RFC-0001: AGENTOBS â€" Observability Schema Standard for Agentic AI Systems

RFC-0001: Observability Schema Standard for Agentic AI Systems

AGENTOBS Community Draft S. Sriram
Document: RFC-0001 SpanForge
Category: Specification Draft April 3, 2026
Status: Public Review

Status of This Memo

This document is a proposed open specification for observability in agentic AI systems, published for public technical review and feedback. Distribution of this memo is unlimited.

This is a **Draft Specification**. This document is the product of open community collaboration. Please send feedback, objections, and implementation reports to the Issues tracker at https://github.com/veerarag1973/Spanforge/issues.

This document is **not** an IETF RFC and is not published by the IETF. The term â€œRFCâ€ in this repository denotes an open request-for-comments process operated by the SpanForge community.

Copyright Notice

Abstract

Agentic AI systems â€" applications in which large language models (LLMs) orchestrate multi-step workflows, invoke external tools, delegate to sub-agents, and engage in iterative reasoning â€" produce observability data that is fundamentally different from, and substantially more complex than, traditional distributed-systems telemetry.

Yet today there is no broadly adopted cross-vendor standard for what an â€œAI observability eventâ€ looks like: what fields it carries, how it identifies its place in a multi-agent trace tree, how cost is attributed across nested steps, how personally identifiable information (PII) is handled before data reaches a backend, or how the integrity of an audit trail is guaranteed.

This RFC defines **AGENTOBS** â€" an open event-schema standard for observability of agentic AI systems. AGENTOBS specifies:

1. A structured, typed **Event Envelope** that every LLM-adjacent instrumentation tool can emit and every observability backend can consume.

2. A **Namespace Taxonomy** of thirty-six event types spanning eleven domains â€" ten observability domains (trace, cost, cache, eval, guard, fence, prompt, redact, diff, template) and one security domain (audit).
3. A first-class **Agent Span Hierarchy** for representing multi-step runs, tool invocations, reasoning steps, and decision points as an OpenTelemetry-compatible span tree.
4. A vendor-neutral **Provider Normalisation Protocol** that maps raw API responses from any LLM provider to the canonical schema in one call.
5. **HMAC-SHA256 Audit Chains** for tamper-evident, compliance-grade event integrity.
6. A **PII Redaction Framework** that enforces privacy before event data enters any log pipeline.
7. An **Export Abstraction** compatible with OTLP, Datadog, Grafana Loki, webhook, and JSONL.
8. **Governance, Deprecation, and Migration** primitives for managing schema evolution across a distributed ecosystem of tools.
9. Four **Conformance Profiles** (Core, Security, Privacy, Enterprise) that allow incremental adoption â€" from basic standardised AI spans to full compliance-grade audit and governance.

An open reference implementation exists in the SpanForge project (Python), and independent implementations in other languages are encouraged.

Table of Contents

---

1. Introduction

1.1 Background

Large language models (LLMs) have moved rapidly from experimental curiosity to production workloads. The operational profile of a production AI system differs substantially from that of a conventional web service:

- A single user request may spawn **dozens of sequential model calls**, each consuming tokens, incurring cost, and potentially interacting with external tools.
- Model outputs trigger **guard and safety checks** that may block, modify, or re-route the request.
- Intermediate results are stored in **semantic caches** whose hit rate directly affects both latency and cost.
- **Evaluation pipelines** run continuously on production traffic to detect quality regression.
- **Personally Identifiable Information** (PII) routinely appears in user inputs and must be redacted before any data leaves the trust boundary.
- Regulatory frameworks (GDPR, HIPAA, SOC 2, EU AI Act) require **tamper-evident audit trails** for AI-generated decisions.

These requirements cannot be satisfied by general-purpose APM tools alone. They require a shared event language that is intentionally designed for the semantics of AI applications.

1.2 Scope

This RFC specifies the **event schema standard** â€" the data model, serialisation format, identifier scheme, security primitives, privacy primitives, and export abstractions. It does not mandate a specific transport protocol beyond requiring compatibility with OTLP/HTTP. It does not mandate a specific storage backend. It does not mandate a specific programming language for implementations, though a Python reference implementation (`agentobs`) is provided.

1.3 Out of Scope

The following are explicitly out of scope for this version:

- Model evaluation methodology (which metrics to use, not how to record them)
- Prompt content storage or management
- Fine-tuning pipelines
- Model registry or model card formats
- Inference serving infrastructure metrics

---

2. Conventions and Terminology

The key words **MUST**, **MUST NOT**, **REQUIRED**, **SHALL**, **SHALL NOT**, **SHOULD**, **SHOULD NOT**, **RECOMMENDED**, **NOT RECOMMENDED**, **MAY**, and **OPTIONAL** in this document are to be interpreted as described in BCP 14 [RFC 2119] [RFC 8174] when, and only when, they appear in all capitals, as shown here.

2.1 Definitions

Agent â€" an AI system that uses an LLM to select and invoke tools, plan multi-step sequences, and produce a final result through an iterative reasoning loop.

Agent Run â€" a single end-to-end invocation of an agent, consisting of one or more ordered steps, each potentially involving model calls, tool calls, and reasoning activity.

Canonical JSON â€" a deterministic JSON serialisation where object keys are sorted alphabetically at every nesting level, compact separators (`","` and `":"`) are used, and `null` values are omitted. This definition is normative for all signing and hashing operations in this specification.

Decision Point â€" a branching moment within an agent step where the agent explicitly chooses among two or more options. Defined in Â§8.3.

Event â€" the atomic unit of observability defined by this specification. An Event consists of an Envelope (fixed fields) and a Payload (namespace-specific data). Defined in Â§5.

EventType â€" a namespaced string of the form `llm.<namespace>.<entity>.<action>` that identifies the semantic meaning of an Event. Defined in Â§7.

Exporter â€" any component that provides a batch-export operation over a sequence of Events. This is a structural protocol â€" no inheritance or base-class registration is required. Defined in Â§13.

Namespace â€" one of the defined domains registered in Â§7.2. Each namespace owns a set of EventTypes and a set of typed Payload schemas. Defined in Â§7.

Org Secret â€" a secret string held by the operator and used as the HMAC key for signing and verifying an audit chain. MUST NOT appear in any log, exception, or serialised representation.

Provider â€" an entity that hosts and serves an LLM API. Represented by the `GenAISystem` enumeration defined in Â§10.1.

Reasoning Step â€" a discrete unit of chain-of-thought computation produced by a model that exposes its internal reasoning (e.g., OpenAI o1/o3, Anthropic Claude extended thinking, Google Gemini thinking mode). Defined in Â§8.2.

Redactable â€" a string value annotated with a sensitivity level at the point of construction, indicating that a RedactionPolicy may replace it before the Event is written to any log. Defined in Â§12.

Sensitivity Level â€" a five-level ordinal scale (`LOW` , `MEDIUM` , `HIGH` , `PII` , `PHI`) used to classify the privacy risk of a field value. Defined in Â§12.1.

Span â€" an OpenTelemetry-compatible unit of work with a 128-bit trace ID, a 64-bit span ID, an optional parent span ID, a start timestamp, and an end timestamp. Defined in Â§8.1.

ULID â€" Universally Unique Lexicographically Sortable Identifier. A 128-bit, time-ordered, zero-coordination identifier. Defined in Â§6.

---

3. Problem Statement and Motivation

3.1 The Fragmentation Problem

There is currently no broadly adopted standard format for AI observability events across providers and tooling ecosystems. Many tools, frameworks, vendors, and enterprise teams define their own log format. The consequences compound:

Domain	Consequence of No Shared Schema
Observability	Dashboards must be rebuilt per service; no cross-service event correlation
Compliance / Audit	No tamper-evident trail; impossible to reconstruct the exact sequence of AI decisions
Data Privacy	PII surfaces in logs because no one owns the redaction contract at schema level
Cost Attribution	Token costs are `Optional[float]` â€" not auditable, not reproducible 6 months later
Agent Traceability	Multi-step agent runs are not representable as a coherent span tree in any existing schema
Tooling	Every new observability tool must write a custom parser for every upstream log format
CI/CD	Schema drift reaches production silently because there is no compatibility gate

3.2 Why Existing Standards Are Insufficient

OpenTelemetry (OTel) Semantic Conventions for GenAI (`gen_ai.*` , current at time of drafting) define span attributes for individual LLM calls. They are necessary but not sufficient:

- They address the individual span, not the agent run as a whole.
- They do not define how cost is structured and attributed.
- They do not define how PII is redacted before span export.
- They do not define how eval scores attach to spans.
- They do not define an audit-chain integrity mechanism.
- They do not define a deprecation or migration protocol for schema evolution.

This RFC extends OTelâ€™s semantic conventions rather than replacing them. All AGENTOBS field names in the trace namespace are direct OTel `gen_ai.*` attribute names wherever a corresponding attribute exists (Â§14). Where OTel has no corresponding concept â€" agent steps, reasoning steps, decision points, cost breakdowns, redaction records â€" AGENTOBS introduces new, OTel-compatible fields.

In short: OTel gives you individual LLM spans. AGENTOBS gives you the agent run that ties those spans together, the cost model that makes them auditable, the privacy layer that makes them safe to export, and the audit chain that makes them tamper-evident. If you only need individual spans, use OTel directly. If you need anything above, you need AGENTOBS.

