4D-ID 1.1

Unified Spatiotemporal Identity for Shared Reality

Author: D. Lorenzini

May 2026

Status: Working draft. Not yet submitted to a standards body.

Executive summary

4D-ID is a unified spatiotemporal identity framework. It assigns every object, surface, agent, and environment — physical or virtual — a stable, hierarchical identity that links global Earth-anchored reference, local high-precision coordinates, orientation, time, and optional motion.

It is designed to sit alongside and extend the OGC GeoPose and DGGS standards, providing the missing layer that connects standards-body geospatial primitives to the runtime needs of AI perception, Gaussian-splat reconstruction, shared XR, and robotics.

4D-ID is not a coordinate system. It is an *identity and addressing layer* on top of existing coordinate systems. Its contribution is not new geometry but a shared way to name, address, and reason about spatial entities across the systems that already exist.

The framework targets four convergent use cases:

- Shared XR that does not drift across users, sessions, or devices
- Digital twins that hold registration with their physical referent over time
- Multi-agent robotics and AI perception working in a shared spatial substrate
- AI-readable spatial memory queryable through vector databases and retrieval-augmented reasoning

---

1. The problem 4D-ID addresses

Modern spatial computing fragments along five lines:

Domain	Native frame
GIS and Earth observation	Global geographic (WGS-84, ECEF)
AR / XR devices	Device-local SLAM frames
Game engines and virtual worlds	Scene graphs

Domain	Native frame
Robotics	Odometry and map frames
Neural reconstruction (NeRF, Gaussian splat)	Reconstruction frames

These frames do not align natively. Every pair of systems that needs to interoperate must re-project, re-register, or re-anchor. The result is brittle: shared AR drifts, digital twins lose registration, multi-agent systems disagree about where things are, and AI systems built on top inherit the inconsistency.

The existing standards landscape provides important pieces — OGC GeoPose for position and orientation, OGC DGGS for global discrete indexing, ISO 8601 for time, and emerging work on moving features. What is missing is an identity and addressing layer that composes these pieces into a single stable reference per entity, with the runtime properties (low latency, GPU friendliness, AI legibility) that modern spatial computing demands.

4D-ID is that layer.

---

2. Architecture

2.1 The hierarchical model

4D-ID models spatial reality as a nested transform hierarchy. Each node in the hierarchy has:

- An identity (the 4D-ID itself)
- A reference to its parent in the hierarchy
- A local coordinate frame
- A transform expressing its pose relative to its parent
- A timestamp marking the validity of that pose
- Optional motion state
- Optional confidence and provenance metadata

The hierarchy spans from planetary scale (Earth, anchored to DGGS) down through regional, facility, room, object, and sub-object scales. Virtual entities can be anchored anywhere in the same hierarchy as physical ones.

The base specification treats the hierarchy as a strict tree: each entity has exactly one parent. This is the model assumed by conformance profiles P1–P3 (§10). Strict-tree hierarchies bound traversal cost at O(depth) and support the cached-composition optimization described in §2.5.

Implementations that require multi-anchor entities (a virtual overlay simultaneously anchored to a physical landmark and to a user's session frame, for example) must declare conformance to a profile that explicitly permits multi-anchor graphs. Such profiles carry additional well-formedness and traversal requirements; the multi-anchor formalism is reserved for a future revision.

Profiles supporting strict-tree hierarchies must enforce a maximum depth of 12 levels. This bound is sufficient for any realistic spatial composition (planet → region → city → district → facility → building → floor → room → fixture → object → sub-component → feature) and prevents pathological deepening from degrading global query performance.

2.2 Identity

Every entity in the hierarchy carries a 4D-ID: a stable, persistent identifier that does not change as the entity moves, ages, or is observed by different consumers. A delivery vehicle has one 4D-ID across its operational lifetime; a building has one 4D-ID from construction through renovation; a virtual overlay has one 4D-ID across the sessions in which it is rendered.

A 4D-ID is composed of three logical parts:

1. **A global anchor** — derived from a DGGS cell ID and an optional vertical reference. This places the entity in Earth-anchored space at the resolution appropriate for its scale.
2. **A local descriptor** — identifying the entity within its local frame. This may be a generated identifier, a hash, or a pseudonymous label depending on profile (see §5).
3. **A genesis marker** — an ISO 8601 timestamp recording when the identity was minted. This is distinct from the pose timestamp carried in a 4D-State record and does not advance as the entity moves.

The 4D-ID is opaque to most consumers. Systems that need to act on an entity reference its 4D-ID; systems that need to render or reason about its current condition consume a 4D-State record (see §2.3).

