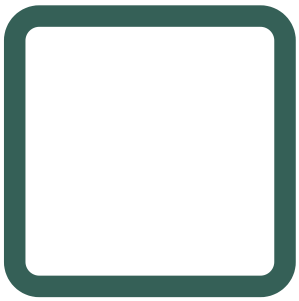
Quuipu

QUUIPU RESEARCH & ARCHITECTURE SERIES

WHITEPAPER 10

Web Protocol

Developer & Integration Specification



Technical / Specification Whitepaper · English

Quuipu Edition 2 · v1

Publication identity

Publication
Web Protocol

Series
Quuipu Research & Architecture Series

Publication number
Whitepaper 10

Edition
Quuipu Edition 2 · v1

Language
English

Historical edition
Cognisphere Edition 1

This publication succeeds Cognisphere Edition 1. Historical PDFs remain immutable.

Contents

01. Why a protocol

02. Protocol architecture

03. Protocol stack

04. Resource model

05. API conventions

06. Context API

07. Context schemas

08. Experience API

09. Experience Plan schema

10. Semantic API

11. Host API

12. Source API

13. SDK design

14. Authentication

15. Authorization and permissions

16. Policy API

17. Events

18. Event catalog

19. Webhooks

20. Telemetry

21. Audit and decision trace

22. Errors

23. Rate limits and quotas

24. Versioning

25. Schema evolution

26. Pagination, filtering and search

27. Long-running operations

28. Web security

29. Privacy and data boundaries

30. Multi-tenant protocol

31. Web SDK integration pattern

32. Server integration pattern

33. Webhook integration pattern

34. Developer tooling

35. Testing and conformance

36. Developer experience

37. Governance of the protocol

38. Protocol invariants

39. Roadmap to a Developer Ecosystem

40. The protocol principle

A. Illustrative endpoint catalog

B. Canonical error model

C. Webhook verification sequence

D. Protocol invariants

Preamble

Developer & Integration Specification

> This publication succeeds Cognisphere Edition 1. Quuipu Edition 2 preserves the historical thesis while updating active brand and publication identity.

Preamble

Developer & Integration Specification

PROTOCOL MAP

A stable boundary for the Quuipu ecosystem The protocol separates integration, cognition and rendering through versioned APIs and schemas.

Application / Website ↓

SDK or Direct Client ↓

Host API ↓

Context · Semantic · Experience APIs ↓

↓

Sources · State · AI/Model Adapters

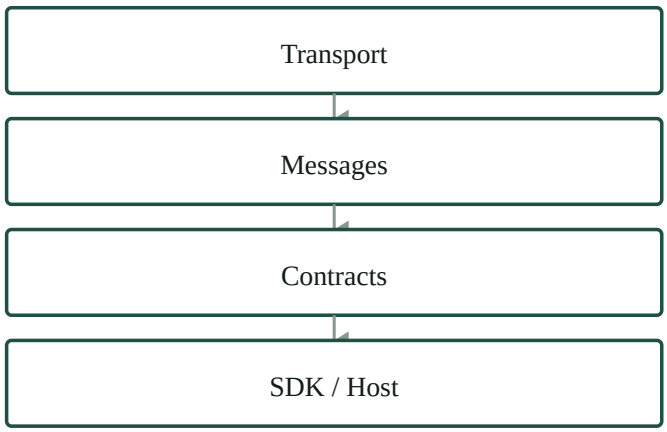


Figure 1 — Quuipu Web Protocol stack.

API PLANES

Public protocol surfaces Plane Primary resources Purpose

Host Host, Capabilities, Source Bind runtime to a digital surface

Context ContextItem, Intent, Goal, Session Represent and transition bounded context

Semantic Concept, Transformation, SourceIdentity Preserve meaning and provenance

Experience ExperienceRun, ExperiencePlan, Component Compose adaptive presentation

Events Event, Subscription, Delivery Asynchronous integration

Governance Actor, Permission, Policy, Trace Control access and behavior

CHAPTER 01

Why a protocol

Quuipu becomes a developer ecosystem only when its capabilities can be integrated through stable, versioned and governed contracts. The Web Protocol defines that boundary without turning Quuipu into a CMS or coupling integrators to its internals.

Specification thesis The protocol describes observable objects, lifecycles, commands, queries, events and policies. Core implementation may evolve as long as public contracts remain stable.