3.3 Design Goals

An AGENTOBS implementation MUST:

1. Be expressible as standard JSON without loss of information.
2. Require no mandatory runtime dependencies beyond what is needed to parse and produce JSON.
3. Be fully compatible with the W3C Trace Context specification [W3C-TC].
4. Produce span representations that OTel-aware backends (such as Jaeger, Grafana Tempo, Datadog APM, Honeycomb) can generally render without AGENTOBS-specific knowledge.
5. Be implementable in any programming language.

---

4. Design Principles

The following principles are normative for all AGENTOBS implementations. They are stated as principles rather than rules because they govern the *intent* behind specific rules elsewhere in this document.

P1 â€" Zero Mandatory Dependencies

The core event creation, validation, signing, and redaction operations MUST be implementable using only the standard library of the target programming language. Optional integrations (Pydantic models, OTel SDK wiring, Kafka ingestion) MAY require additional packages but MUST be isolated behind optional install targets.

Rationale: A foundational schema library that imposes dependencies becomes a dependency conflict waiting to happen. Every downstream tool inherits those conflicts.

P2 â€" Immutability After Construction

Once an Event has been constructed and validated, its Envelope fields MUST be read-only. Any attempt to mutate an Envelope field MUST raise an error immediately.

Rationale: Events are passed across module, process, and service boundaries. If any consumer could mutate an event after creation, signed audit chains would become unreliable, and the tamper-evidence guarantee of Â§11 would be invalidated.

P3 â€" Deterministic Serialisation

`Event.to_json()` MUST always produce byte-for-byte identical output for the same Event, regardless of programming language version, operating system, or call time. Keys MUST be sorted alphabetically at every nesting level. `null` values MUST be omitted. Datetimes MUST be formatted as `YYYY-MM-DDThh:mm:ss.ffffffZ` (ISO 8601 with lower-case time designators indicating hours, minutes, seconds).

Rationale: The HMAC signature defined in Â§11 is computed over the Canonical JSON of the payload. Non-deterministic serialisation would make signing and verification produce different results on different machines, rendering the audit chain useless.

P4 â€" Typed Exceptions

Every validation failure MUST raise a typed exception carrying three structured fields: the field name, the received value, and a human-readable reason. Bare `ValueError("bad input")` exceptions and bare `assert` statements MUST NOT be used in conformant implementations.

Rationale: Structured exceptions are machine-parseable. Downstream tooling can inspect the exceptionâ€™s fields to produce actionable error messages without parsing a string.

P5 â€" Namespace-Based Event Types

Standard AGENTOBS EventType strings MUST follow the pattern `llm.<namespace>.<entity>.<action>`. Extension EventTypes MAY use reverse-domain namespaces (for example `com.example.<entity>.<action>`) and MUST NOT use unregistered `llm.*` namespaces.

Rationale: Namespace-based strings enable prefix routing (`e.event_type.startswith("llm.guard.")`), prevent naming collisions between independent tools, and mirror OpenTelemetryâ€™s semantic convention naming scheme.

P6 â€" Vendor Neutrality

No specific LLM provider MUST be required or privileged. The schema MUST support all providers via the `GenAISystem` enumeration (Â§10.1), and MUST provide a `_custom` escape hatch for private or enterprise deployments.

Any `_custom` deployment MUST be annotated with a `custom_system_name` field; `_custom` MUST NOT be used as a silent catch-all.

Rationale: Collapsing all unknown providers into `_custom` without naming makes dashboards unqueryable and cost attribution impossible.

P7 â€" Privacy at Source

PII redaction MUST be expressible at the field level, at the point of field construction, not as a post-processing step. The schema MUST provide a `Redactable` type that carries a sensitivity annotation with every string value that may contain PII. RedactionPolicies MUST act on these annotations before event data is written to any log or exported to any backend.

Rationale: Post-hoc redaction loses the context that identifies which fields are sensitive. Redacting at the data source is the only robust strategy.

---

5. The AGENTOBS Event Envelope

The **Event Envelope** is the outermost, fixed-structure container applied uniformly to every AGENTOBS-compliant observability event. The Envelope is logically separate from the **Payload**, which carries namespace-specific data.

5.1 Envelope Field Specification

Field	Type	Required	Constraints
`schema_version`	`string`	REQUIRED	MUST be `"2.0"` for this revision; implementations MUST also accept `"1.0"` for backward compatibility
`event_id`	`string`	AUTO	ULID (Â§6); MUST be auto-generated if not provided by caller
`event_type`	`string`	REQUIRED	MUST match a registered AGENTOBS type (`llm.<namespace>.<entity>.<action>`) or a registered extension namespace (reverse-domain form)
`timestamp`	`string`	AUTO	ISO 8601 UTC with microsecond precision; MUST be set to `now(UTC)` if not provided
`source`	`string`	REQUIRED	Pattern `<name>@<semver>` â€” e.g.Â `"my-app@1.0.0"`
`payload`	`object`	REQUIRED	Non-empty JSON object; namespace-specific
`trace_id`	`string`	OPTIONAL	32 lowercase hexadecimal characters â€” W3C TraceContext compatible
`span_id`	`string`	OPTIONAL	16 lowercase hexadecimal characters â€” W3C TraceContext compatible
`parent_span_id`	`string`	OPTIONAL	16 lowercase hexadecimal characters
`org_id`	`string`	OPTIONAL	Organisation identifier for multi-tenant isolation
`team_id`	`string`	OPTIONAL	Team within the organisation
`actor_id`	`string`	OPTIONAL	User identifier or service account that produced the event
`session_id`	`string`	OPTIONAL	Conversation or request session grouping
`tags`	`object`	OPTIONAL	Flat map of non-empty string keys to non-empty string values
`checksum`	`string`	OPTIONAL	MUST match `^sha256:[0-9a-f]{64}$` when present
`signature`	`string`	OPTIONAL	MUST match `^hmac-sha256:[0-9a-f]{64}$` when present

Field	Type	Required	Constraints
`prev_id`	`string`	OPTIONAL	ULID of the immediately preceding event in an AuditStream

5.2 Validation Rules

An Event MUST be rejected by a conformant validator if any REQUIRED field is absent, or if any field value violates the constraints in Â§5.1.

An implementation SHOULD provide a `.validate()` method that raises a typed `SchemaValidationError` (Â§4, P4) rather than returning a boolean, so that upstream code can act on specific field failures.

5.3 Serialisation

An implementation MUST provide `to_json()` â†' `str`, `to_dict()` â†' `dict`, `from_json(s: str)` â†' `Event`, and `from_dict(d: dict)` â†' `Event` operations satisfying the determinism requirement in P3.

5.4 Example Envelope (Canonical JSON)

```
{
  "actor_id": "user:alice",
  "event_id": "01HW4Z3RXVP8Q2M6T9KBJDS7YN",
  "event_type": "llm.trace.span.completed",
  "org_id": "org_acme",
  "payload": {
    "duration_ms": 340.5,
    "finish_reason": "stop",
    "input_tokens": 512,
    "model": {"name": "gpt-4o", "system": "openai"},
    "output_tokens": 128
  },
  "schema_version": "2.0",
  "source": "my-app@1.0.0",
  "tags": {"env": "production"},
  "timestamp": "2026-03-04T14:32:11.042817Z",
  "trace_id": "4bf92f3577b34da6a3ce929d0e0e4736"
}
```

---

6. Event Identifier: ULID

Every Event MUST carry a ULID as its `event_id`. A ULID is a 128-bit identifier composed of:

- **48 bits** â€" millisecond-precision Unix timestamp (most significant bits, allowing time-based sorting)
- **80 bits** â€" cryptographically random bits

ULIDs are encoded as 26-character Crockford Base32 strings (e.g., `01HW4Z3RXVP8Q2M6T9KBJDS7YN`).

6.1 Properties

Compared to UUID v4:

Property	UUID v4	ULID
Time-sortable	No	Yes â€” 48-bit timestamp prefix
Lexicographic order matches chronological order	No	Yes
Monotonic within same millisecond	No	Yes (counter increment on collision)
Coordination required	No	No
Database index locality	Poor	Excellent

6.2 Generation Requirements

An AGENTOBS implementation MUST:

1. Generate ULIDs using the standard Crockford Base32 alphabet.
2. Guarantee monotonic ordering within the same millisecond by incrementing the random portion on collision.
3. NOT require an external ULID package â€” the generation algorithm MUST be implementable in any languageâ€™s standard library.

6.3 ULID Validity Check

A string is a valid ULID if and only if: - It is exactly 26 characters. - Every character belongs to the set `[0123456789ABCDEFGHJKMNPQRSTVWXYZ]` (Crockford Base32). - The first character MUST be in the range `0` â€“ `7` (the two most-significant bits of the 130-bit Crockford Base32 space are always zero because a ULID is 128 bits; the maximum valid ULID is `7ZZZZZZZZZZZZZZZZZZZZZZZZZ`, representing dates through approximately `10889-08-02`).