2.3 State records and identity continuity

A 4D-State is the transmitted record that represents an entity at a moment. It carries the entity's 4D-ID, its current pose, and any active extensions (motion, history, verifiable presence). Two 4D-State records sharing the same 4D-ID describe the same entity at different moments.

Each 4D-State record carries three timestamps rather than one:

- **Source time:** the timestamp at which the producer's clock observed the pose. Reflects the producer's view of reality.
- **Publish time:** the timestamp at which the record was emitted into the propagation layer. May differ from source time when batching, retry, or store-and-forward apply.
- **Receive time:** stamped by each receiving system on arrival. Not propagated downstream; rewritten at each hop.

Consumers reason about all three. Source-to-publish gap indicates producer-side delay (queueing, batching). Publish-to-receive gap indicates network delay. Total source-to-receive gap is what determines staleness. This three-timestamp model is required by all conformance profiles because clock skew between producers at planetary scale cannot be assumed bounded below ~10ms even with NTP, and many edge devices are much worse.

This distinction supports identity continuity through change. Pose changes — the entity moves, rotates, or is re-measured — produce new 4D-State records under the same 4D-ID. Structural changes — a building gains a wing, a vehicle is rebuilt, a virtual entity is forked — produce a new 4D-ID with a derivation reference back to the prior identity. The prior 4D-ID remains queryable; consumers can resolve the chain of derivations when temporal continuity is needed.

State records additionally carry a per-producer sequence number, monotonic per 4D-ID. Sequence numbers enable consumers to detect dropped records (gap detection) and out-of-order delivery without relying on timestamps, which may have drifted.

Each entity declares a motion mode that governs how its 4D-State records should be interpreted:

- **Static:** the entity does not move. Consumers must not extrapolate or interpolate. The motion extension, if present, is not normative.
- **Physical:** the entity moves under continuous physical constraints — bounded velocity, continuous trajectory, predictable extrapolation. The motion extension carries meaningful velocity and acceleration data. Consumers may interpolate between state records and extrapolate forward within the motion extension's confidence window.
- **Teleportable:** the entity may undergo discrete pose changes (teleports, scene cuts, virtual-camera jumps) that violate physical continuity. The entity is physical-mode during normal motion and explicitly signals each discontinuity.

When a teleportable entity undergoes a discrete jump, the resulting 4D-State record carries a discontinuity marker referencing the prior pose. Consumers receiving a record with this marker must invalidate any predictions derived from the prior pose and must not interpolate between the two poses. The marker is semantically distinct from the sequence-number gap detection used to identify dropped records: the gap is unintentional and represents missing data, while the discontinuity is intentional and represents a deliberate non-physical motion.

The motion mode is part of the entity's stable metadata, not part of its state record. An entity that is sometimes physical and sometimes teleportable (a remote visitor walking in their physical room and teleporting in the virtual space) declares teleportable mode and uses the discontinuity marker selectively. Producers must not silently switch an entity's declared mode between updates.

This specification does not mandate a maximum derivation depth. Implementations may prune long derivation chains under documented retention policies, provided the prior identities remain resolvable through the policy's defined window.

2.4 Foundational alignment with OGC standards

4D-ID is designed to compose with, not compete with, existing OGC work:

OGC DGGS provides the global indexing layer. Every 4D-ID's global anchor resolves to a DGGS cell at an appropriate resolution. This gives 4D-ID immediate compatibility with the DGGS API (OGC 21-038) for spatial query, storage, and aggregation. Cell resolution is selected per use case; planetary overviews live in low-resolution cells, room-scale interactions in high-resolution cells.

OGC GeoPose 1.0 provides the position and orientation representation carried in each 4D-State. The 4D-State's pose is expressed as a GeoPose-compatible structure, ensuring interchange with any system that already speaks GeoPose. 4D-ID adds the identity, hierarchical context, and extension framework that GeoPose alone does not provide.

ISO 8601 and OGC Moving Features provide the temporal layer. Static entities carry timestamps; dynamic entities optionally extend into the moving-features model for trajectory representation.

Alignment with the OGC stack is treated as foundational to this specification, not as an integration appendix. 4D-ID is positioned as the identity layer that lets GeoPose and DGGS be used together at runtime, in AI and XR contexts, in a way the standalone standards do not directly enable.

2.5 Local Coordinate Systems

Global coordinates are computationally unwieldy for the millimeter-precision interactions XR and robotics demand. 4D-ID introduces Local Coordinate Systems (LCS) as small, stable frames anchored to a parent 4D-ID.