Not an implementation leak Context resolution, semantic algorithms and internal reasoning are not exposed as client-side orchestration details.

Host independence A website, portal, application or SDK can consume Quuipu through the same conceptual model.

Draft status This V1 substantive edition is a specification architecture: it proposes the normative shape to consolidate once the real API is frozen.

CHAPTER 02

Protocol architecture

The protocol organizes integration into separate Host, Context, Semantic, Experience, Events and Governance planes.

Host plane Lifecycle, capabilities, tenant binding, source registration and surface integration.

Context plane Declared context, intent, goal, session, state projection, transitions and portability.

Semantic plane Concept resolution, source identity, transformations, provenance and validation.

Experience plane Experience run, modes, plans, components, feedback, fallback and resume.

Event plane Asynchronous lifecycle and domain events with stable envelopes.

Governance plane Authentication, authorization, permissions, policies, telemetry controls, versioning and audit.

Host Plane ↓

Context Plane ↓

Semantic Plane ↓

Experience Plane ↓

Event Plane ↓

Governance Plane

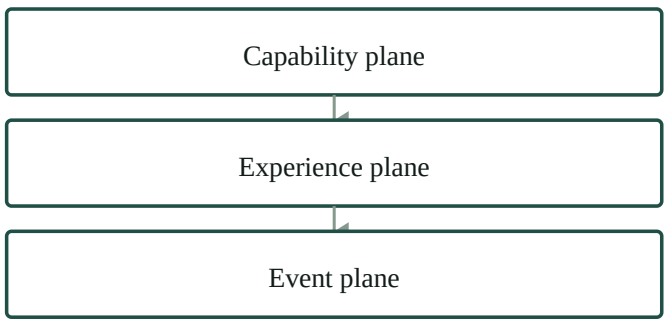


Figure 2 — Protocol planes.

CHAPTER 03

Protocol stack

SDK and API are two surfaces of the same protocol: the SDK simplifies consumption but does not create parallel semantics.

Stack Application -> SDK/Client -> Host API -> Context/Semantic/Experience APIs -> Quuipu Runtime -> Source/Model/State adapters.

Thin SDK principle The SDK handles serialization, retries, type safety, event subscriptions and ergonomic helpers; cognitive decisions remain runtime-side.

Direct API Integrators may use HTTP/event transports directly when they do not want an SDK.

Conformance Official and third-party SDKs must pass the same contract tests.

CHAPTER 04

Resource model

APIs operate on explicit resources with identity, tenant scope, version and lifecycle.

Core resources Tenant, Host, Source, Session, ContextItem, Intent, Goal, ExperienceRun, ExperiencePlan, Component, SemanticConcept, Transformation, EventSubscription.

Identifiers IDs are opaque and stable; they do not encode PII or business semantics.

References Resources link through typed refs rather than embedding uncontrolled copies.

Lifecycle Every resource declares valid states and permitted transitions.

Resource Scope Typical lifecycle

Session tenant/host created -> active -> suspended -> closed

ContextItem session/task/tenant declared/inferred -> valid -> expired

ExperienceRun tenant/host/session started -> composed -> presented -> completed

ExperiencePlan run valid -> superseded/expired

Transformation source/concept candidate -> validated/rejected -> invalidated

Subscription tenant active -> paused -> revoked

CHAPTER 05

API conventions

Uniform conventions reduce ambiguity and make SDKs generatable and testable.

Base shape /v1/{tenantScope}/{resource} is illustrative; final URI design depends on deployment.

Commands vs queries Semantic mutations use explicit commands; queries do not create cognitive side effects.

Idempotency Retryable commands accept Idempotency-Key or equivalent.

Concurrency ETag/resourceVersion supports optimistic concurrency and prevents lost updates.

Correlation requestId, runId and traceId correlate calls, events and observability.

CHAPTER 06

Context API

The Context API exposes Context Model operations without permitting arbitrary mutation of a session blob.

Commands declareContext, activateIntent, setGoal, transitionContext, expireContext, resetContext, importContextPackage.

Queries getResolvedContext, getActiveIntent, getGoal, getContextTrace, exportContextProjection.

Projection first Consumers receive authorized projections, not all available context.

Precedence Declared context cannot be silently overwritten by inferred context through the API.

