# Library-Level Solid Access-Control Conformance: WAC and ACP Decision-Parity Ratchets Joining a Cross-Family Scoreboard

Jesse Wright · the sparq project

*A resources / in-use contribution. The evidence is a set of deterministic, CI-enforced conformance ratchet floors (machine-independent scenario counts that may only rise); no wall-clock performance is claimed, and no security, soundness, or completeness property of the authorization engine is asserted. The conformance surface is scoped to* library-level decision parity *— the authorization content of the WAC and ACP specifications — explicitly* not *HTTP / Conformance-Test-Harness wire conformance, which has no library entry point here.*

## Abstract

The Solid ecosystem standardises two access-control models — Web Access Control (WAC) and the newer Access Control Policy (ACP) — and a server is expected to honour whichever a pod uses. We describe a library-level conformance signal for both: for each spec we maintain a corpus of minimal, independent scenarios, each isolating one normative construct (`acl:agent`, `acl:agentClass`, `acl:default` inheritance; ACP `anyOf` / `allOf` / `noneOf` matchers, policy `deny`-overrides, member-access inheritance), and assert the engine's `(agent, client, mode, resource) -> allow | deny` decision against the construct's specified semantics. We make *no* wall-clock claim and assert *no* security property of the engine. Instead the contribution is *conformance discipline* applied to access control: each corpus carries a monotone ratchet floor — a lower bound on the number of scenarios whose decisions all match the spec — that CI enforces and that may only rise, and both floors are registered as rows of the project's single cross-family conformance scoreboard alongside the SPARQL, inference, SHACL, and GeoSPARQL ratchets, with a guard test proving each declared floor stays in lock-step with the runner that enforces it. We are explicit about the scoping limitation: this is the authorization *content* of the specs at the library layer, not HTTP-shaped wire conformance.

## 1. Contributions

This paper substantiates the following claims; each is refutable and forward-references its evidence. None is a performance claim, and none asserts a security, soundness, or completeness property of the authorization engine.

- **Two access-control models, one decision interface** (§3) — WAC `.acl` documents and ACP access-control resources are both materialized into the same internal authorization index, so a single `(agent, client, mode, resource) -> allow | deny` decision procedure serves both models, and the same scenario shape tests each.
- **A per-construct, fail-closed-checked decision corpus per model** (§4) — each spec construct is isolated in a minimal scenario whose expected allow/deny decisions (including the anonymous and unauthorized-agent *deny* cases) are asserted; a scenario reports its mismatches independently so a regression is localised.
- **Each corpus carries a deterministic, CI-enforced ratchet floor** (§4) — Solid WAC decision parity is ratcheted at a floor of 12 passing scenarios and Solid ACP at 12, each a monotone lower bound that may only rise.
- **Both floors are rows of one cross-family ratchet, guarded against drift** (§5) — the project registers 7 conformance suites across crates in one central scoreboard with a combined floor of 3442, and a guard test reads each crate-local floor constant textually and asserts it equals the value the scoreboard declares — so the WAC and ACP rows cannot silently diverge from what CI enforces.

## 2. Two models, one decision interface

Solid resources are governed by an access-control description attached to the resource (or inherited from an ancestor container). WAC expresses this as an `.acl` document of `acl:Authorization` statements naming subjects (`acl:agent`, `acl:agentClass`, `acl:agentGroup`), modes (`acl:Read` / `acl:Write` / … via `acl:mode`), and scope (`acl:accessTo` for the resource itself, `acl:default` for descendants). ACP instead attaches an access-control resource whose policies combine *matchers* (`acp:anyOf` / `acp:allOf` / `acp:noneOf` over agent / client / issuer) with `acp:allow` / `acp:deny` mode sets, and inherits down a container's `acp:memberAccessControl`.

Implementing each faithfully is table-stakes parity with the Solid reference servers; it is *not* a research novelty, and we do not claim one. The only mild systems angle is that both models are compiled — `materialize_wac` for WAC, `materialize_acp` for ACP — into the *same* internal authorization index, so one decision procedure (`AuthIndex::accessible`) answers `(agent, client, mode, resource) -> allow | deny` for either. That shared decision interface is what lets a single scenario shape — a small access-control document plus a table of expected decisions for several principals — serve as the conformance unit for both specs.

## 3. A per-construct decision corpus, and its ratchet

We make *no* wall-clock claim, and we assert *no* security property: nothing here proves the engine is complete, that it admits no unsafe grant, or that it matches a reference server on every input. The conformance surface is narrower and exactly stated: for each spec we curate a corpus of minimal, *independent* scenarios, each isolating one normative construct, and assert that the engine's decision for each `(agent, client, mode, resource)` tuple matches the semantics that construct specifies.

The scenarios are derived from the specs' normative semantics, not vendored over HTTP from the Solid Conformance-Test-Harness corpus. That distinction is load-bearing and is the honest limitation of this signal: the harness corpus asserts on HTTP-shaped outputs (status codes, the `WAC-Allow` header) that have no library entry point here, so wire conformance is out of scope. What a library-level oracle *can* reproduce is the authorization *content* of those tests — the access-control document shapes and their expected allow/deny per principal — which is precisely what each scenario encodes. Each scenario also pins the fail-closed cases (the anonymous requestor and an unauthorized agent must be *denied*), so a corpus catches an over-permissive regression, not only an over-restrictive one. Scenarios fail independently and report all mismatches at once, so a regression localises to the offending construct.