An LCS provides:

- A bounded coordinate space (typically meters-to-hundreds-of-meters extent)
- A transform expressing its pose relative to its parent 4D-ID
- A validity window during which the local frame is consistent
- A synchronization tolerance describing how often it must reconcile with the parent
- A cached composed transform to global coordinates, refreshed within the validity window

An LCS is anchored to the smallest DGGS cell that fully contains its extent. This rule resolves the parent-cell ambiguity that arises when an LCS straddles multiple cells at a given resolution: the implementation selects the next-coarser resolution at which a single cell contains the extent. When an LCS subsequently expands beyond its parent cell, continuity is preserved by migrating to the next-coarser cell and carrying a derivation reference; the LCS identity itself does not change.

The cached composed transform is the performance-critical optimization at scale. Without it, every query that needs an entity's global pose walks the parent chain and composes transforms at each level. At billions of entities and millions of concurrent queries, chain-walking dominates query cost. With caching, the global pose is read directly from the LCS; the cache is refreshed when the validity window expires or when any parent in the chain emits a structural update. Implementations must support cached composed transforms for any profile permitting hierarchy depth greater than 4.

LCS frames are GPU-friendly by design. Fine-grained geometry — Gaussian splats, mesh vertices, avatar skeletons, sensor point clouds — lives in LCS frames. Global consistency is preserved by chaining transforms up the hierarchy when needed; most runtime computation never leaves a single LCS.

---

3. Real-to-virtual registration

4D-ID treats real and virtual geometry as first-class peers in the same hierarchy.

Real-world reconstructions — Gaussian splats from photogrammetry, point clouds from LiDAR, meshes from SLAM — are each anchored to a 4D-ID, given an LCS for their internal coordinate frame, and registered to the global DGGS layer through GeoPose alignment.

Virtual content — game objects, AR overlays, avatars, simulation entities — attaches its own LCS frames to the same 4D-IDs as the real reconstructions. A virtual object pinned to a real table shares an anchor with that table; an avatar walking through a real room moves through the same coordinate substrate the room itself lives in.

Mixed entities — for example, an AR hologram bound to a physical desk — carry dual registration, with a 4D-ID referencing both the physical anchor and the virtual scene.

This creates a shared spatial substrate where multiple users, multiple devices, and multiple AI agents can reason about the same world without each maintaining its own private coordinate frame.

The registration quality — RMS error, source provenance, time of last update — is carried in the 4D-ID's metadata. Consumers can filter or weight by quality without needing out-of-band information.

---

4. Extensions

The core 4D-ID is intentionally minimal: identity, hierarchy, pose, time. Three extensions provide capability that subsets of users need without burdening the base spec.

4.1 Motion extension

For entities whose pose changes over time, the motion extension carries velocity, optional angular velocity, and optional acceleration. The motion extension is sampled at a specific timestamp; consumers can extrapolate forward or interpolate within a trajectory window.

The motion extension is normative for entities in physical motion mode (§2.3). For entities in teleportable mode, the motion extension is normative during continuous-motion intervals and undefined across discontinuities; consumers must consult the discontinuity marker before applying motion-extension data across a state-record gap. For entities in static mode, the motion extension carries no normative meaning and consumers must not extrapolate.

The motion extension carries a confidence value sampled at the source-time of the state record. Consumers extrapolating an entity's pose forward from source-time to a later moment (rendering deadline, prediction horizon, query time) must apply a confidence decay as a function of the elapsed extrapolation interval. The decay function is implementation-defined but must be documented, monotonic in the elapsed interval, and exposed to downstream consumers as a decayed-confidence field on the extrapolated pose. Renderers and decision systems then have a clean signal for when to degrade extrapolated entities visually (fading, outlining, suppressing) rather than presenting low-confidence predictions as if they were authoritative.

This requirement matters most for AR and XR rendering, where the gap between a state record's source-time and the rendering deadline routinely exceeds 30ms and may grow to hundreds of milliseconds during connectivity degradation. Without confidence decay, downstream renderers cannot distinguish a fresh confident prediction from a stale extrapolation, and the user experience fails silently.

Static entities omit motion entirely and remain fully conformant. Systems that cannot process motion data treat the fields as absent without compatibility break.

The motion extension composes cleanly with OGC Moving Features for systems that need full trajectory representation.