Operation class Examples Invariant

Declare declareContext Origin remains declared

Resolve getResolvedContext Returns authorized projection

Transition transitionContext Version/policy checked

Export exportContextProjection Minimized + audience-bound

Import importContextPackage Candidate until local resolution

Expire expireContext Derived dependencies invalidated

CHAPTER 07

Context schemas

ContextItem and ContextPackage make context interoperable across SDK, host and runtime.

ContextItem

`{id, kind, value, origin, scope, confidence?, observedAt, effectiveUntil?, expiresAt?, purpose, sensitivity, version}.`

ContextPackage

`{schemaVersion, packageId, issuer, audience, tenantScope, projection, sourceRefs, issuedAt, expiresAt, policyRef, integrity}.`

Typed values kind determines the value schema; free-form metadata does not replace normative fields.

Validation Unknown fields follow an explicit forward-compatibility policy rather than free interpretation.

CHAPTER 08

Experience API

The Experience API manages the complete lifecycle of an adaptive experience run.

Commands startExperience, submitInput, selectMode, chooseNode, submitFeedback, resetExperience, resumeExperience, closeExperience.

Queries getExperienceRun, getExperiencePlan, getAvailableModes, getFallbackState, getExperienceTrace.

Run scope Every run belongs to a defined tenant, host, session and source scope.

Canonical access Even when enhancement fails, the API can return canonicalSourceRef/fallback.

startExperience ↓

submitInput / selectMode ↓

getExperiencePlan ↓

render ↓

submitFeedback ↓

context/state update ↓

complete / continue

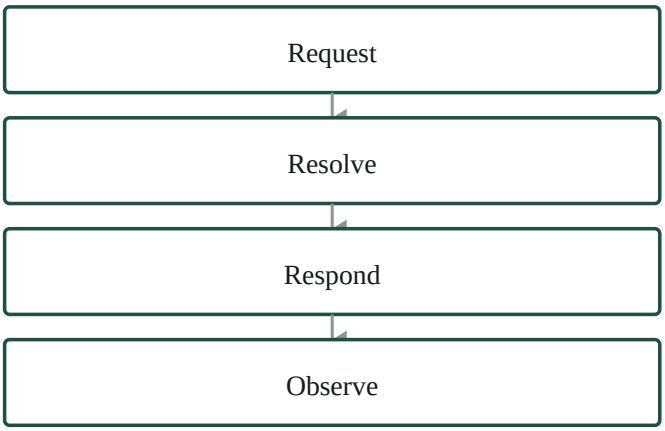


Figure 3 — Experience API lifecycle.

CHAPTER 09

Experience Plan schema

The Experience Plan is the runtime's principal public output and separates cognitive composition from rendering.

Plan

`{experienceId, runId, mode, resolvedContextRef, intentRef, goalRef?, sourceSet, semanticViewRef, components, provenance, trustState,`

fallbackRef, expiresAt}.

Components Each component has type, purpose, contentRef/payload, sourceRefs, trustState, interactions, guards and fallback.

Graph Components may be linear or nodes in an Experience Graph with allowed transitions.

Surface neutrality The protocol does not mandate DOM, React, Astro or any other framework.

CHAPTER 10

Semantic API

The Semantic API resolves concepts and validates transformations without exposing proprietary internals.

Queries resolveConcept, getConcept, getConceptRelations, getSourceIdentity, getTransformationRecord.

Commands validateTransformation, registerSourceMapping, invalidateSemanticArtifact.

Semantic identity Concept ID and source identity remain stable across derived representations.

Boundary The API returns outcomes, evidence refs and reason codes, not chain-of-thought.

CHAPTER 11

Host API

The Host API connects Quuipu to the digital surface embedding it.

Lifecycle create/configure/initialize/start/ready/shutdown/dispose can be represented as a capability/lifecycle contract.

Capabilities Host declares rendering, navigation, locale, auth delegation, storage, source access and accessibility capabilities.

Host context The runtime receives HostCapabilities and HostSecurityContext, not ambient browser/server access.

Isolation A host adapter cannot bypass tenant, source or policy boundaries.

CHAPTER 12

Source API

CMS, knowledge base and other systems integrate through source contracts rather than content ownership.

Registration registerSource describes authority class, connector type, schema, freshness, canonical URL/ref and tenant scope.

Read contract The runtime may request content units, metadata, versions and evidence refs through adapters.