---

7. Event Namespace Taxonomy

7.1 Namespace Structure

All standard AGENTOBS EventTypes follow the BNF:

```
event-type    = "llm." namespace "." entity "." action
namespace     = "trace" / "cost" / "cache" / "eval" / "guard" /
                "fence" / "prompt" / "redact" / "diff" / "template" /
                "audit"
entity        = 1*( ALPHA / "_" )
action        = "started" / "completed" / "failed" / "recorded" /
                "detected" / "blocked" / "hit" / "missed" / "step" /
                1*( ALPHA / "_" )
```

Extension EventTypes use reverse-domain namespace prefixes and follow:

```
extension-event-type = reverse-domain "." entity "." action
reverse-domain       = label "." label *( "." label )
label                = ALPHA *( ALPHA / DIGIT / "-" )
entity               = 1*( ALPHA / DIGIT / "_" )
action               = 1*( ALPHA / DIGIT / "_" )
```

7.2 Built-in Namespace Registry

Namespace	Payload Type	Observability Domain	Stability
`llm.trace.*`	`SpanPayload` / `AgentStepPayload` / `AgentRunPayload`	Model calls, agent runs, reasoning	Frozen v1 / v2 Active
`llm.cost.*`	`CostRecordedPayload`	Per-call cost attribution and rollups	Stable
`llm.cache.*`	`CachePayload`	Semantic cache hit/miss, TTL, key hash	Stable
`llm.eval.*`	`EvalScorePayload` / `EvalRegressionPayload`	Quality scores, regression detection	Stable
`llm.guard.*`	`GuardPayload`	Safety classifier output, block decisions	Stable
`llm.fence.*`	`FencePayload`	Structured output constraints, retry loops	Stable
`llm.prompt.*`	`PromptPayload`	Prompt template versions, rendered text	Stable
`llm.redact.*`	`RedactPayload`	PII audit record â€" categories found, removed	Stable
`llm.diff.*`	`DiffPayload`	Prompt/response delta between two events	Stable
`llm.template.*`	`TemplatePayload`	Template registry metadata, variable bindings	Stable
`llm.audit.*`	`AuditPayload`	Security: key rotation, audit-chain control events	Stable

7.3 Stability Classifications

Frozen â€" No field will be removed or renamed in any minor or patch release. Breaking changes require a major version bump with a two-release deprecation window.

Stable â€" Breaking changes are prohibited within a minor version. Additive changes (new optional fields) are permitted in minor releases.

Experimental â€" Subject to breaking changes in minor releases. Experimental namespaces MUST carry a warning in all documentation.

7.4 Reserved Namespaces

The following prefixes are reserved for future standardisation: `llm.rag.*`, `llm.memory.*`, `llm.planning.*`, `llm.multimodal.*`, `llm.finetune.*`. Third-party namespaces MUST use a reverse-domain prefix outside the `llm.*` tree (e.g., `com.example.custom.event.action`).

7.5 Complete Trace Namespace EventTypes

```
llm.trace.span.started
llm.trace.span.completed
llm.trace.span.failed
llm.trace.agent.step          (v2.0+)
llm.trace.agent.completed     (v2.0+)
llm.trace.reasoning.step      (v2.0+)
```

See Appendix B for the complete EventType registry across all namespaces.

---

8. Agent Span Hierarchy

This section constitutes the core extension that AGENTOBS makes beyond OTel GenAI semconv. It is the primary motivation for a new standard.

8.1 SpanPayload (v2.0)

A `SpanPayload` represents a single unit of LLM work â€" a model call, an embedding request, a tool execution, or a complete agent invocation.

REQUIRED fields:

Field	Type	OTel Attribute	Constraints
`span_id`	`string`	â€"	Exactly 16 lowercase hex characters
`trace_id`	`string`	â€"	Exactly 32 lowercase hex characters
`span_name`	`string`	`gen_ai.operation.name` + model	Non-empty
`operation`	`GenAIOperationName`	`gen_ai.operation.name`	Defined in Â§10.2
`span_kind`	`SpanKind`	`SpanKind`	Defined in Â§10.3
`status`	`string`	â€"	One of `"ok"`, `"error"`, `"timeout"`
`start_time_unix_nano`	`integer`	`startTimeUnixNano`	Non-negative
`end_time_unix_nano`	`integer`	`endTimeUnixNano`	MUST be â‰¥ `start_time_unix_nano`
`duration_ms`	`float`	â€"	MUST equal `(end â^' start) / 1_000_000` Â± 1 ms

OPTIONAL fields: `parent_span_id`, `agent_run_id`, `model` (`ModelInfo`), `token_usage` (`TokenUsage`), `cost` (`CostBreakdown`), `tool_calls` (list, MUST default to `[]` not `null`), `finish_reason` (string â€" e.g., `"stop"`, `"length"`, `"tool_calls"`; mapped to `gen_ai.response.finish_reasons` at export per Â§14.1), `error`, `error_type`, `attributes`.

Envelope alignment: When an Event carries a `SpanPayload`, the envelope `trace_id`, `span_id`, and `parent_span_id` fields â€" if present â€" MUST be identical to the corresponding fields in the payload. The payload fields are authoritative for span hierarchy operations; the envelope copies are OPTIONAL and exist for routing and filtering by consumers that do not inspect the payload.

8.2 ReasoningStep

A `ReasoningStep` represents one discrete chain-of-thought unit emitted by a model that exposes its reasoning (OpenAI o1/o3, Anthropic Claude extended thinking, Google Gemini thinking mode).

Field	Type	Required	Notes
`step_index`	`integer`	REQUIRED	Zero-based index within the parent span
`reasoning_tokens`	`integer`	REQUIRED	Tokens consumed by this step
`duration_ms`	`float`	OPTIONAL	Wall-clock time
`content_hash`	`string`	OPTIONAL	SHA-256 of reasoning content, encoded as 64 lowercase hex characters (no prefix)

Critical Design Decision â€" Reasoning Content Is Never Stored: Raw reasoning content MUST NOT appear in an AGENTOBS event payload. Only the SHA-256 hash of the content MAY be stored. This decision reflects three independent requirements:

1. **IP Protection** â€" internal chain-of-thought representations are a proprietary artifact of the model provider and MAY constitute confidential information.
2. **Privacy** â€" reasoning steps often contain re-statements of sensitive user context and SHOULD be subject to the same PII treatment as input prompts.
3. **Audit Sufficiency** â€" the hash is sufficient to verify that the reasoning content has not been tampered with; the full content is not needed in the audit trail.

8.3 DecisionPoint

A `DecisionPoint` records an explicit branching decision made by an agent during a step.

Field	Type	Required	Valid Values
`decision_id`	`string`	REQUIRED	Unique within the agent run
`decision_type`	`string`	REQUIRED	`"tool_selection"`, `"route_choice"`, `"loop_termination"`, `"escalation"`
`options_considered`	`List[string]`	REQUIRED	Options the agent evaluated
`chosen_option`	`string`	REQUIRED	The selected option
`rationale`	`string`	OPTIONAL	Absent for black-box models that do not expose reasoning

8.4 AgentStepPayload

Represents one iteration of a multi-step agent loop. MUST link into the same span tree via `trace_id`, `span_id`, and `parent_span_id`.

Field	Type	Required
`agent_run_id`	`string`	REQUIRED
`step_index`	`integer`	REQUIRED â€” zero-based
`span_id`	`string`	REQUIRED â€” 16 hex chars
`trace_id`	`string`	REQUIRED â€” 32 hex chars
`parent_span_id`	`string`	OPTIONAL
`operation`	`GenAIOperationName`	REQUIRED
`model`	`ModelInfo`	OPTIONAL
`token_usage`	`TokenUsage`	OPTIONAL
`cost`	`CostBreakdown`	OPTIONAL
`tool_calls`	`List[ToolCall]`	REQUIRED â€” empty list if none
`reasoning_steps`	`List[ReasoningStep]`	REQUIRED â€” empty list if none
`decision_points`	`List[DecisionPoint]`	REQUIRED â€” empty list if none
`status`	`string`	REQUIRED
`start_time_unix_nano`	`integer`	REQUIRED
`end_time_unix_nano`	`integer`	REQUIRED
`duration_ms`	`float`	REQUIRED

8.5 AgentRunPayload

Root-level summary event for a complete agent run. Emitted as `llm.trace.agent.completed` when the final step resolves.

Field	Type	Required
`agent_run_id`	`string`	REQUIRED â€” stable across all steps of this run
`agent_name`	`string`	REQUIRED â€” human-readable agent identifier
`trace_id`	`string`	REQUIRED
`root_span_id`	`string`	REQUIRED â€” the root span containing all children
`total_steps`	`integer`	REQUIRED
`total_model_calls`	`integer`	REQUIRED
`total_tool_calls`	`integer`	REQUIRED
`total_token_usage`	`TokenUsage`	REQUIRED â€” aggregated
`total_cost`	`CostBreakdown`	REQUIRED â€” aggregated
`status`	`string`	REQUIRED â€” `"ok"`, `"error"`, `"timeout"`, `"max_steps_exceeded"`
`start_time_unix_nano`	`integer`	REQUIRED
`end_time_unix_nano`	`integer`	REQUIRED
`duration_ms`	`float`	REQUIRED
`termination_reason`	`string`	OPTIONAL â€” human-readable

8.6 Span Tree Rendering

Because every span in an AGENTOBS agent run carries `trace_id`, `span_id`, and `parent_span_id` with W3C TraceContext-compatible representations, OTel-aware backends can generally render the complete agent span tree without any AGENTOBS-specific knowledge. Each agent step is a child span of the root `AgentRunPayload` span, and model calls within a step are children of the step span. This recursive parent-child structure is the same structure used by most OTel instrumentation libraries â€” AGENTOBS agents are simply first-class OTel spans.