4.2 History and prediction extension

For applications that require temporal continuity — robotic planning, multiplayer synchronization, historical replay, AI reasoning over change — the history and prediction extension carries:

- A bounded window of prior poses
- A bounded window of predicted future poses
- A confidence value for each prediction
- A model identifier indicating how predictions were generated

Predictions are explicitly advisory. This specification requires that predictions be marked as such and that confidence be communicated. Consumers must treat predictions as estimates and degrade gracefully when prediction quality is low or absent.

This is the right place to flag a design constraint: 4D-ID does not specify *how* predictions are generated. That is left to implementations. This specification ensures predictions can be communicated in a standard way; it takes no position on what model produces them.

Two scaling considerations apply to this extension at planetary scale:

History storage. Per-entry retention metadata is the conformance requirement (§5.3). For implementations holding history at scale, bucket-level retention is the recommended pattern: history is stored in fixed-duration buckets (typically 1 hour), and retention is enforced by dropping whole expired buckets rather than checking each entry. Implementations supporting bucket-level retention declare the bucket granularity in their conformance statement; consumers querying history understand that resolution at the bucket boundary may degrade.

Prediction materialization. Predictions may be ephemeral — generated at read time from a documented motion model and the entity's current state — or materialized as stored future-state records. The ephemeral form is recommended at scale because materialized predictions otherwise multiply storage by the prediction-window size. Conformance requires that consumers cannot distinguish ephemeral from materialized predictions through the data interface; both must carry the same fields and confidence semantics.

Profile P3 (§10) permits per-entity opt-in to history and prediction rather than mandating either globally. An implementation conforming to P3 may carry history+prediction for a subset of its entities (typically dynamic, high-value, or compliance-sensitive ones) and omit both for the remainder, with the per-entity capability declared in the entity's metadata.

4.3 Verifiable presence extension (optional)

Some use cases require cryptographic attestation that a specific entity occupied a specific pose at a specific time — for compliance in regulated drone corridors, for chain-of-custody in autonomous logistics, for provenance in AI training data sourced from spatial captures, for event verification in commercial XR.

The verifiable presence extension provides a signed-attestation structure that wraps a 4D-ID with:

- A signature from a known identity (operator, sensor, agent)
- A hash binding the pose, time, and identifier
- Optional integration with W3C Verifiable Credentials

This extension is deliberately separable from the core specification. Implementations that do not need verifiable presence omit it entirely. Implementations that do need it have a standard structure to use.

A design note on this extension: the underlying need — cryptographically anchored proof of presence — is real and recurring across compliance, logistics, and provenance contexts. Earlier approaches to this problem assumed proofs must be minted as on-chain tokens, which introduced ledger economics and chain-of-trust assumptions the spatial use case does not require. Signed attestations with W3C Verifiable Credential compatibility cover the same use cases at lower cost. Implementations that choose to anchor attestations to a public ledger may do so as a deployment choice; this specification does not require it.

---

5. Identity, privacy, and access

A spatial identity layer with precise pose, history, and prediction is, in deployment, indistinguishable from a surveillance system unless it carries explicit privacy primitives. This section defines the primitives that conformant implementations must support when the entities being tracked include humans or human-associated objects (phones, wearables, vehicles, residences).

Three concepts are required:

5.1 Pseudonymous identifiers

The local descriptor component of a 4D-ID (§2.2) may be a stable pseudonym rather than a human-meaningful label. A pseudonym is generated such that the binding between the 4D-ID and the underlying real-world identity (a person's name, a phone's IMEI, a vehicle's VIN) is held only by parties with explicit credentials to dereference it.

This specification does not mandate a specific pseudonymization scheme. It mandates that implementations dealing with human-associated entities support pseudonymous 4D-IDs as a profile-level capability and document the dereferencing-credential model in deployment.

5.2 Scoped disclosure

A single 4D-State record may carry pose at multiple resolutions, disclosed to different consumers based on their credentials. A logistics platform might disclose a delivery vehicle's pose to the customer at the level of “within two blocks” and to the dispatcher at meter-level precision. The vehicle has one 4D-ID; the disclosed precision varies by consumer.

Implementations supporting scoped disclosure carry a disclosure policy reference in the 4D-State and resolve precision at delivery time based on the requesting consumer's credentials. The policy reference is opaque to this specification; specific policy languages are out of scope.

5.3 Retention and time-to-live

The history extension (§4.2) carries prior poses indefinitely unless retention is bounded. This specification requires that the history extension include a retention metadata field for each entry, expressed as either an absolute expiry timestamp or a time-to-live interval from the entry's validity timestamp. Consumers must honor retention metadata: expired entries must be deleted from local stores and must not be propagated to downstream systems.