Write boundary Quuipu does not write to the canonical source unless an explicit authorized capability exists.

Invalidation Source update events invalidate dependent semantic artifacts, caches and Experience Plans.

CHAPTER 13

SDK design

SDKs should make the correct path easy without hiding protocol trust properties.

Core modules client, context, experience, semantic, events, auth, types and testing utilities.

Typed errors ContextExpiredError, PolicyDeniedError, VersionConflictError, UnsupportedModeError and RateLimitError are preferable to generic errors.

Async model Promise/future for commands and queries; async iterator/observable for event streams where supported.

No secret state The SDK does not maintain a second proprietary Context Model on the client.

CHAPTER 14

Authentication

Authentication establishes who is calling the protocol; it does not by itself decide what the caller may do.

Actors Service application, end-user delegated session, administrator, connector, webhook receiver.

Credentials OAuth/OIDC, signed service credentials or deployment-specific mechanisms may implement the contract.

Token minimization Tokens should carry minimal identity/scopes and references, not sensitive context payloads.

Rotation Credential rotation and revocation must be supported without invalidating semantic identities.

CHAPTER 15

Authorization and permissions

Authorization combines actor, tenant, resource, action, scope and policy.

Permission tuple Permit(actor, action, resource, tenant, purpose, contextClass) -> allow/deny/conditional.

Scopes context:read, context:declare, experience:run, semantic:read, source:register, events:subscribe are illustrative scopes.

Fine-grained control An application may start experiences without permission to export context or read audit trace.

Deny by default Cross-tenant, admin, source-write and sensitive-context permissions are not implicit.

Scope example Allows Does not imply

experience:run Start/continue runs Context export

context:declare Declare bounded context Read all context

semantic:read Resolve concepts Register sources

events:subscribe Receive permitted events Admin trace access

source:register Register source metadata Write canonical content

CHAPTER 16

Policy API

Policy is not an optional header; it is a versioned dependency of runtime decisions.

Policy resources TenantPolicy, ExperiencePolicy, DataPolicy, AIUsePolicy, SourcePolicy and TelemetryPolicy.

Evaluation policy/evaluate may return allow, qualify, require_confirmation, require_review or deny with reason codes.

Version pinning ExperienceRun records policyVersion for reproducibility.

No client override A client may request a mode; it cannot disable server-side policy.

CHAPTER 17

Events

Events decouple runtime, host, permitted analytics and downstream integrations.

Envelope

`{eventId, type, schemaVersion, tenantId, hostId, sessionId?, runId?, timestamp, actorRef, payloadRef/payload, reasonCode?, traceId}.`

Families context.*, intent.*, semantic.*, source.*, experience.*, policy.*, webhook.*.

Delivery semantics At-least-once delivery is compatible with idempotent consumers; ordering is guaranteed only within declared partitions.

Privacy Event payloads follow minimization and must not become copies of context state.

Event field Role

eventId Idempotency/replay identity

type Namespaced event semantics

schemaVersion Payload compatibility

tenantId / hostId Isolation scope

sessionId / runId Optional lifecycle correlation

timestamp Event time

actorRef Originating actor

payloadRef/payload Minimized event data

reasonCode Operational explanation

traceId Cross-plane correlation

CHAPTER 18

Event catalog

A stable event catalog lets partners and tools react without polling.

Context context.declared, context.updated, context.expired, context.transferred.

Experience experience.started, experience.composed, experience.presented, experience.fallback, experience.completed.

Semantic semantic.resolved, transformation.validated, transformation.rejected.

Source source.updated, source.invalidated, source.unavailable.

Policy policy.qualified, policy.blocked, review.required.

CHAPTER 19

Webhooks

Webhooks carry Quuipu events to external systems under an explicit security model.

Subscription endpoint, event filters, tenant scope, secret/public key, status and delivery policy.

Signing Every delivery includes timestamp, event ID and a verifiable signature.

Replay protection Receiver checks freshness, signature and previously processed eventId.

Retries Backoff, dead-letter state and delivery attempt metadata must be observable.

No inbound command by default A webhook is outbound delivery; inbound commands require a separate authenticated API.