Why parent/child links, not embedded lists:

An alternative design would embed all child spans as a nested list on the parent event. This was rejected because:

1. Backends such as Jaeger, Grafana Tempo, Datadog APM, and Honeycomb commonly rely on parent/child relationships encoded as `parent_span_id` on child spans. Embedding would require every backend to implement a custom tree reconstruction step.
2. Emitting child spans independently allows streaming backends to begin processing before the root span completes.
3. It aligns with the core OTel trace model.

---

9. Token and Cost Model

9.1 TokenUsage

`TokenUsage` covers common token categories exposed by current major LLM providers. All fields beyond `input_tokens`, `output_tokens`, and `total_tokens` are OPTIONAL to accommodate providers that do not expose

granular breakdowns.

Field	Type	OTel Attribute	Provider Mapping
`input_tokens`	`integer`	`gen_ai.usage.input_tokens`	OAI: `prompt_tokens`, Anthropic: `input_tokens`
`output_tokens`	`integer`	`gen_ai.usage.output_tokens`	OAI: `completion_tokens`, Anthropic: `output_tokens`
`total_tokens`	`integer`	â€"	Sum as reported by provider
`cached_tokens`	`integer`	â€"	OAI: `prompt_tokens_details.cached_tokens`, Anthropic: `cache_read_input_tokens`
`cache_creation_tokens`	`integer`	â€"	Anthropic: `cache_creation_input_tokens`
`reasoning_tokens`	`integer`	â€"	OAI: `completion_tokens_details.reasoning_tokens`
`image_tokens`	`integer`	â€"	Providers that expose multimodal token counts

Naming rationale: `input_tokens` / `output_tokens` (not `prompt_tokens` / `completion_tokens`) is required because:

1. OTel semconv 1.27 defines `gen_ai.usage.input_tokens` and `gen_ai.usage.output_tokens`.
2. Anthropicâ€™s API uses `input_tokens` / `output_tokens` natively.
3. In agentic contexts with multi-turn memory, â€œpromptâ€ is semantically ambiguous â€" what the user typed is not the same as the full context window sent to the model.

9.2 ModelInfo

Field	Type	OTel Attribute	Notes
`system`	`GenAISystem`	`gen_ai.system`	Enum; replaces `provider: str`
`name`	`string`	`gen_ai.request.model`	Model as requested by caller
`response_model`	`string`	`gen_ai.response.model`	OPTIONAL; differs in gateway/fallback
`version`	`string`	â€"	OPTIONAL
`custom_system_name`	`string`	â€"	REQUIRED when `system == _custom`

9.3 CostBreakdown

A `float` is not a cost record. A cost record without per-category breakdown is not auditable. `CostBreakdown` is a typed value object that makes cost attribution transparent and reproducible.

Field	Type	Notes
`input_cost_usd`	`float`	REQUIRED â€" cost for input/prompt tokens
`output_cost_usd`	`float`	REQUIRED â€" cost for output/completion tokens
`cached_discount_usd`	`float`	OPTIONAL â€" cache savings (stored as positive; subtracted from total); default `0.0`
`reasoning_cost_usd`	`float`	OPTIONAL â€" cost for reasoning tokens; default `0.0`
`total_cost_usd`	`float`	REQUIRED â€" MUST equal `input + output + reasoning â^' cached_discount` Â± 1e-6 (absolute tolerance)
`currency`	`string`	OPTIONAL â€" ISO 4217; default `"USD"`
`pricing_date`	`string`	OPTIONAL â€" ISO 8601 date of the pricing snapshot

Zero-fill allowance: Implementations that do not yet have access to pricing data MAY populate all cost fields with `0.0`, provided the `CostBreakdown` object is structurally valid (all REQUIRED fields present, `total_cost_usd` equals the sum formula). This allows teams to adopt the schema immediately and backfill pricing logic later.

Why `CostBreakdown` is not a bare float:

1. A single float cannot satisfy a compliance audit question like â€œhow much did response generation cost versus cached input tokens?â€
2. Nested agent chains require cost attribution at each step and rollup at the run level; a scalar float cannot represent this hierarchy.
3. Provider prices change. A cost record recorded today that does not capture the pricing in effect at the time of the call cannot be reproduced or audited six months later.

9.4 PricingTier

Stores the exact pricing rates used to compute a `CostBreakdown`, so cost calculations remain reproducible indefinitely.

Field	Type	Notes
`system`	`GenAISystem`	Provider
`model`	`string`	Model name
`input_per_million_usd`	`float`	USD per 1M input tokens
`output_per_million_usd`	`float`	USD per 1M output tokens
`cached_input_per_million_usd`	`float`	OPTIONAL
`reasoning_per_million_usd`	`float`	OPTIONAL
`effective_date`	`string`	ISO 8601 date these rates were valid

---

10. Provider Normalisation Protocol

Different LLM providers return response objects with different field names, token category names, and cost representations. The provider normalisation layer maps raw provider responses to the canonical schema in one call, ensuring that instrumentation code is provider-agnostic.

10.1 GenAISystem Enumeration

All provider identifiers in AGENTOBS MUST use the `GenAISystem` enumeration, which maps directly to OTel semconv `gen_ai.system` values.

Value	Provider	Notes
`openai`	OpenAI	
`anthropic`	Anthropic	
`cohere`	Cohere	
`vertex_ai`	Google Cloud Vertex AI	
`aws_bedrock`	Amazon Bedrock	
`az.ai.inference`	Azure AI Studio / GitHub Models	
`groq`	Groq	
`ollama`	Ollama	
`mistral_ai`	Mistral AI	
`together_ai`	Together AI	
`hugging_face`	Hugging Face	
`_custom`	Private or enterprise deployment	Requires `custom_system_name`

Why an enum, not a free string:

1. Typo safety â€” `"OpanAI"` passing silently makes dashboards unqueryable.
2. Exhaustive pattern matching â€” tooling can verify all providers are handled.
3. Direct OTel alignment â€” no translation layer at export time.
4. The `_custom` escape hatch handles everything else without losing structural validation.

10.2 GenAIOperationName Enumeration

Value	Meaning
`chat`	Conversational completion
`text_completion`	Single-turn text completion
`embeddings`	Embedding generation
`image_generation`	Image synthesis
`execute_tool`	Tool / function call execution
`invoke_agent`	Sub-agent delegation
`create_agent`	Agent instantiation
`reasoning`	Isolated reasoning / thinking step

10.3 SpanKind Enumeration

Maps to OTel `SpanKind` values relevant to LLM operations.

Value	Meaning
`CLIENT`	Outbound LLM API call â€" most common
`SERVER`	Incoming agent request
`INTERNAL`	Internal reasoning or routing step
`CONSUMER`	Tool execution triggered by LLM output
`PRODUCER`	Event emitted by an agent for downstream consumption

10.4 ProviderNormalizer Protocol

```
ProviderNormalizer:
  system: GenAISystem
  normalize_model(response: dict) -> ModelInfo
  normalize_tokens(response: dict) -> TokenUsage
  normalize_cost(
    token_usage: TokenUsage,
    model: str,
    pricing: PricingTier?
  ) -> CostBreakdown?
```

This is a **structural protocol** â€" any object satisfying the above interface is a valid normalizer, without inheriting from a base class.

A conformant implementation SHOULD provide built-in normalizers for commonly used providers (for example OpenAI, Anthropic, and Ollama). A `GenericNormalizer` that attempts best-effort extraction from unknown response shapes MUST be provided as a fallback.

10.5 normalize_response() Helper

```
normalize_response(
  response: dict,
  system: GenAISystem,
  pricing: PricingTier?,
  custom_normalizer: ProviderNormalizer?
) -> (ModelInfo, TokenUsage, CostBreakdown?)
```

Single call that returns everything needed to populate a `SpanPayload` from any raw provider response dictionary.

---

11. Security: HMAC Audit Chains

11.1 Threat Model

The AGENTOBS audit chain is designed to detect:

1. **Modification** â€" signed event material (`payload` , `event_id` , `prev_id` linkage) was changed after signing.
2. **Deletion** â€" any event was removed from the stream.
3. **Insertion** â€" a fabricated event was inserted into the stream.
4. **Reordering** â€" events were rearranged within the stream.

It is NOT designed to provide:

- Entity authentication (who produced an event) â€" use mTLS or JWT for service identity.
- Confidentiality â€" events are signed, not encrypted.
- Non-repudiation between mutually distrusting parties â€" the shared `org_secret` means any party in possession of the secret could theoretically produce valid signatures.

11.2 Signing Algorithm

Single-event signing is a two-step computation using only standard SHA-256 and HMAC-SHA256 primitives (available in any modern programming languageâ€™s standard library):

```
step 1:  checksum  = "sha256:"      + SHA-256( canonical_payload_json ).hexdigest()
step 2:  sig_input = event_id + "|" + checksum + "|" + (prev_id or "")
         signature = "hmac-sha256:" + HMAC-SHA256( sig_input, org_secret ).hexdigest()
```

Where `canonical_payload_json` is defined as Canonical JSON per Â§2.1 applied to the `payload` field only (not the entire envelope, so that envelope fields like `signature` and `prev_id` are not part of the signed material and can be added without invalidating the payload checksum).

Conformant verifiers MUST also validate `event_id` and `prev_id` linkage as part of chain verification. Envelope fields outside signed material SHOULD be treated as unauthenticated metadata unless separately signed by profile extensions.

