a solid functional core for:

- Parsing MIM source artifacts into IR (`mim-portal`)
- Modeling and validating IR (`mim-ir`)
- Generating Foundry ontology code (`mim-backend-foundry`)

Overall assessment: core pipeline is operational, platform expansion layer is mostly planned, and documentation/packaging consistency is the main current risk.

WHAT IS WORKING NOW

1. Canonical IR + JSON Schema

- `mim-ir` provides model types, serialization helpers, and schema validation.
- Schema file exists at `mim-ir/schema/mim_ir_schema.json`.
- IR export/validation utility exists in `mim-backends/mim-backend-jsonschema/export_validate_ir.py`.

Observed output on existing model: - `mim_model.json` validated successfully. - Summary: `1208 classes`, `522 enumerations`.

2. Parsing Layer (`mim-portal`)

- High-level parse API supports HTML, XSD, and XMI.
- `parse_html()`, `parse_xsd()`, `parse_xmi()` are implemented and tested.
- Portal client and model bundle enrichment flows are implemented (`get_model()` with XSD/XMI enrichment).

3. Foundry Backend (`mim-backend-foundry`)

- Conversion pipeline and codegen are implemented.
- Foundry models, mapper, config, and TypeScript emitter are present.
- High-level API (`palantir.generate_mim_packages`) is implemented.

4. Frontend Viewer App (`apps/mim-viewer`)

- Real React/Vite app is present (not scaffold-only).
- Includes generation script to pull and serialize model data for frontend visualization.

IMPLEMENTATION FOOTPRINT

- Repository tracked file count (`rg --files`): **333**
- `.gitkeep` placeholders: **64**
- Data directory size: **224M**
- Large generated artifacts:
- `mim_model.json`: **9.4M**
- `ontology_definition.ts`: **1.4M**, ~49k lines
- `docs/samples/mim_json/mim_foundry_ontology.json`: **4.9M**, ~143k lines

Approximate Python footprint by area:

- `mim-ir`: 38 `.py` files, 11 test files
- `mim-portal`: 55 `.py` files, 10 test files
- `mim-backend-foundry`: 27 `.py` files, 2 test files
- `shared/mim-kernel`: 8 `.py` files, 3 test files
- `mim-backend-jsonschema`: 1 `.py` file

TEST/VALIDATION STATUS

Passing

- `uv run pytest mim-portal/tests -q` → **106 passed, 38 skipped**

- `uv run pytest mim-backends/mim-backend-foundry/tests -q` → **9 passed, 5 skipped**
- `uv run python mim-backends/mim-backend-jsonschema/export_validate_ir.py --input mim_model.json` → **schema validation passed**

Failing (collection/import level)

- `uv run pytest mim-ir/tests -q` → **11 collection errors**
- `uv run pytest shared/mim-kernel/tests -q` → **3 collection errors**

Primary failure mode: tests import `mim_ir` / `mim_kernel`, while packages are currently under `mim.ir` / `mim.kernel`.

MATURITY BY AREA

Area	Status	Notes
`mim-ir` core models/schema	Strong implementation, broken test imports	Code exists; tests/docs still reference old module names
`mim-portal` parse + portal client	Strong	Best test signal in repo; CLI migration still incomplete
`mim-backend-foundry`	Strong	Working backend despite README saying "planned"
`mim-backend-jsonschema`	Emerging	Utility script exists; not yet full backend package
Other backends (`openapi`, `python`, `ts`, `sql`)	Mostly scaffold	Readmes/placeholders, little or no code
Adapters	Mostly scaffold	Readme-only for most; dagster skeleton has empty files
Future packages (`contracts`, `mim-sdk`, `transform`)	Scaffold	Mostly planning docs; minimal executable code
CI/CD	Not implemented	`.github` has no workflows

KEY GAPS AND RISKS

1. Namespace and Packaging Drift

- Source code uses `mim.*` namespace packages.
- Documentation and tests often still reference old names (`mim_ir`, `mim_kernel`, `mim_parse`, `mim.core`).
- This causes immediate test collection failures and onboarding friction.

2. Documentation Drift vs Reality

- Core README references paths and commands that no longer match repository layout.
- Several component READMEs still state "Planned; implementation pending" even where code is already substantial (notably Foundry backend).

3. CLI Surface Is Fragmented

- `mim-portal` CLI is explicitly disabled in pyproject.
- `mim-ir` tools CLI is placeholder-only.
- Future `mim-cli` shim points to stale/broken path.

4. Root Package Build Ambiguity

- Root `pyproject.toml` builds from `_mim/mim`, but that tree currently appears empty of source files.
- Release/distribution risk if root package artifacts are expected to contain runtime code.

5. Coverage Asymmetry

- Parse/backend core has real tests and passing signal.
- IR/kernel tests currently fail at import stage; no integrated CI to catch this continuously.

IMMEDIATE PRIORITIES (RECOMMENDED)

1. **Unify import namespace strategy** — Decide and enforce one public namespace (`mim.*` vs legacy names), then update tests/docs.

2. **Repair test collection for `mim-ir` and `mim-kernel`** — Fix imports first so actual behavioral failures (if any) can surface.
3. **Align README and command paths with current repo reality** — Especially root README and backend/adapters status sections.
4. **Define one supported CLI path for core workflows** — At minimum: parse/export/validate/convert commands with clear entry points.
5. **Add baseline CI** — Start with lint + core test suites + schema validation smoke test.

30/60/90 VIEW

Next 30 days

- Fix import/module naming drift.
- Make core tests green or explicitly quarantined.
- Update top-level docs to match current package layout.

Next 60 days

- Consolidate CLI into one supported interface.
- Promote JSON schema backend from utility script to proper package API + tests.
- Add trace/provenance manifest emission in Foundry output.

Next 90 days

- Start one additional backend (Python or TS) with real output contract.
- Move one adapter from placeholder to runnable (likely files/sql first).
- Add release automation for package versioning and publishing checks.

BOTTOM LINE

The project is no longer a pure scaffold: it has a meaningful operational core for MIM ingest → IR → Foundry generation. The next phase is **stabilizing interfaces, removing naming drift, and hardening release/CLI/CI practices** so expansion work (contracts, SDKs, adapters, extra backends) can proceed

safely.