Retention applies to predictions as well. Prediction entries carry expiry timestamps; predictions that have passed their expiry without being confirmed against an observed pose must be discarded rather than retained as historical data.

The verifiable presence extension (§4.3) is treated specially: attestations carry their own retention semantics defined by the issuing identity and the use case. A drone-corridor compliance attestation may be retained for the regulatory window; an event-verification attestation may be ephemeral. Implementations must distinguish operational history (subject to retention) from attestation records (subject to the attestation's own retention policy).

5.4 Profile-level requirements

Conformance profiles (§10) layer privacy requirements as follows: P1–P3 implementations dealing with human-associated entities must support pseudonymous identifiers and retention metadata. P4 and P5 implementations dealing with human-associated entities must additionally support scoped disclosure. Implementations dealing exclusively with non-human-associated entities (geographic features, infrastructure, machinery without human operators) may declare the privacy requirements not applicable in their conformance statement.

---

6. AI and runtime considerations

4D-ID is designed for AI-readable spatial computing and for the interactive query patterns that AR, XR, and robotics impose. This section describes the design properties that make the framework AI-legible, then specifies two normative query types — ray-casting and visibility — that close the gap between “entities exist in the substrate” and “the rendering pipeline can find what it needs to draw.”

6.1 Design properties

Four design choices make 4D-ID AI-legible:

Embedding compatibility. A 4D-State's pose, time, and motion fields compose naturally into a fixed-length vector suitable for use in modern vector databases and retrieval-augmented systems. This specification defines the canonical vector layout per profile so that embeddings produced by different implementations remain comparable.

Hierarchical query. Because 4D-ID is hierarchical and the hierarchy is rooted in DGGS, spatial queries can be expressed at any level of resolution. “All entities in this DGGS cell at this time” is a primitive operation; so is “all child frames of this LCS.” This composes well with retrieval-augmented generation pipelines that need to ground language model output in spatial context.

Multi-resolution indexing. DGGS cell density varies by many orders of magnitude across the planet: dense urban cells may contain 10^6 entities at high resolution while remote cells at the same resolution contain none. Implementations indexing by DGGS cell must support multi-resolution indexing to avoid hot-spotting on dense cells. Queries should be expressible at the coarsest resolution that returns a manageable result set, with implementations free to subdivide hot cells dynamically. The recommended pattern is a tree of indexes keyed by parent cell, where dense cells carry sub-indexes at finer resolution and sparse cells aggregate up.

Stable identity over time. A 4D-ID's identity persists across pose updates. An entity that moves retains the same identity; its 4D-State records change, but downstream systems can track the same entity through change. This matters for any AI system reasoning about entities rather than instantaneous observations.

6.2 Ray-casting

A ray-cast query takes a ray expressed in either global coordinates (origin position + direction) or local frame coordinates (origin 4D-ID + direction in that frame's local axes) and returns the entities the ray intersects, ordered by distance from the origin.

Ray-casting is the query primitive for use cases where a user or system points at something and needs to know what it is. Spherical-image or video tap-to-identify is the canonical example: the user taps a pixel on a 360 image, the UI converts the tap to a ray in the capture point's local frame, and the ray-cast returns the entity at that direction. AR cursor selection, drone targeting, and beam-based interaction patterns all reduce to ray-casts against the spatial substrate.

A ray-cast returns:

- An ordered list of 4D-IDs intersected by the ray, sorted by distance from the origin
- For each hit, the intersection point expressed in the hit entity's local frame
- For each hit, the same provenance metadata that direct entity queries return — registration quality, source, time of last update
- A combined-uncertainty value reflecting the convolution of the ray's origin uncertainty with the hit entity's geometry uncertainty

Implementations may cap the result list at a documented maximum count and may cap the ray distance at a documented maximum range. When the ray passes through entity hierarchies (a building containing floors containing rooms), implementations may return either the outermost intersected entity or recurse into children to return the finest-grained hit; the choice is documented per implementation and may be specified by the query caller.

Ray-casting against Gaussian-splat reconstructions is supported (§3): each splat cluster has its own 4D-ID and the ray-cast returns the splat hit point. Implementations exposing splat-level hit detail must document the geometric model used (point-cloud nearest-neighbor, surfel intersection, signed-distance approximation).

6.3 Visibility query

A visibility query takes a viewpoint (pose plus field-of-view parameters) and returns the entities visible from that viewpoint, ordered by distance and optionally with occlusion considered. Visibility queries are the primitive for AR and XR rendering: the device asks the substrate “what do I need to render right now” and gets back the entities in view, at a level of detail appropriate to each entity's distance and importance.