11.3 Audit Chain Construction

An `AuditStream` is an ordered, append-only sequence of signed events where each eventâ€™s `prev_id` references the `event_id` of the preceding event. The signature of each event is computed as shown in Â§11.2, with `prev_id` embedded in `sig_input` .

```
event[0]: prev_id=null,          sig = HMAC(id[0] | chk[0] | "")
event[1]: prev_id=event[0].id,   sig = HMAC(id[1] | chk[1] | id[0])
event[2]: prev_id=event[1].id,   sig = HMAC(id[2] | chk[2] | id[1])
  ...                             ...
```

Any modification, insertion, or deletion causes every signature after that point to be invalidated. An adversary cannot produce a valid chain without the `org_secret` .

11.4 Chain Verification

A conformant implementation MUST provide a `verify_chain(events, org_secret)` function that returns:

- `valid: bool` â€" whether the entire chain is intact.
- `first_tampered: string?` â€" `event_id` of the first event with an invalid signature.
- `gaps: List[string]` â€" list of `prev_id` values that point to a missing predecessor (evidence of deletion).
- `tampered_count: integer` â€" total count of invalidated signatures.

11.5 Key Rotation

Key rotation MUST be supported without breaking chain continuity. A key rotation event (`llm.audit.key.rotated`) MUST be inserted into the chain before the new key takes effect. Verifiers that receive a `key_map` argument mapping rotation event IDs to new secrets MUST be able to verify chains that span multiple key generations.

11.6 Implementation MUST Requirements

1. The `org_secret` MUST NOT appear in any exception message, log statement, or object debug/string representation output.
2. All signature comparisons MUST use a constant-time comparison function to prevent timing-based side-channel attacks.
3. Empty or whitespace-only `org_secret` values MUST be rejected at signing time with a `SigningError` , not silently accepted.
4. Signing failures MUST raise a typed `SigningError` â€" they MUST NOT silently produce an unsigned event.

---

12. Privacy: PII Redaction Framework

12.1 Sensitivity Levels

AGENTOBS defines five sensitivity levels as an ordered scale:

```
LOW  <  MEDIUM  <  HIGH  <  PII  <  PHI
```

Level	Meaning
`LOW`	Internal-only data; not personally identifying
`MEDIUM`	Business-sensitive; should not leave the internal tier
`HIGH`	Could indirectly identify an individual
`PII`	Personally Identifiable Information (names, email, phone, address)
`PHI`	Protected Health Information (HIPAA-regulated)

12.2 The Redactable Type

A `Redactable` is a string value annotated with a sensitivity level AT THE POINT OF CONSTRUCTION. It carries the sensitivity annotation through all subsequent operations until a `RedactionPolicy` resolves it.

Conformant implementations MUST guarantee that:

1. `Redactable` content NEVER appears in any exception message, log statement, stack trace, or object debug/string representation output.
2. If a `Redactable` value triggers a validation error, the exception MUST report only the field name and sensitivity level â€" not the value.

12.3 RedactionPolicy

A `RedactionPolicy` specifies a `min_sensitivity` threshold and a `redacted_by` label. When applied to a data structure:

- Fields with sensitivity â‰¥ `min_sensitivity` MUST be replaced with `"[REDACTED by <redacted_by>]"`.
- Fields with sensitivity < `min_sensitivity` MUST be passed through unchanged.
- Non-`Redactable` fields MUST be passed through unchanged.

12.4 Redaction MUST Happen Before Export

A conformant implementation MUST apply the `RedactionPolicy` before passing events to any `Exporter`. Events containing raw `Redactable` values with sensitivity â‰¥ `PII` MUST NOT be written to any persistent storage without first passing through a `RedactionPolicy`.

12.5 Post-Redaction Verification

A conformant implementation SHOULD provide `contains_pii(data)` â†' `bool` and `assert_redacted(data, min_sensitivity)` â†' `void | error` functions to allow callers to assert that redaction has occurred before data crosses a trust boundary.

---

13. Observability and Export

13.1 The Exporter Protocol

Any component that provides a batch-export operation over a sequence of events is a conformant Exporter. This is a **structural protocol** â€" no inheritance or class hierarchy is required.

13.2 Built-in Exporters

Conformant implementations MUST define the Exporter protocol and support at least one interoperable transport. The exporters below are RECOMMENDED reference mappings and MAY be provided as optional modules.

Exporter	Transport	Target
`JSONLExporter`	File I/O	Newline-delimited JSON to local file; optional gzip
`WebhookExporter`	HTTP POST	Any HTTP(S) endpoint; headers, retry backoff, timeout
`OTLPExporter`	OTLP/HTTP JSON	Any OTel collector; gzip, configurable batch size
`DatadogExporter`	Datadog Agent trace API + Datadog metrics API	Datadog APM + metrics
`GrafanaLokiExporter`	Loki `/loki/api/v1/push`	Grafana Loki; multi-tenant via `X-Scope-OrgID`
`OTelBridgeExporter`	OTel SDK span lifecycle	Any registered `SpanProcessor`

13.3 EventStream

An `EventStream` is an immutable, ordered sequence of events with a fluent API for filtering and fan-out export.

Minimum required capabilities:

- Filter by predicate.
- Filter by event type.
- Export (drain) to an Exporter.
- Route subsets of events to specific exporters.
- Ingest from persisted event logs.

Implementations SHOULD provide streaming (memory-efficient) ingestion for large log files.

---

14. OpenTelemetry Alignment

14.1 Required OTel Mappings

Every `SpanPayload` exported via `OTLPExporter` or `OTelBridgeExporter` MUST map AGENTOBS fields to the following OTel span fields:

OTel Field	Source in AGENTOBS Schema
`gen_ai.system`	`payload.model.system`
`gen_ai.request.model`	`payload.model.name`
`gen_ai.response.model`	`payload.model.response_model`
`gen_ai.usage.input_tokens`	`payload.token_usage.input_tokens`
`gen_ai.usage.output_tokens`	`payload.token_usage.output_tokens`
`gen_ai.operation.name`	`payload.operation`
`gen_ai.response.finish_reasons`	`payload.finish_reason` (or omitted when unavailable)
`deployment.environment.name`	`tags["env"]`
`span.kind`	`payload.span_kind`
Trace-context flags	sampling decision (`01` sampled, `00` unsampled)
`endTimeUnixNano`	`payload.end_time_unix_nano`
`status.code`	`ERROR` for `"error"/"timeout"` , `OK` for `"ok"`

14.2 W3C TraceContext Compatibility

Conformant implementations MUST provide:

- `make_traceparent(trace_id, span_id, sampled?)` â†' `string` â€" produces a valid W3C TraceContext `traceparent` header in the form `00-<32-hex-trace-id>-<16-hex-span-id>-<flags>` , where `<flags>` is `01` (sampled â€" the default) or `00` (not sampled).
- `extract_trace_context(headers)` â†' `dict` â€" parses incoming headers to resume a distributed trace.

`trace_id` MUST be exactly 32 lowercase hexadecimal characters (W3C 16-byte trace ID). `span_id` MUST be exactly 16 lowercase hexadecimal characters (W3C 8-byte span ID).

---

15. Governance and Schema Evolution

15.1 EventGovernancePolicy

Operators MUST be able to configure blocking and warning rules on event types at runtime:

- `blocked_types` â€" event types that are hard-blocked; emitting them MUST raise a `GovernanceViolationError`.
- `warn_deprecated` â€" event types that are soft-blocked; emitting them MUST emit a `DeprecationWarning` but MUST NOT raise.
- `custom_rules` â€" arbitrary callables returning a pass indicator or an error string (fail); support complex cross-field governance constraints.
- `strict_unknown` â€" if `true`, event types not registered in the known EventType set MUST be hard-blocked.

15.2 Consumer Compatibility Registry

A `ConsumerRegistry` allows tools to declare their schema version dependencies at startup and fail fast before processing any events:

```
ConsumerRegistry.register(
  consumer_id: string,
  requires_namespace: string,
  min_version: semver,
  max_version: semver
)
ConsumerRegistry.assert_compatible() -> void | error
```

15.3 Deprecation Registry

Every deprecated EventType or field MUST have a `DeprecationRecord` containing:

- `deprecated_since` â€" the version in which the deprecation was announced.
- `sunset_version` â€" the version in which the type will be removed.
- `sunset_policy` â€" one of `NEXT_MAJOR`, `NEXT_MINOR`, `LONG_TERM`, `UNSCHEDULED`.
- `replacement` â€" the new EventType or field name.
- `field_renames` â€" a map of old field names to new field names.

Old types MUST remain importable and MUST emit a `DeprecationWarning` via the `deprecations` registry. They MUST NOT be fully removed until the `sunset_version` is released.

15.4 Migration Roadmap

Conformant implementations MUST provide a programmatic interface to the migration roadmap so that operators can audit outstanding migrations without reading release notes:

```
migration_roadmap() -> List[DeprecationRecord]  # sorted by event_type
```

15.5 Schema Version Field

Every Event produced under this revision carries `schema_version: "2.0"`. Events produced under earlier revisions carry `"1.0"`. Implementations MUST accept both `"1.0"` and `"2.0"` events but MUST NOT silently ignore an unrecognised `schema_version` â€" they MUST raise a `SchemaVersionError` and stop processing.