Runtime Event ↓

Filter ↓

Signed Webhook Delivery ↓

Receiver Verification ↓

Idempotent Processing ↓

ACK / Retry / Dead Letter

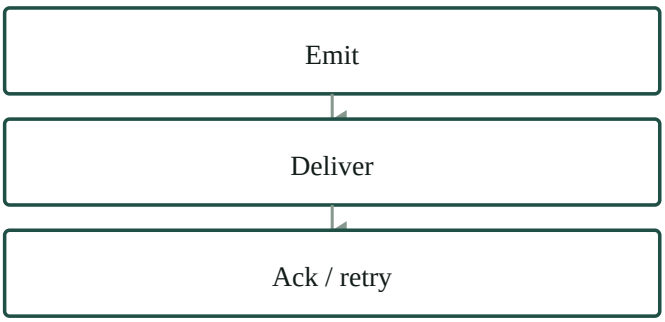


Figure 4 — Webhook delivery model.

CHAPTER 20

Telemetry

Telemetry measures protocol health and quality without becoming user surveillance.

Planes Operational, reliability, quality, security and cost telemetry.

Examples Latency by endpoint/stage, error rate, fallback rate, schema rejection, webhook delivery and AI invocation cost.

Excluded by default Raw prompts, full context, sensitive values and full content bodies are not metrics by default.

Tenant controls TelemetryPolicy may disable classes, reduce retention or impose regional processing.

CHAPTER 21

Audit and decision trace

Auditability requires separating access logs, protocol traces and cognitive decision traces.

Access log Who called which endpoint and with what outcome.

Protocol trace Request/event correlation, versions, retries and dependency status.

Decision trace Source refs, policy decisions, reason codes, transformations and fallback used by the runtime.

No hidden reasoning No trace exposes model chain-of-thought; it records verifiable decisions.

CHAPTER 22

Errors

Errors are part of the protocol and must indicate whether a caller can retry, correct or fall back.

Error envelope

`{code, message, category, retryable, requestId, traceId, details?, remediation?}.`

Categories AUTH, PERMISSION, VALIDATION, CONFLICT, EXPIRED, POLICY, SOURCE, SEMANTIC, EXPERIENCE, RATE_LIMIT, INTERNAL_SAFE.

Safe details details do not include secrets, raw model prompts or other-tenant data.

Fallback hint For cognitive errors the server may provide canonicalSourceRef or safeMode.

CHAPTER 23

Rate limits and quotas

Limits protect availability, cost and fairness without changing protocol semantics.

Dimensions Requests, concurrent runs, event throughput, AI-assisted operations, source sync and webhook deliveries.

Headers/metadata Remaining, reset and retry-after may be exposed through standard metadata.

Tenant quotas Quota may vary by plan but must not weaken trust invariants.

Graceful degradation Exhausting AI quota can degrade to deterministic/canonical paths where possible.

CHAPTER 24

Versioning

Versioning must support multi-year evolution without forcing all integrators to migrate simultaneously.

Layers Protocol version, resource schemaVersion, event schemaVersion, policyVersion and SDK version are distinct.

Compatibility Additive optional fields are preferred; semantic breaking changes require a major version.

Deprecation Announce -> dual support -> migration tooling -> sunset with documented windows.

Pinning Critical integrations can pin compatible versions by run or client.

Version layer Example Breaking-change rule

Protocol v1 Major for semantic break

Resource schema schemaVersion=3 Migration/compatibility policy

Event schema event schemaVersion Consumer compatibility

Policy policyVersion Pinned to decision/run

SDK sdk 1.x May advance independently within protocol compatibility

CHAPTER 25

Schema evolution

Schemas are public contracts and need machine-testable compatibility rules.

Canonical schemas JSON Schema/OpenAPI/AsyncAPI or equivalents can represent REST/event contracts.

Unknown fields Robust consumers ignore non-critical additive fields, but not unknown enum values when materially relevant.

Migration Versioned transformers convert persisted payloads or portable packages.

Examples as tests Published examples are validated in CI against the real schema.

CHAPTER 26

Pagination, filtering and search

Collection APIs must be predictable and bounded.

Pagination Cursor-based pagination avoids instability with dynamic datasets.

Filtering Supported filters are explicit; there is no arbitrary query over internal state.

Search Semantic search may be a separate capability with source scope and policy.

Limits Server enforces maximum page size and complexity budget.

CHAPTER 27

Long-running operations

Source ingestion, large semantic validation or exports may require asynchronous operations.