The conformance signal is a *ratchet floor* per corpus: a monotone lower bound on the number of scenarios whose decisions all match the spec, which CI enforces and which may only rise.

| Suite | Ratchet floor | Basis |
|---|---:|---|
| Solid WAC decision parity | 12 | per-construct `.acl` allow/deny scenarios (library-level) |
| Solid ACP decision parity | 12 | per-construct ACR allow/deny scenarios (library-level) |

Table 1: The two access-control corpora's deterministic conformance floors: passing per-construct decision-parity scenarios. Each is a monotone lower bound a CI test enforces — machine- independent (the value of a `const` an assertion checks over a fixed scenario table), not a timing. Each floor may only rise.

evidence: `crates/sparq-solid/tests/conformance_wac.rs::WAC_SCENARIO_FLOOR` (mirrored in `crates/sparq-conformance/src/scoreboard.rs`; guarded by `tests/scoreboard_floors.rs`) (environment: canonical)

Each floor is deterministic and machine-independent: it is the value of a constant the corresponding corpus test asserts, identical on any host, with no dependence on wall-clock timing. That property is exactly why it can back a headline here while a latency number could not — and it is also why the

floor is a *coverage* guarantee (the corpus may not shrink), not a soundness or completeness guarantee about the engine.

## 4. One cross-family ratchet, guarded against drift

The WAC and ACP floors do not stand alone. The project maintains a *single central scoreboard*: a registry of every conformance suite it gates — across crates — each row carrying the spec family, the runner that executes it, the CI job that enforces it, and its ratchet floor. The scoreboard registers 7 suites with a combined floor of 3442 passing assertions; the two access-control corpora are the two newest rows:

| Suite | Family | Floor |
|---|---|---|
| W3C SPARQL (query + update + syntax) | W3C SPARQL | 1229 |
| Inference (rdf-mt / OWL 2 RL / N3 / entailment) | W3C RDF Semantics + OWL + N3 | 1967 |
| W3C SHACL core | W3C SHACL | 98 |
| W3C SHACL-SPARQL | W3C SHACL | 5 |
| OGC GeoSPARQL topology | OGC GeoSPARQL | 197 |
| Solid WAC decision parity | Solid WAC | 12 |
| Solid ACP decision parity | Solid ACP | 12 |
| **Total (7 suites)** | | **3442** |

Table 2: The cross-family conformance ratchet: every suite the project gates, in one registry, each with a monotone floor. The two access-control corpora (last two rows) are suites among 7 spanning six spec families. Each floor is a deterministic integer CI enforces and that may only rise.

The scoreboard is a *reporting* registry, not a merged runner: each suite's runner stays in the crate where its dependencies live — the WAC and ACP corpora run as crate-local tests in the Solid crate, so the central scoreboard crate does not take the Solid crate as a dependency. Consolidation happens at the reporting layer. To keep the central declaration honest, a guard test reads each crate-local runner's floor constant *textually* and asserts it equals the value copied into the scoreboard — so the two can never silently diverge: if a corpus raises its floor, the guard fails until the registry is updated to match. The cross-family total is therefore the sum of independently CI-enforced, drift-guarded floors, not a hand-maintained number.

## 5. Related work and honest positioning

WAC and ACP are implemented by the Solid reference servers and client libraries; we claim no advance over their authorization semantics, and the constructs tested here are exactly those the specs define. The genuinely-novel access-control composition this project also explores — bridging ODRL usage policies into a queryable WAC/ACP view, and the downstream ZK / MPC disclosure boundary — is reported elsewhere as honest design and is *not* part of this paper's evidence; in particular the cryptographic estate is research-grade and has not been externally audited, so no security property is claimed here. What this paper contributes is the conformance *discipline*: a per-construct, fail-closed-checked decision corpus for each of the two access-control models, expressed as a monotone ratchet that joins a single cross-family scoreboard. The methodology is engineering rigour rather than a research result; it is presented as the machine-checkable *evidence* for the in-use claim, and as the mechanism that lets an access-control suite join a cross-family floor without a bespoke harness. The honest limitation — library-level decision parity over the authorization content of the specs, not HTTP / CTH wire conformance — is stated, not papered over.

## 6. Conclusion

A pod server must honour whichever access-control model a resource uses, and an in-use claim for either deserves a machine-checkable conformance signal rather than a prose assurance. We give one for both Solid models at the library layer: a per-construct, fail-closed-checked decision corpus whose passing-scenario count is ratcheted — Solid WAC at a floor of 12 and Solid ACP at 12 — and registered as two rows of a single cross-family ratchet totalling 3442 across 7 suites, every floor deterministic, CI-enforced, monotone, and guarded against drift. We make no wall-clock claim and assert no security property; the floors are a coverage guarantee, and HTTP / CTH wire conformance is deferred.

---

sparq project · evidence traces to `crates/sparq-solid/tests/conformance_wac.rs` and `conformance_acp.rs`, the cross-family scoreboard in `crates/sparq-conformance/src/scoreboard.rs`, and its drift guard `tests/scoreboard_floors.rs`. Numbers in this document are injected at build time from the paper-bound evidence file; see the provenance stamp on the published page.