---

16. Compliance Checks

16.1 Built-in Checks

Conformant implementations MUST provide at minimum the following programmatic compliance checks, executable without a test framework:

Check ID	What It Verifies
`CHK-1`	All required Envelope fields are present on every event
`CHK-2`	All `event_type` values are registered EventType strings matching `llm\.<ns>\.<entity>\.<action>` or a registered extension namespace (reverse-domain form per Â§7.1)
`CHK-3`	All `source` identifiers follow the `<name>@<semver>` pattern
`CHK-4`	All `event_id` values are valid ULIDs

16.2 Multi-tenant Isolation Checks

For multi-tenant deployments, implementations SHOULD provide:

- `verify_tenant_isolation(events)` â€" detects cross-tenant data leakage (events with different `org_id` values mixed in the same stream).
- `verify_events_scoped(events, org_id)` â€" asserts all events in a stream belong to a single expected organisation.

16.3 Chain Integrity Check

```
verify_chain_integrity(events, org_secret) -> ComplianceResult
```

MUST check for tampering, missing `prev_id` links (deletions), and timestamp non-monotonicity, returning structured results per Â§11.4.

16.4 CI Pipeline Integration

Conformant implementations MUST provide a CLI entry point that can run compliance checks against a JSON or JSONL log file and exit with a non-zero status code on failure. The output MUST be both human-readable and machine-parseable (via `--json` flag).

```
agentobs check-compat events.json
agentobs check-compat events.json --json
```

---

17. Schema Validation

17.1 JSON Schema

Conformant implementations MUST publish a versioned JSON Schema (Draft 2020-12, or a later backward-compatible revision) at a stable, well-known path (`schemas/v<version>/schema.json`) encoding all Envelope field constraints.

Scope: This requirement applies to **independent implementations** (libraries, SDKs) that define their own event-creation and validation code. Applications that consume events using an existing conformant SDK do not need to republish the schema â€" they inherit conformance from the SDK.

17.2 Validation Backends

Implementations MUST support two validation paths:

1. **Full JSON Schema validation** using the published schema file when a JSON Schema validator library is available.
2. **Structural stdlib fallback** performing field-type checks without any additional dependency.

Both paths MUST raise the same typed `SchemaValidationError` on failure.

---

18. Conformance Profiles

The previous sections define the full AGENTOBS capability set. This section groups those capabilities into **four conformance profiles** so that implementers can adopt incrementally and claim precise conformance without implementing every feature at once. Profiles exist to reduce implementation burden, not to increase it.

An implementation that is a **library or SDK** MUST declare which profile(s) it conforms to in its documentation and in the output of its CLI `--version` command (if a CLI is provided). Applications that consume events using an existing conformant SDK are not required to make formal profile claims (see Â§18.6). Profiles are **cumulative** â€" each higher profile includes all requirements of the profiles below it.

18.1 Core Profile

The Core Profile defines the minimum interoperable AGENTOBS implementation. It standardises AI spans, token usage, cost attribution, and agent run telemetry. An implementation claiming **Core** conformance MUST:

#	Requirement	Spec Reference
C-1	Implement the Event Envelope	Â§5
C-2	Generate ULIDs as `event_id`	Â§6
C-3	Define and enforce the EventType namespace taxonomy	Â§7
C-4	Provide `SpanPayload`	Â§8.1
C-5	Provide `AgentStepPayload` and `AgentRunPayload`	Â§8.4, Â§8.5
C-6	Provide `TokenUsage` with `input_tokens` / `output_tokens` naming	Â§9.1
C-7	Provide `CostBreakdown` as a typed value object	Â§9.3
C-8	Deterministic Canonical JSON serialisation	Â§4 P3
C-9	Provide the Exporter protocol and at least one transport	Â§13.1, Â§13.2
C-10	Produce OTel-compatible span attributes at export	Â§14
C-11	Publish a versioned JSON Schema file	Â§17

An implementation claiming only Core conformance MAY omit HMAC signing, `Redactable` types, `RedactionPolicy`, governance primitives, and CLI compliance checks entirely.

Intended audience: Teams that want standardised AI spans and cost attribution without security, privacy, or governance overhead.

18.2 Security Profile

The Security Profile adds tamper-evident audit chains. An implementation claiming **Security** conformance MUST satisfy all Core requirements AND:

#	Requirement	Spec Reference
S-1	Implement HMAC-SHA256 single-event signing	Â§11.2
S-2	Implement audit chain construction (`prev_id` linkage)	Â§11.3
S-3	Provide `verify_chain()` with structured results	Â§11.4
S-4	Support key rotation without breaking chain continuity	Â§11.5
S-5	Use constant-time comparison for all HMAC verification	Â§11.6 item 2
S-6	Reject empty or whitespace-only `org_secret` at signing time	Â§11.6 item 3
S-7	Prevent `org_secret` leakage (exceptions, logs, debug output)	Â§11.6 item 1, Â§19.3

Intended audience: Teams with compliance or audit requirements (SOC 2, ISO 27001, financial services).

18.3 Privacy Profile

The Privacy Profile adds field-level PII controls. An implementation claiming **Privacy** conformance MUST satisfy all Core requirements AND:

#	Requirement	Spec Reference
P-1	Provide the `Redactable` type with sensitivity annotation at construction	Â§12.2
P-2	Implement the five-level sensitivity scale (`LOW` through `PHI`)	Â§12.1
P-3	Provide `RedactionPolicy` with threshold-based redaction	Â§12.3
P-4	Apply redaction before any export (`Exporter` integration)	Â§12.4
P-5	Prevent `Redactable` content from appearing in exceptions, logs, or debug output	Â§12.2
P-6	Provide post-redaction verification functions	Â§12.5

The Privacy Profile does NOT require the Security Profile. An implementation MAY claim Core + Privacy without implementing HMAC signing.

Intended audience: Teams operating under GDPR, HIPAA, CCPA, or EU AI Act obligations.

18.4 Enterprise Profile

The Enterprise Profile adds schema lifecycle governance. An implementation claiming **Enterprise** conformance MUST satisfy **all** of Core, Security, AND Privacy, AND:

#	Requirement	Spec Reference
E-1	Provide `EventGovernancePolicy` with blocked/deprecated/custom rules	Â§15.1
E-2	Provide `ConsumerRegistry` for version-dependency declarations	Â§15.2
E-3	Provide `DeprecationRegistry` with sunset tracking	Â§15.3
E-4	Provide programmatic `migration_roadmap()` interface	Â§15.4
E-5	Provide CLI compliance checks (`check-compat`) with JSON output	Â§16.4
E-6	Provide built-in compliance checks CHK-1 through CHK-4	Â§16.1
E-7	Provide chain integrity check	Â§16.3

The Enterprise Profile is a strict superset of all other profiles.

Intended audience: Multi-team organisations managing schema evolution across a distributed ecosystem of tools and services.

18.5 Profile Compatibility Matrix

```
â”Œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”¬â”€â”€â”€â”€â”€â”€â”€â”€â”¬â”€â”€â”€â”€â”€â”€â”€â”€â”¬â”€â”€â”€â”€â”€â”€â”€â”€â”¬â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”
â”‚ Capability                 â”‚ Core   â”‚ Security â”‚ Privacy â”‚ Enterprise â”‚
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”¼â”€â”€â”€â”€â”€â”€â”€â”€â”¼â”€â”€â”€â”€â”€â”€â”€â”€â”¼â”€â”€â”€â”€â”€â”€â”€â”€â”¼â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”¤
â”‚ Event Envelope             â”‚  MUST  â”‚  MUST    â”‚  MUST   â”‚   MUST     â”‚
â”‚ ULID                       â”‚  MUST  â”‚  MUST    â”‚  MUST   â”‚   MUST     â”‚
â”‚ Namespace Taxonomy         â”‚  MUST  â”‚  MUST    â”‚  MUST   â”‚   MUST     â”‚
â”‚ SpanPayload                â”‚  MUST  â”‚  MUST    â”‚  MUST   â”‚   MUST     â”‚
â”‚ AgentStep/Run              â”‚  MUST  â”‚  MUST    â”‚  MUST   â”‚   MUST     â”‚
â”‚ TokenUsage                 â”‚  MUST  â”‚  MUST    â”‚  MUST   â”‚   MUST     â”‚
â”‚ CostBreakdown              â”‚  MUST  â”‚  MUST    â”‚  MUST   â”‚   MUST     â”‚
â”‚ Canonical JSON             â”‚  MUST  â”‚  MUST    â”‚  MUST   â”‚   MUST     â”‚
â”‚ Exporter protocol          â”‚  MUST  â”‚  MUST    â”‚  MUST   â”‚   MUST     â”‚
â”‚ OTel alignment             â”‚  MUST  â”‚  MUST    â”‚  MUST   â”‚   MUST     â”‚
â”‚ JSON Schema                â”‚  MUST  â”‚  MUST    â”‚  MUST   â”‚   MUST     â”‚
â”‚ HMAC signing               â”‚   â€"   â”‚  MUST    â”‚   â€"    â”‚   MUST     â”‚
â”‚ Audit chain                â”‚   â€"   â”‚  MUST    â”‚   â€"    â”‚   MUST     â”‚
â”‚ Key rotation               â”‚   â€"   â”‚  MUST    â”‚   â€"    â”‚   MUST     â”‚
â”‚ Redactable type            â”‚   â€"   â”‚   â€"     â”‚  MUST   â”‚   MUST     â”‚
â”‚ RedactionPolicy            â”‚   â€"   â”‚   â€"     â”‚  MUST   â”‚   MUST     â”‚
â”‚ Pre-export redact          â”‚   â€"   â”‚   â€"     â”‚  MUST   â”‚   MUST     â”‚
â”‚ Governance policy          â”‚   â€"   â”‚   â€"     â”‚   â€"    â”‚   MUST     â”‚
â”‚ ConsumerRegistry           â”‚   â€"   â”‚   â€"     â”‚   â€"    â”‚   MUST     â”‚
â”‚ DeprecationRegistry        â”‚   â€"   â”‚   â€"     â”‚   â€"    â”‚   MUST     â”‚
â”‚ CLI compliance             â”‚   â€"   â”‚   â€"     â”‚   â€"    â”‚   MUST     â”‚
â””â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”´â”€â”€â”€â”€â”€â”€â”€â”€â”´â”€â”€â”€â”€â”€â”€â”€â”€â”´â”€â”€â”€â”€â”€â”€â”€â”€â”´â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”˜
```

18.6 Conformance Claim Syntax

Libraries and SDKs MUST state their conformance using one or more of the following labels:

- `AGENTOBS-Core-2.0`
- `AGENTOBS-Security-2.0`
- `AGENTOBS-Privacy-2.0`
- `AGENTOBS-Enterprise-2.0`

Multiple profiles MAY be combined (e.g., `AGENTOBS-Core-2.0, AGENTOBS-Security-2.0` for an implementation that provides audit chains but not PII redaction). The Enterprise label implies all four.