A visibility query returns:

- An ordered list of 4D-IDs visible from the viewpoint, sorted by distance
- For each entity, a level-of-detail hint based on its angular size at the viewpoint
- An occlusion flag indicating whether the entity is fully visible, partially occluded, or culled by an opaque occluder closer to the viewpoint

Implementations must support frustum culling against the field-of-view parameters and must support distance-based culling against a documented maximum range. Occlusion testing is optional but, when supported, must use entity bounding extents or geometric hulls rather than just point positions; occlusion based on point distance produces incorrect results when occluders are large.

Visibility queries are expected to run at interactive rates — typically once per rendered frame, which on AR devices means 30–90 Hz. Implementations supporting visibility queries must declare their maximum sustainable query rate and the entity-count bound at that rate. Above the bound, visibility queries are subject to the same overload handling as state-record propagation (§7.3).

This specification does not mandate per-entity visibility tiering within a single query (foveated visibility, where entities in the central field of view return at higher fidelity than peripheral entities). Implementations may support this, and a future revision may specify it.

This specification does not mandate a specific vector database, embedding model, or AI runtime. It defines the data shape that makes such systems possible.

---

7. State propagation

4D-State records flow from producers (sensors, simulation engines, AI agents) to consumers (renderers, planners, analytics, other agents). At planetary scale this flow exhibits the same

properties that distinguish modern stream-processing systems from request-reply architectures: the producer count is unbounded, the consumer interest is spatial rather than entity-keyed, and the update rate dominates the query rate by orders of magnitude. This section describes the propagation pattern that conformant implementations should follow.

7.1 The recommended pattern

Producers write 4D-State records to a region-keyed log. Consumers subscribe to regions of interest. Routing is by DGGS cell at a resolution chosen per deployment. This pattern — region-keyed publish-subscribe with spatial indexing — scales to the workloads this specification targets and composes naturally with the hierarchical query model (§6).

Specifically:

- Producers tag each record with the DGGS cell at the resolution declared by the entity's anchor
- Records are written to a topic or partition keyed on that cell
- Consumers express interest in regions (one cell or a contiguous set) and receive records for those regions
- Cell resolution at the routing layer is independent of cell resolution at the indexing layer; routing typically uses coarser cells than indexing

7.2 Interest management

A consumer's spatial interest typically changes over time — an AR device moves through a city, a robot navigates a warehouse, an AI agent shifts attention. Interest management is how the propagation layer tracks and updates these subscriptions efficiently.

Conformant implementations must support interest expressions at the granularity of DGGS cells. Implementations may additionally support derived interest expressions (radius around a moving point, viewshed from a pose, predicted future positions) that resolve to DGGS cell sets at evaluation time. The base interest expression is the cell set; richer expressions are conveniences over it.

7.3 Update fan-out and overload

A high-density cell (a city block at high resolution) may have thousands of producers and millions of consumers. Naive fan-out (every record to every consumer) is infeasible at this scale. Implementations must support at least one of the following fan-out strategies:

- **Aggregated fan-out:** records are batched at the cell boundary and delivered as bundles. Reduces per-record overhead at the cost of latency.

- **Sampled fan-out:** records are delivered at a reduced rate to consumers whose declared latency tolerance allows. Producers continue at full rate; consumers receive a documented sample.
- **Tiered fan-out:** premium consumers receive full rate; standard consumers receive a downsampled stream. The tier is declared at subscription time and visible in the consumer's conformance statement.

Backpressure semantics are mandated: when a consumer cannot keep up, the propagation layer must drop records rather than block producers. Dropped records are signaled to the consumer with a documented dropped-record indicator; the consumer reasons about completeness from the indicator. Producer-side blocking on a slow consumer is a non-conformant behavior at scale.

7.4 Partition behavior

Network partitions between regions are routine at planetary scale. This specification mandates that on partition, each side continues operating with the records it has, marking propagated state from the unreachable side as stale. On partition heal, the rejoining side reconciles state through the failure handling defined in §8.

This specification leans toward availability over consistency on partition: state remains queryable on each side, with staleness explicit in the timestamps and sequence numbers carried by each record. Implementations requiring consistency over availability must declare so in their conformance statement and document the consistency model they implement.

---

8. Failure model

Conformant implementations must handle the following failure classes in defined ways. Other failure modes may occur; this section enumerates the cases for which this specification mandates handling.

The failure classes fall into three groups: structural failures (data integrity issues within the protocol), connectivity failures (issues with the propagation layer between participants), and operational failures (issues arising from load or clock disagreement).