Operation resource

`{operationId, type, status, progress?, resultRef?, error?, createdAt, expiresAt}.`

States queued, running, succeeded, failed, cancelled.

Polling/events Client may poll the operation or subscribe to operation.* events.

Cancellation Cancellation is best-effort and does not undo already committed side effects.

CHAPTER 28

Web security

The protocol must be integrable on the Web without relying on ambient privileges.

Browser clients CORS, CSRF protections, PKCE/delegated tokens and origin restrictions according to deployment.

Secrets Service secrets must not be exposed in the browser.

Content security Rendered components respect host CSP and safe-link policy.

Input security Prompt injection and malicious source content are untrusted input, not system instructions.

CHAPTER 29

Privacy and data boundaries

Data minimization is embedded in schemas and permission checks.

Classification public, tenant-confidential, user-contextual, sensitive-restricted are illustrative classes.

Residency Deployment may apply region/processor constraints through policy.

Retention Event, trace, context and telemetry have separate retention classes.

Deletion Delete/revoke invalidates future projections and derived artifacts where required.

CHAPTER 30

Multi-tenant protocol

Tenant scope exists before every resolution, query, event or delivery.

Namespace Resource IDs are resolved within the tenant namespace.

Cross-tenant There is no generic cross-tenant API; exchange goes through authorized export/import or gateway.

Admin plane Operator administration is separated from application APIs.

Testing The conformance suite includes negative tests for tenant leakage.

CHAPTER 31

Web SDK integration pattern

A typical Web integration remains progressive: the canonical site works first and Quuipu activates as enhancement.

Bootstrap Host configures tenant, locale, source scope and capabilities.

Start SDK opens session/run and declares minimum context.

Render Experience Plan maps to host components.

Events UI actions produce protocol commands; they do not directly mutate runtime state.

Fallback If SDK/API is unavailable, the site continues serving canonical content.

Canonical Website ↓

Host Bootstrap ↓

Context API ↓

Experience API ↓

Experience Plan ↓

Host Components ↓

Protocol Events ↓

Canonical Fallback

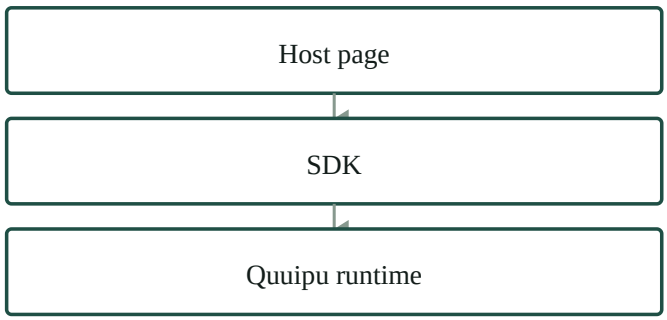


Figure 5 — Progressive Web SDK integration.

CHAPTER 32

Server integration pattern

Server-side integrations can orchestrate source access, authorization and rendering without exposing credentials to the browser.

BFF A Backend-for-Frontend may exchange tokens and apply tenant policy.

SSR Experience Plan may be obtained server-side for initial rendering, with optional hydration.

Connector Source adapters may live in the customer's trust domain.

Isolation The host server receives no Quuipu internals beyond public contracts.

CHAPTER 33

Webhook integration pattern

Enterprise systems can react to events without entering the synchronous runtime path.

Example review.required -> workflow system -> human approval -> authenticated API command -> experience artifact released.

Loose coupling The consumer may be unavailable without blocking presentation unless policy requires approval.

Replay Event ID and delivery log support deterministic recovery.

Boundary Webhook is not a backdoor for changing context or policy.

CHAPTER 34

Developer tooling

A healthy ecosystem needs tools that turn the specification into immediate feedback for integrators.

CLI Validate config, schemas, permissions, webhook signatures and conformance fixtures.

Sandbox Isolated tenant with synthetic sources and deterministic fixtures.

Inspector Displays Experience Plan, source refs, reason codes, policy outcomes and event timeline.

Code generation Typed clients may be generated from canonical schemas without losing semantic helpers.

Tool Developer value

CLI Validate config, schemas, permissions, signatures

Sandbox Safe synthetic integration environment

Inspector See plans, sources, policy and event timeline

Schema registry Canonical machine-readable contracts

Conformance runner Prove compatibility against a protocol version

CHAPTER 35

Testing and conformance

Compatibility is not asserted; it is demonstrated through public contract and invariant tests.