Applications (services, scripts, notebooks) that use an existing conformant SDK are not expected to make formal conformance claims. They inherit the profile conformance of the SDK they depend on.

18.7 SHOULD Requirements (All Profiles)

Regardless of profile, an implementation SHOULD:

1. Run property-based tests against signing, serialisation, and ULID generation.
2. Provide built-in `ProviderNormalizer` implementations for widely used providers.
3. Provide framework integrations as optional modules.

18.8 MAY Provisions (All Profiles)

An implementation MAY:

1. Provide additional named exporters beyond the required minimum.
2. Provide Kafka ingestion support.
3. Provide additional typed model layers (e.g., Pydantic, TypeScript Zod).
4. Define custom namespaces using a reverse-domain prefix outside `llm.*`.

18.9 Adoption Guide

This subsection is informative. It summarises practical adoption paths to help teams get started quickly.

18.9.1 Span-Only Adoption (Minimal Instrumentation)

An implementation that does not perform agent orchestration MAY emit only `llm.trace.span.*` events and omit `AgentStepPayload` and `AgentRunPayload` entirely. This is a valid Core-conformant deployment. The `AgentStepPayload` and `AgentRunPayload` types MUST exist in the SDK surface for interoperability (C-5), even if the application never emits those events.

This allows teams that wrap a single LLM call to adopt AGENTOBS without implementing agent-run bookkeeping.

18.9.2 What You Can Ignore (Core Only)

If you adopt only the **Core Profile**, you MAY ignore these sections entirely:

- **Â§11** â€" Security: HMAC Audit Chains
- **Â§12** â€" Privacy: PII Redaction Framework
- **Â§15** â€" Governance and Schema Evolution
- **Â§16** â€" Compliance Checks
- **Â§19** â€" Security Considerations (applies only to Security Profile)
- **Â§20** â€" Privacy Considerations (applies only to Privacy Profile)

You only need: Envelope (Â§5), ULID (Â§6), Namespaces (Â§7), Span Hierarchy (Â§8), Token & Cost (Â§9), Export (Â§13), OTel Alignment (Â§14), and Schema Validation (Â§17).

18.9.3 Quick Start Path

A small team can reach `AGENTOBS-Core-2.0` conformance by implementing five things:

1. Event Envelope with auto-generated ULID and timestamp
2. `SpanPayload` with `TokenUsage`
3. Deterministic JSON serialisation (sorted keys, no nulls)
4. One Exporter (JSONL to file is the simplest)
5. OTel attribute mapping at export time

A minimal instrumented LLM call looks like this:

```
{
  "event_id": "01HW4Z3RXVP8Q2M6T9KBJDS7YN",
  "event_type": "llm.trace.span.completed",
  "schema_version": "2.0",
  "source": "my-app@1.0.0",
  "timestamp": "2026-03-04T14:32:11.042817Z",
  "payload": {
    "span_id": "a1b2c3d4e5f6a7b8",
    "trace_id": "4bf92f3577b34da6a3ce929d0e0e4736",
    "span_name": "chat gpt-4o",
    "operation": "chat",
    "span_kind": "CLIENT",
    "status": "ok",
    "start_time_unix_nano": 1741099931000000000,
    "end_time_unix_nano": 1741099931340500000,
    "duration_ms": 340.5,
    "model": {"name": "gpt-4o", "system": "openai"},
    "token_usage": {
      "input_tokens": 512,
      "output_tokens": 128,
      "total_tokens": 640
    },
    "cost": {
      "input_cost_usd": 0.0,
      "output_cost_usd": 0.0,
      "total_cost_usd": 0.0
    },
    "finish_reason": "stop"
  }
}
```

No signing. No redaction. No governance. That is a complete, valid, `AGENTOBS-Core-2.0` event.

18.9.4 Migration from Plain Logs

Teams already logging LLM calls as ad-hoc JSON can migrate incrementally. A typical before/after:

Before (ad-hoc):

```
{"model": "gpt-4o", "tokens": 640, "latency_ms": 340}
```

After (AGENTOBS Core):

```
{
  "event_id": "01HW4Z3RXVP8Q2M6T9KBJDS7YN",
  "event_type": "llm.trace.span.completed",
  "schema_version": "2.0",
  "source": "my-app@1.0.0",
  "timestamp": "2026-03-04T14:32:11.042817Z",
  "payload": {
    "span_id": "a1b2c3d4e5f6a7b8",
    "trace_id": "4bf92f3577b34da6a3ce929d0e0e4736",
    "span_name": "chat gpt-4o",
    "operation": "chat",
    "span_kind": "CLIENT",
    "status": "ok",
    "start_time_unix_nano": 1741099931000000000,
    "end_time_unix_nano": 1741099931340500000,
    "duration_ms": 340.5,
    "token_usage": {
      "input_tokens": 640,
      "output_tokens": 0,
      "total_tokens": 640
    }
  }
}
```


The delta is: a standard envelope (5 fields), a structured payload instead of flat keys, and a ULID instead of no identifier. The data you already capture maps directly into the payload.

---

19. Security Considerations

19.1 HMAC Key Management

The `org_secret` used for HMAC signing is a symmetric secret shared among all parties authorised to verify the audit chain. Operators MUST:

1. Treat `org_secret` values with the same rigour as database passwords â€" store them in a secrets manager (e.g., HashiCorp Vault, AWS SSM Parameter Store, Azure Key Vault), not in source code.
2. Rotate secrets on a schedule consistent with organisational security policy, using the key rotation mechanism defined in Â§11.5.
3. Restrict access to the secret to the minimum set of services that must verify the chain.

19.2 Timing Attacks

All HMAC digest comparisons MUST use constant-time comparison functions. Variable-time string comparison (e.g., a naive byte-by-byte equality check that short-circuits on the first mismatch) leaks information about the length of the common prefix of two HMAC values, enabling a timing oracle attack. Implementations MUST use a constant-time digest comparison primitive (for example, `hmac.compare_digest` in Python, `crypto.timingSafeEqual` in Node.js, or the equivalent in other languages).

19.3 Secret Leakage Prevention

The `org_secret` MUST NOT appear in:

- Exception messages (`SchemaValidationError`, `SigningError`, etc.)
- Log statements at any level
- Object debug/string representations
- Stack traces

- Any serialised Event field

Implementations SHOULD use a dedicated secret-holder type for `org_secret` parameters to prevent accidental logging by third-party libraries.

19.4 Denial of Service

Implementations that parse Events from untrusted input (e.g., an HTTP webhook receiver) MUST:

1. Enforce a maximum event size limit (RECOMMENDED: 1 MB).
2. Enforce a maximum `payload` nesting depth (RECOMMENDED: 10 levels).
3. Enforce a maximum number of `tags` keys (RECOMMENDED: 50).

19.5 Reasoning Content

As specified in Â§8.2, raw reasoning content from chain-of-thought models MUST NOT be stored in events. Only the SHA-256 hash MAY be stored. This mitigates the risk of proprietary model artifacts and user PII being written to observability backends that may have broader access controls than primary data stores.

---

20. Privacy Considerations

20.1 GDPR / CCPA Compliance

AI observability pipelines frequently process personal data â€" user messages, names, email addresses, phone numbers in prompts. Operators deploying AGENTOBS-conformant systems in jurisdictions subject to GDPR, CCPA, or equivalent legislation MUST:

1. Apply a `RedactionPolicy` with `min_sensitivity=PII` before events are exported to any backend.
2. Classify all fields that may contain user-supplied text as `Sensitivity.PII` at the point of construction.
3. Document the `redacted_by` label in their data processing record.

20.2 HIPAA Compliance

For healthcare use cases where PHI may appear in prompts, operators MUST:

1. Apply a `RedactionPolicy` with `min_sensitivity=PHI`.
2. Ensure HIPAA-covered data never reaches a log backend that is not itself HIPAA Business Associate Agreement (BAA) covered.

20.3 Right to Erasure

Because AGENTOBS events are intended to be cryptographically chained, deleting a specific event from an audit chain invalidates all subsequent signatures. Operators MUST document this constraint in their privacy notices and provide a mechanism for producing a *successor chain* that excludes the deleted event while preserving the integrity of remaining events.

20.4 Data Minimisation

The `payload` field SHOULD contain only data necessary for the stated observability purpose. Entire model prompt/response bodies SHOULD NOT be stored â€" only hashes, length statistics, and redacted summaries. The `content_hash` fields on `ReasoningStep` exemplify this principle.