8.1 Structural failures

Failure class	Required handling
unreachable_parent	A 4D-State references a parent 4D-ID that cannot be resolved within the synchronization tolerance. Implementations must mark the state as unanchored and continue operating in the local frame. Composition up the hierarchy is suspended; downstream consumers must be informed

Failure class	Required handling
	that global registration is degraded.
pose_conflict	Two 4D-State records under the same 4D-ID at overlapping timestamps disagree on pose beyond the validity tolerance. Implementations must surface the conflict to consumers rather than silently selecting one. Resolution policy (last-writer-wins, source-priority, manual reconciliation) is implementation-defined but must be documented.
stale_attestation	A verifiable presence attestation arrives with a valid signature but a timestamp older than the attestation's freshness window. Implementations must reject the attestation as proof while preserving it as historical record. Downstream systems acting on attestations must be informed of the staleness.
resolution_mismatch	Two implementations exchanging 4D-State records operate at incompatible DGGS resolutions for a shared region. Implementations must coerce to the coarser of the two resolutions via the algorithm specified in §8.4 and emit a loss-of-precision indicator on the coerced records.
hierarchy_loop	A chain of parent references forms a cycle. Implementations must detect cycles at write time and reject the offending state record. Reads must be robust to cycles in stored data (defensive traversal with depth bounds).

8.2 Connectivity failures

Connectivity failures are particularly relevant at planetary scale, where producers and consumers span fiber-backed datacenters, mobile networks, satellite links, low-power wide-area networks, and devices that go offline entirely for extended periods. Conformant implementations distinguish these cases rather than collapsing all communication problems into a single error.

Failure class	Required handling
intermittent_disconnect	A consumer loses propagation-layer connectivity for seconds to minutes, then reconnects. On reconnect, the consumer must detect gaps in per-producer sequence numbers and either (a) fetch missing records from the propagation layer's replay buffer if available, or (b) mark the gap explicitly and reason about state across the gap with degraded confidence.
extended_offline	A producer or consumer is offline for minutes to hours but is known to be still operating (a drone in flight, a robot in a basement, a device in a tunnel). The entity's last-known state remains queryable, marked with offline_active status and the source-to-now gap. On reconnect, the producer's accumulated state records are replayed in order; consumers reconcile the replay against any state they predicted in the interim.

Failure class	Required handling
lossy_channel	Records arrive but a fraction are dropped in transit. Consumers detect loss through per-producer sequence number gaps and treat dropped records as missing data, not as evidence of state change. The propagation layer must report measured loss rate to subscribers; consumers above a documented loss threshold must degrade gracefully or escalate.
partition	Two regions of the network cannot reach each other. Each side continues operating per §7.4; state from the unreachable side is marked stale at the partition boundary. On heal, gap detection and replay apply as in intermittent_disconnect, but at the region level.
asymmetric_reachability	Participant A can reach B but B cannot reach A. Implementations must not assume bidirectional reachability for propagation purposes. Subscriptions, acknowledgments, and replay requests that require return paths must use the same channels they were established on, not assume a reverse channel exists.
lost	A producer has been offline for longer than its declared lost-threshold and is presumed terminated or destroyed. The entity's state moves to terminal status; further state records under that 4D-ID are rejected. Consumers querying historical state can still retrieve it through the history extension within its retention window.

8.3 Operational failures

Failure class	Required handling
timestamp_drift	A 4D-State record arrives with a source-time that is implausibly far from the receiver's observed-arrival time. Implementations must not silently rewrite or correct the source-time; they must accept the record, flag the implied drift, and downstream consumers may filter or weight by the flag. The three-timestamp model in §2.3 provides the data needed to reason about drift without forcing correction.
overload	Producer rate or consumer demand exceeds the propagation layer's capacity. Implementations must degrade gracefully via the fan-out strategies in §7.3: aggregated batching, sampled delivery, or tiered fan-out. Producers must not block on slow consumers. Dropped records under overload are signaled to consumers via the dropped-record indicator from §7.3.
velocity_violation	A 4D-State record for an entity in physical motion mode reports a pose that is inconsistent with the entity's prior pose and declared velocity bounds, with no accompanying discontinuity marker. Implementations must accept the record, flag the violation, and forward it to consumers with the flag attached. Rejection is not permitted because the cause may be a malfunctioning sensor, a spoofed pose, or an undeclared mode change — and consumers reasoning about authenticity

Failure class	Required handling
	(especially verifiable presence) need to see the anomaly rather than have it filtered. For entities in teleportable mode, motion that would otherwise violate physical bounds is expected and signaled by the discontinuity marker (§2.3); absence of the marker is what makes a teleportable entity's discontinuity a violation.