Contract tests Request/response/event schemas, auth failures, idempotency and version conflicts.

Invariant tests No expired context, no unauthorized source, provenance requirements, tenant isolation, safe fallback.

Golden fixtures Known context + sources + policy -> expected plan/event sequence.

Certification An SDK/connector may claim Quuipu compatibility only against a conformance version.

CHAPTER 36

Developer experience

Developer experience should expose Quuipu concepts rather than hide them behind one magic function.

Progressive path Start with canonical host -> add Context API -> Experience API -> events -> advanced semantic/source integrations.

Local clarity Errors and reason codes should identify the violated boundary.

Docs Reference, guides, recipes and conceptual architecture are separate but cross-linked.

Examples Examples always show auth, tenant scope, fallback and cleanup, not only happy paths.

CHAPTER 37

Governance of the protocol

A protocol intended for partners requires explicit evolution governance.

Change process Proposal -> review -> experimental -> stable -> deprecated.

Normative language MUST/SHOULD/MAY will be used in the future normative specification.

Extension registry New event types, component types and capabilities use namespaces and registration rules.

Security process Vulnerability handling and security advisories are part of the developer ecosystem.

CHAPTER 38

Protocol invariants

Invariants define what no SDK, host or version may bypass.

P-01 Tenant scope precedes every resource operation.

P-02 Declared context cannot be silently overwritten by inference.

P-03 Experience enhancement never removes canonical fallback.

P-04 Derived representations preserve required provenance.

P-05 Permissions are deny-by-default for sensitive and cross-tenant actions.

P-06 AI/provider state is not canonical Quuipu state.

CHAPTER 39

Roadmap to a Developer Ecosystem

The Web Protocol is the starting point for moving Quuipu from an integrable product to an extensible platform.

Phase 1 Canonical APIs + official Web SDK + schemas + sandbox.

Phase 2 Events/webhooks + connector SDK + conformance suite.

Phase 3 Component/experience extension registry + partner certification.

Phase 4 Multi-language SDKs, marketplace/registry governance and enterprise developer tooling.

Constraint Ecosystem growth must not weaken semantic integrity, context boundaries or human control.

CHAPTER 40

The protocol principle

The protocol must make Quuipu composable without making cognitive behavior arbitrary.

Stable contracts Developers integrate against resources and semantics, not implementation internals.

Bounded power Permissions, policy and schemas constrain every extension.

Observable evolution Versions, events and traces make change inspectable.

Interoperability Context, semantics and experience can cross surfaces through explicit contracts.

Final thesis A developer ecosystem becomes trustworthy when integration is open at the boundary and governed at the core.

CHAPTER A

Illustrative endpoint catalog

The paths below are architectural examples, not a claim that these endpoints are already frozen or deployed.