---

21. Interoperability Considerations

21.1 Language Neutrality

Although the reference implementation is in Python, this specification is language-neutral. Implementations in Go, Rust, TypeScript, Java, and .NET are encouraged. The only shared dependencies are:

- A JSON serialiser that supports key sorting.
- A SHA-256 / HMAC-SHA256 implementation (available in every languageâ€™s standard library).
- A Crockford Base32 codec.

21.2 Versioned JSON Schema as the Portability Contract

The versioned JSON Schema at `schemas/v1.0/schema.json` is the normative, language-neutral definition of the Event Envelope. Any implementation that validates events against this schema is interoperable with any other conformant implementation.

The schema file path intentionally uses the `v1.0/` directory. The `v1.0/` schema is the stable normative artifact and validates events across all AGENTOBS releases, including this v2.0 data model revision. A separate `v2.0/` schema path will not be created; the `v1.0/` schema will be updated in place under its existing path when breaking schema changes are ratified.

21.3 Backward Compatibility Policy

- **Patch releases** â€" bug fixes only; no API changes.
- **Minor releases** â€" additive changes only; new optional fields, new EventTypes, new exporters.
- **Major releases** â€" breaking changes permissible; MUST be preceded by a deprecation period of at least two minor releases or six calendar months, whichever is longer.

The migration roadmap API (Â§15.4) allows operators to programmatically enumerate outstanding breaking changes and plan migration work before a major release.

---

22. Normative References

- [RFC 2119] Bradner, S., â€œKey words for use in RFCs to Indicate Requirement Levelsâ€, BCP 14, March 1997.
- [RFC 8174] Leiba, B., â€œAmbiguity of Uppercase vs Lowercase in RFC 2119 Key Wordsâ€, BCP 14, May 2017.
- [W3C-TC] W3C, â€œTrace Contextâ€, W3C Recommendation. https://www.w3.org/TR/trace-context/
- [OTel-GenAI] OpenTelemetry Project, â€œSemantic Conventions for Generative AI systemsâ€, semconv 1.27, 2025. https://opentelemetry.io/docs/specs/semconv/gen-ai/
- [ULID] Feerasta, A., â€œUniversally Unique Lexicographically Sortable Identifier (ULID)â€, 2016. https://github.com/ulid/spec
- [SEMVER] Preston-Werner, T., â€œSemantic Versioning 2.0.0â€. https://semver.org/
- [ISO-4217] ISO, â€œCodes for the representation of currenciesâ€, 2015.
- [ISO-8601] ISO, â€œDate and time â€" Representations for information interchangeâ€, 2019.

---

23. Informative References

- [OTel-Core] OpenTelemetry Project, â€œOpenTelemetry Specificationâ€, 2024. https://opentelemetry.io/docs/specs/otel/
- [GDPR] European Parliament, â€œGeneral Data Protection Regulationâ€, Regulation (EU) 2016/679.

- [HIPAA] U.S. Department of Health and Human Services, â€œHealth Insurance Portability and Accountability Actâ€, 1996.
- [EU-AI-Act] European Parliament, â€œArtificial Intelligence Actâ€, Regulation (EU) 2024/1689.
- [agentobs] SpanForge reference implementation, v2.0.0. https://github.com/veerarag1973/Spanforge

24. Public Review Questions

The following questions are explicitly opened for public comment before ratification:

1. **Signature coverage:** Should AGENTOBS define an optional profile that signs selected envelope fields in addition to `payload`?
2. **Reasoning privacy:** Is `content_hash` sufficient for auditability in regulated environments, or should a sealed escrow profile be defined?
3. **Cost model scope:** Should `CostBreakdown` add non-USD currency normalization guidance beyond ISO-4217 labeling?
4. **Namespace governance:** Should third-party extension namespaces be registered centrally, or remain fully decentralized?
5. **OTel evolution:** What compatibility policy should AGENTOBS adopt when OTel GenAI semantic conventions rename or deprecate fields?
6. **Profile boundaries:** Are the four conformance profiles (Core, Security, Privacy, Enterprise) at the right boundaries? Should any capability move between profiles?

Appendix A: Complete Event Envelope Field Reference

```
â”Œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€
â”‚                          Event Envelope                            â”‚
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”¬â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”¬â”€
â”‚ Field           â”‚ Type        â”‚ Required  â”‚ Constraint          â”‚
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”¼â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”¼â”€
â”‚ schema_version  â”‚ string      â”‚ REQUIRED  â”‚ "2.0"               â”‚
â”‚ event_id        â”‚ string      â”‚ AUTO      â”‚ Valid ULID          â”‚
â”‚ event_type      â”‚ string      â”‚ REQUIRED  â”‚ Â§5.1 / Â§7.1         â”‚
â”‚ timestamp       â”‚ string      â”‚ AUTO      â”‚ ISO 8601 UTC        â”‚
â”‚ source          â”‚ string      â”‚ REQUIRED  â”‚ name@semver         â”‚
â”‚ payload         â”‚ object      â”‚ REQUIRED  â”‚ Non-empty           â”‚
â”‚ trace_id        â”‚ string      â”‚ OPTIONAL  â”‚ 32 hex chars        â”‚
â”‚ span_id         â”‚ string      â”‚ OPTIONAL  â”‚ 16 hex chars        â”‚
â”‚ parent_span_id  â”‚ string      â”‚ OPTIONAL  â”‚ 16 hex chars        â”‚
â”‚ org_id          â”‚ string      â”‚ OPTIONAL  â”‚                     â”‚
â”‚ team_id         â”‚ string      â”‚ OPTIONAL  â”‚                     â”‚
â”‚ actor_id        â”‚ string      â”‚ OPTIONAL  â”‚                     â”‚
â”‚ session_id      â”‚ string      â”‚ OPTIONAL  â”‚                     â”‚
â”‚ tags            â”‚ object      â”‚ OPTIONAL  â”‚ stringâ†’string map    â”‚
â”‚ checksum        â”‚ string      â”‚ OPTIONAL  â”‚ sha256:[64 hex]     â”‚
â”‚ signature       â”‚ string      â”‚ OPTIONAL  â”‚ hmac-sha256:[64 hex] â”‚
â”‚ prev_id         â”‚ string      â”‚ OPTIONAL  â”‚ Valid ULID          â”‚
â””â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”´â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”´â”€
```

Appendix B: Canonical Namespace Event Types

```
llm.trace.span.started
llm.trace.span.completed
llm.trace.span.failed
llm.trace.agent.step
llm.trace.agent.completed
llm.trace.reasoning.step

llm.cost.token.recorded
llm.cost.session.recorded
llm.cost.attributed

llm.cache.hit
llm.cache.miss
llm.cache.evicted
llm.cache.written

llm.eval.score.recorded
llm.eval.regression.detected
llm.eval.scenario.started
llm.eval.scenario.completed

llm.guard.input.blocked
llm.guard.input.passed
llm.guard.output.blocked
llm.guard.output.passed

llm.fence.validated
llm.fence.retry.triggered
llm.fence.max_retries.exceeded

llm.prompt.rendered
llm.prompt.template.loaded
llm.prompt.version.changed

llm.redact.pii.detected
llm.redact.phi.detected
llm.redact.applied

llm.diff.computed
llm.diff.regression.flagged

llm.template.registered
llm.template.variable.bound
llm.template.validation.failed

llm.audit.key.rotated
```

Appendix C: GenAI System Registry

This appendix reproduces the normative `GenAISystem` values from Â§10.1 for reference.

AGENTOBS Value	OTel `gen_ai.system` Value	Provider
`openai`	`openai`	OpenAI
`anthropic`	`anthropic`	Anthropic
`cohere`	`cohere`	Cohere
`vertex_ai`	`vertex_ai`	Google Cloud Vertex AI
`aws_bedrock`	`aws_bedrock`	Amazon Bedrock
`az.ai.inference`	`az.ai.inference`	Azure AI Studio / GitHub Models
`groq`	`groq`	Groq
`ollama`	`ollama`	Ollama
`mistral_ai`	`mistral_ai`	Mistral AI
`together_ai`	`together_ai`	Together AI
`hugging_face`	`hugging_face`	Hugging Face
`_custom`	*(not mapped)*	Private / enterprise deployment

Appendix D: Change History

The entries below are project milestone notes in this draft document and are subject to update during ratification.

Version	Date	Changes
v1.0.0	2025-09-01	Initial event envelope, ULID, HMAC signing, 10 namespaces
v1.1.0	2026-01-15	GuardPolicy, FencePolicy, TemplatePolicy runtime enforcement; OTel compliance table
v1.1.2	2026-03-03	W3C Trace Context propagation updates; Datadog and Grafana Loki exporter additions
v2.0.0	2026-Q2	GenAISystem enum, TokenUsage v2 (input/output), CostBreakdown, AgentStepPayload, AgentRunPayload, ReasoningStep, DecisionPoint, ProviderNormalizer â€” **this RFC targets v2.0**

Author Contact


```
Sriram
SpanForge
GitHub: https://github.com/veerarag1973/Spanforge
Issues: https://github.com/veerarag1973/Spanforge/issues
```

Contributions, objections, and implementation reports are welcome via the GitHub issue tracker. This document is a living standard prior to ratification. Public comment period is open from March 4, 2026 through June 4, 2026.

End of RFC-0001-AGENTOBS