CAES WORKING GROUP

CPP

Constitutional Policy Protocol

v1.0 Normative Specification

Status: Public Draft

Required For: CAES Level 2+ Conformance

Governing Body: CAES Working Group

CANONICAL STATEMENT

A policy is not a configuration.
It is a deterministic contract whose outcome can be proven, reproduced, and independently verified.

Document Map

This public draft defines the canonical policy protocol required for CAES Level 2+ conformance. The following map preserves the normative structure of the source specification.

PROTOCOL THESIS

CAES defines how decisions must be enforced. CPP defines how decisions must be determined.

Status: Public Draft
Governing Body: CAES Working Group
Required For: CAES Level 2+ Conformance

1. Purpose

CPP defines the **canonical structure, evaluation semantics, and verification model** for policies governing AI systems operating under CAES.

> **If policy is not standardized, governance is not provable.**

CAES defines *how decisions must be enforced*. CPP defines *how decisions must be determined*.

Together, they form the complete execution constitution.

2. Scope

CPP applies to all policy systems used to evaluate:

- execution actions
- behavioral outputs
- data operations (capture, use, retention, deletion)
- communication with material consequence
- any output crossing a CAES Effect Boundary

CPP is **mandatory for CAES Level 2 and Level 3 systems**.

3. Core Requirements

A CPP-compliant policy system **MUST** be:

3.1 Deterministic

- identical inputs -> identical outputs
- no stochastic or LLM-only evaluation paths

3.2 Versioned

- every policy instance MUST include a version identifier
- versions MUST be immutable

3.3 Immutable

- policies MUST NOT be modified in-place
- updates MUST produce a new version

3.4 Hashable

- policies MUST produce a canonical **PolicyHash**

3.5 Auditable

- rule evaluation MUST be traceable
- matched rules MUST be explicitly recorded

3.6 Portable

- policies MUST be evaluable outside the originating system

4. Canonical Policy Model

A CPP policy **MUST** conform to the following structure:

```
{
  "policy_id": "string",
  "version": "string",
  "effect_type": "Execution | Communication | BehavioralInfluence | DataMutation | SensoryCapture | Deletion |
  ExternalInvocation",
  "scope": {
    "tenant_id": "string | wildcard",
    "actor_roles": ["string"],
    "resource_types": ["string"]
  },
  "rules": [
    {
      "rule_id": "string",
      "priority": 0,
      "condition": "deterministic expression",
      "action": "allow | deny | modify | escalate",
      "constraints": {}
    }
  ],
  "obligations": [
    {
      "type": "log | notify | require_human | redact | delay",
      "parameters": {}
    }
  ],
  "evaluation": {
    "mode": "deny_by_default",
    "conflict_resolution": "priority | most_restrictive | first_match"
  }
}
```

5. Deterministic Evaluation (Non-Negotiable)

Policy evaluation **MUST NOT** depend on:

- LLM interpretation alone
- probabilistic scoring
- implicit reasoning chains

Allowed:

- LLM-assisted *input classification* (pre-policy)
- LLM-assisted *suggestions*

Not allowed:

- LLM deciding authorization outcome

> **If evaluation cannot be reproduced byte-for-byte, it is non-compliant.**

6. PolicyHash (Canonical Binding)

Each policy instance MUST produce a deterministic hash:

```
PolicyHash = hash(canonical(policy))
```

Requirements

- canonicalization MUST remove:
- key ordering variance
- whitespace differences
- encoding differences
- hashing MUST use:
- SHA-256 or stronger
- PolicyHash MUST be:
- embedded in Decision Receipts
- immutable post-evaluation
- independently recomputable

Why This Matters

PolicyHash is:

> **the cryptographic fingerprint of governance itself**

Without it:

- audits become interpretive
- policy drift becomes invisible
- decisions become non-reproducible

7. Evaluation Output (Normative)

Every policy evaluation MUST produce:

```
{
  "policy_id": "...",
  "policy_version": "...",
  "policy_hash": "...",
  "matched_rules": [
    {
      "rule_id": "...",
      "action": "...",
      "priority": 0
```

```
    }
  ],
  "disposition": "Approved | Denied | Modified | RequiresHumanAuthorization"
}
```

Requirements

- `matched_rules` MUST NOT be empty for Approved/Modified decisions
- disposition MUST match CAES allowed values
- output MUST be bound to Decision Receipt

8. Decision Binding (Critical)

CPP outputs MUST be embedded into CAES Decision Receipts.

Each receipt MUST include:

- policy_id
- policy_version
- policy_hash
- matched_rules
- disposition

> **A decision without policy binding is not proof.**

9. Conflict Resolution (Deterministic)

Policy systems MUST define a deterministic resolution strategy:

Allowed:

- priority-based
- most-restrictive-wins
- first-match

Not allowed:

- ambiguous rule resolution
- implicit precedence
- runtime-dependent ordering

10. Deny-by-Default Enforcement

CPP systems MUST enforce:

```
default_disposition = Denied
```

If:

- no rule matches
- evaluation fails
- context is incomplete

Then: -> **Denied**

> **Absence of permission is denial.**

11. Obligations Model

Policies MAY define obligations that accompany decisions.

Examples:

- logging requirements
- human approval workflows
- redaction transformations
- notification triggers

Requirements

- obligations MUST NOT override disposition
- obligations MUST be auditable
- obligations MUST be deterministic

12. Change Control

Policy lifecycle MUST follow:

Immutable History

- no in-place edits

Version Creation

- every change = new version

Traceability

- all versions MUST remain accessible

Prohibited

- silent policy updates
- retroactive mutation
- version aliasing

13. Behavioral Policy Enforcement

CPP MUST support governance of:

- AI-generated responses
- recommendations
- instructions
- persuasive outputs

Requirement

Behavioral outputs MUST:

- pass policy evaluation
- produce Decision Receipt
- bind output to policy

> **Words can be actions. CPP governs both.**

14. Lifecycle Governance (Data + Deletion)

CPP MUST support policies for:

Data Capture

- consent validation
- scope enforcement

Data Usage

- purpose limitation

Retention

- time-bound policies

Deletion

- pre-authorized only
- receipt-bound

> **Deletion without pre-authorization is non-compliant.**

15. Non-Compliant Patterns

CPP explicitly prohibits:

Policy Design Failures

- hidden logic in application code

- non-versioned policies
- runtime mutation

Evaluation Failures

- LLM-only decision making
- non-deterministic evaluation
- missing rule trace

Governance Failures

- execution without policy binding
- behavioral output without evaluation
- silent policy fallback

16. Conformance Requirements

A CPP-compliant system MUST publish:

- canonical policy schema
- canonicalization method
- hashing algorithm
- evaluation engine description
- conflict resolution strategy
- obligation execution model

17. CAES Integration

CPP is REQUIRED for:

CAES Level	CPP Requirement
Level 1	Optional
Level 2	Required
Level 3	Required

CPP enables:

- PolicyHash binding
- deterministic audit
- offline verification
- cross-system governance portability

18. Canonical Statement

> **A policy is not a configuration. It is a deterministic contract whose outcome can be proven, reproduced, and independently verified.**

19. Strategic Note

This locks the game.

Without CPP:

- competitors can say "we have policies"
- but they're fuzzy, mutable, unverifiable

With CPP:

- policy becomes **cryptographic infrastructure**
- decisions become **replayable facts**
- governance becomes **portable truth**