Method Illustrative path Purpose

`POST /v1/experiences Start ExperienceRun`

`POST /v1/experiences/{id}/inputs Submit bounded input`

`GET /v1/experiences/{id}/plan Read current ExperiencePlan`

`POST /v1/context/declarations Declare context`

`GET /v1/context/resolved Read authorized resolved projection`

`POST /v1/context/export Create portable projection`

`POST /v1/semantic/resolve Resolve concept/meaning`

`POST /v1/semantic/transformations/validate Validate derived representation`

`POST /v1/sources Register source contract`

`POST /v1/event-subscriptions Create event/webhook subscription`

CHAPTER B

Canonical error model

Code family Retry? Typical remediation

AUTH_* Usually no Refresh/re-authenticate

PERMISSION_* No Request scope/change policy

VALIDATION_* No Correct request/schema

VERSION_CONFLICT Yes after read Reload resource/version

CONTEXT_EXPIRED Conditional Re-declare/re-resolve

POLICY_DENIED No Use permitted mode/canonical

SOURCE_UNAVAILABLE Yes/conditional Retry or canonical fallback

SEMANTIC_UNCERTAIN No blind retry Qualify/use source

RATE_LIMITED Yes Honor retry-after

INTERNAL_SAFE Conditional Retry/fallback with traceId

CHAPTER C

Webhook verification sequence

Receive HTTPS request ↓

Read timestamp + eventId ↓

Verify signature ↓

Check timestamp tolerance ↓

Check eventId replay store ↓

Validate schemaVersion ↓

Process idempotently ↓

Return success

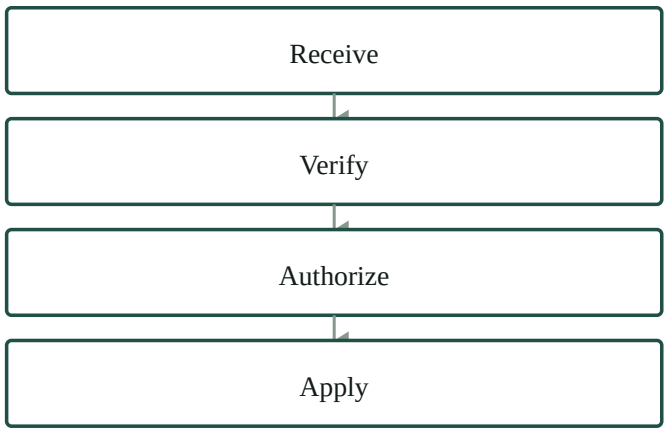


Figure 6 — Recommended receiver verification sequence.

Webhook payloads SHOULD be treated as untrusted until signature, freshness and schema validation succeed.

CHAPTER D

Protocol invariants

P-01 · Tenant first Tenant scope is established before resource resolution or retrieval.

P-02 · Context precedence Declared context is not silently overwritten by inference.

P-03 · Projection by purpose Consumers receive only context required and permitted for the task.

P-04 · Canonical fallback Cognitive enhancement cannot remove access to authoritative content.

P-05 · Provenance survives Required source and transformation references remain attached to derived experience.

P-06 · Policy is server-authoritative Clients cannot bypass policy through request parameters.

P-07 · Permissions default to deny Sensitive, administrative and cross-tenant actions require explicit grants.

P-08 · Events are minimized Event streams are not replicas of full context or content state.

P-09 · AI state is not protocol state Provider conversation memory is not canonical Quuipu state.

P-10 · Version is explicit Resources, events and policies declare compatibility-relevant versions.

P-11 · Webhooks are signed External event delivery is authenticated and replay-resistant.

P-12 · Conformance is testable Compatibility is tied to machine-verifiable contracts and invariants.

APPENDIX E

Conformance profile Profile Required capabilities

HOST-BASE Host lifecycle, tenant binding, canonical fallback

CONTEXT-BASE Declare, resolve, expire, projection, trace

EXPERIENCE-BASE Start run, submit input, plan, mode, feedback, fallback

SEMANTIC-BASE Concept/source identity, transformation validation

EVENTS-BASE Event envelope, idempotent consumer, version handling

WEBHOOK-BASE Signing, replay protection, retry handling

SECURITY-BASE AuthN, AuthZ, tenant isolation, secret handling

OBSERVABILITY-BASE requestId/runId/traceId, safe metrics and errors

APPENDIX F

Relationship to the Quuipu series Whitepaper Protocol relationship

01 · The Cognitive Web Defines the category and human-centered objective.

02 · Architecture Defines Core/Host boundaries exposed through contracts.

03 · Quuipu Web Provides the primary integration surface.

04 · Semantic Integrity Defines provenance and transformation invariants.

05 · Trust Architecture Defines policy, permissions, trace and bounded intelligence.

06 · Quuipu AI Defines AI Gateway as an optional provider boundary.

07 · Applications Supplies domain-specific integration profiles.

08 · Context Model Defines Context API semantics and portability.

09 · Experience Runtime Defines ExperienceRun and ExperiencePlan semantics.

10 · Web Protocol Turns the architecture into a developer-facing contract model.

APPENDIX G

Normative specification roadmap Artifact Future normative role

OpenAPI document HTTP request/response contracts

AsyncAPI/event schemas Events and webhook contracts

JSON Schemas Resource and portable payload validation

SDK reference Language-specific ergonomic binding

Conformance suite Machine-verifiable compatibility

Security profile Auth, signing, token and browser requirements

Extension registry Namespaced components, events and capabilities

Change policy Lifecycle from experimental to deprecated

This edition does not claim the API is already frozen: it prepares the structure that can become normative once real endpoints, schemas and transports are implemented and certified.

Developer & Integration Specification

OPEN AT THE BOUNDARY. GOVERNED AT THE CORE. VERSIONED BY CONTRACT.