RHUMB WORKFLOW PROTOCOL™ PAPER

Rhumb Workflow Protocol™ Implementation Brief

Artifact model, profiles, validation, and Meridian™ compatibility for RWP™ 0.31.0

RWP™ 0.31.0 Tool builders, platform engineers, AI workflow implementers

Contents

CONTRACT

Implement the artifact semantics before implementing a UI

A conformant Rhumb™ implementation must preserve the protocol artifacts and their relationships. The UI can be a CLI, desktop app, web app, agent integration, or non-technical assistant. The durable contract is the artifact model, not the front end.

The core artifacts are plain files so humans and tools can both inspect them. The implementation may index them in a database or expose them through a higher-level interface, but the protocol value comes from being able to emit, read, validate, and hand off the same artifact semantics.

- `INTAKE.yaml`: request capture, pain points, constraints, requirements, and success criteria.
- `PLAN.md`: phases, deliverables, dependencies, risks, tasks, and verification evidence.
- `state.yaml`: plan execution state, current phase, timestamps, errors, handoffs, and recovery context.
- `manifest.yaml`: registry of produced files, prompts, handoffs, audits, and deliverables.
- `dependencies.yaml`: dependency tracking where work can block, fan out, merge, or coordinate across phases.
- `HO-*.yaml|md`: handoff documents that preserve context across phase/session/tool boundaries.
- `IDEA`, `AVD`, `ACS`, `MP`: architecture path from concept capture through component specification and managed execution.

PROFILES

Core RWP™ does not mandate Meridian™ directories

Core RWP™ defines artifacts, identifiers, lifecycle semantics, validation, conformance, and extension behavior. It does not require `.meridian/`, `.private/`, SQLite, a daemon, or a desktop runtime.

Directory trees are implementation profiles. The simplest profile can store plan artifacts under `rwp/plans/<plan-id>/`. Meridian uses a richer reference profile under `.meridian/.private/{runtime,data,knowledge}`.

- Use the Core File-Tree Profile for examples, simple tools, and external adopters.
- Use the Meridian™ Reference Profile when a tool must interoperate with Meridian™ artifacts.
- Use a custom profile only when it preserves artifact formats, identifiers, lifecycle rules, and validation behavior.

REFERENCE PROFILE

Distinguish plan state from implementation runtime state

An RWP™ implementation will typically maintain two layers of state: protocol-level plan artifacts and its own runtime coordination files. The plan-level `state.yaml` inside a plan directory is an RWP™ execution artifact. An implementation's session index or operational database is a separate concern.

For example, YAKKL® Meridian™ (the reference implementation) organizes plans under `.meridian/.private/knowledge/plans/<lifecycle>/` and keeps runtime coordination in `.meridian/.private/runtime/`. Other implementations are free to use any directory structure that preserves the artifact semantics.

- RWP™ artifacts: plan-level `state.yaml`, `PLAN.md`, `manifest.yaml`, handoffs – these must conform to protocol schemas.
- Plan lifecycle buckets: `backlog`, `planning`, `processing`, `completed`, `cancelled`, `onhold`, `archived`.
- Implementation runtime state: session indexes, coordination files, locks, and operational databases – these are implementation-specific.
- The protocol does not prescribe where runtime state lives, only that it must not be confused with plan-level artifacts.

VALIDATION

Conformance is mechanical, not marketing language

`rhumb-validate` runs category checks for schema, template, workflow, adapter, and grammar behavior. A public conformance claim must be backed by the validator and by trademark-policy compliance.

A multi-category failure exits as `6`. The point is mechanical evidence: a tool either validates schemas, template identity, workflows, adapter manifests, and grammar behavior or it does not.

- Schema checks validate JSON/YAML instances against canonical schemas.
- Template checks detect drift from canonical templates after canonicalization.
- Workflow checks verify cross-file invariants such as plan/state agreement.
- Adapter checks validate integration manifests and adapter shape.
- Grammar checks validate phase sequence notation and uppercase `A-Z` sub-phase identifiers.

RESOLVED PROTOCOL DECISIONS

The greenfield protocol shape is explicit

The sub-phase range is `A-Z`. The parser, schemas, fixtures, and prose now converge on that shape.

The state vocabulary has now been made greenfield and explicit. `PLAN-STATE.yaml.template` and the state schema use `planning`, `processing`, `paused`, `completed`, and `failed` for workflow execution, and `pending`, `in_progress`, `completed`, `failed`, and `skipped` for phases. Older aliases are not accepted by the current protocol shape.

- Phase identifier pattern: `^P-[0-9]{2}(-[A-Z])?$`.
- Decided: current RWP™ status vocabulary is canonical and older aliases are not carried forward.
- Implementations should clearly separate their runtime state files from protocol-level `state.yaml`.
- Needs validator work: profile-aware discovery should avoid false positives on implementation control files.

EXTENSIONS

Extensions must be explicit and namespaced

Implementations will need custom fields. That is acceptable if the extension does not change core semantics invisibly. Custom fields should be namespaced and documented so other tools can ignore or preserve them safely.

Meridian-specific runtime features should move into the Meridian™ Reference Profile or namespaced extensions unless they are promoted into vendor-neutral protocol concepts.

IMPLEMENTATION SEQUENCE

Build in this order

Start with artifact generation and parsing. Add schema validation. Add cross-artifact invariants. Add profile-aware discovery. Add UI only after the durable record is correct. A good interface can mask complexity for non-technical users, but it should not invent a second workflow contract.

- Generate `INTAKE.yaml`, `PLAN.md`, `state.yaml`, `manifest.yaml`, `dependencies.yaml`, and handoffs from templates.
- Validate schemas and sequence identifiers locally without network calls.
- Support Core File-Tree Profile first, then Meridian Reference Profile if needed.
- Keep generated PDFs and website pages downstream from protocol sources.
- Treat YAKKL® Meridian™ as proof and pressure-test input, not as an excuse to vendor-lock the protocol.