8.4 Resolution coercion algorithm

When two implementations exchange records at incompatible DGGS resolutions for a shared region, the receiving implementation applies the following coercion procedure:

1. Identify the coarser of the two resolutions in the affected region.
2. For each finer-resolution record, compute its containing cell at the coarser resolution.
3. Rewrite the record's anchor to reference the coarser cell, preserving the original LCS transform.
4. Stamp the record with a loss-of-precision indicator carrying the original (finer) resolution.
5. Propagate to downstream consumers with the indicator present.

Consumers receiving coerced records may upsample if they have access to the original-resolution source; otherwise they treat the record at the coerced resolution. The indicator persists with the record and is propagated alongside it.

All fourteen failure classes are normative. Implementations claiming conformance must document their handling of each class. Profile-level conformance (§10) does not vary by failure class; all profiles inherit these handling requirements.

---

9. Performance and scale

4D-ID is designed for high-performance runtime use. Key properties:

GPU friendliness. LCS transforms are 4×4 matrices in conventional layout. Pose fields use canonical floating-point representations. Vector embeddings are sized to common GPU thread block widths.

Locality of computation. Most operations happen within a single LCS and do not require traversing the global hierarchy. Global synchronization is periodic, not per-frame.

Bounded synchronization cost. This specification defines synchronization tolerances per profile (§10). Implementations can trade off latency against consistency within those bounds without losing conformance.

Scale targets. The framework is designed to support order-of-millions of concurrent active entities in a shared deployment, with the bottleneck being implementation choices about index structure and update propagation rather than this specification itself. Specific throughput numbers are deployment-dependent and not normative.

---

10. Conformance profiles

Profile	Required capabilities
P1 — Static	Core identity, hierarchy, GeoPose-compatible pose, ISO 8601 timestamp.
P2 — Dynamic	P1, plus motion extension.
P3 — Temporal	P2, plus history and prediction extension.
P4 — Verified	P3, plus verifiable presence extension.
P5 — AI-Integrated	P3 (or P4), plus canonical embedding layout and DGGS-rooted hierarchical query support.

Profiles are layered. An implementation conforming to P3 also conforms to P1 and P2. P4 and P5 are independent extensions of P3; an implementation can conform to either, both, or neither.

Implementations dealing with human-associated entities must additionally satisfy the privacy requirements layered on top of these profiles (see §5.4).

A claim of conformance must specify the profile and document handling of all failure classes in §8. Generic claims of “4D-ID conformance” without a profile and failure-handling statement are insufficient.

---

11. What 4D-ID is not

To avoid common misreadings:

4D-ID is not a coordinate system. It is an identity and addressing layer on top of coordinate systems. It uses WGS-84, ECEF, local frames, and DGGS cells; it does not replace them.

4D-ID is not a replacement for GeoPose or DGGS. It is an integration layer that lets those standards be used together at runtime, with hierarchical identity and the extensions modern spatial computing needs.

4D-ID is not a perception system. It does not generate poses; it represents them. Pose generation is the domain of SLAM, photogrammetry, reconstruction pipelines, and sensor fusion. 4D-ID is what those systems output into and what downstream systems consume from.

4D-ID is not a rendering format. It does not specify how Gaussian splats, NeRFs, meshes, or any other geometry are represented internally. It specifies how those representations are anchored, identified, and composed.

4D-ID is not a blockchain. The verifiable presence extension allows cryptographic attestation; it does not require any specific ledger or token economy.

4D-ID is not a discovery mechanism. It does not tell consumers what entities exist or where to find their 4D-IDs. Pairing this specification with a registry, directory, or discovery layer is a deployment concern, not a specification concern.

---

12. Acknowledgments and references

This specification builds on:

- OGC GeoPose 1.0 (OGC 21-056r11)
- OGC Discrete Global Grid System Abstract Specification (OGC 20-040r3) and DGGS API (OGC 21-038)
- ISO 8601 temporal representations
- OGC Moving Features standards family
- W3C Verifiable Credentials Data Model 2.0

The verifiable presence extension is informed by emerging work on signed sensor attestations for autonomous systems, drone-corridor compliance, and AI training data provenance.

Appendix A. Future work and pending considerations

This appendix is non-normative. Implementations do not need to do anything about appendix content for conformance to this specification. The items below identify capabilities and design questions deferred from v1.0 to subsequent revisions, with the rationale for deferral and a sketch of the likely direction. Future revisions of this specification may absorb some of these items; others may remain pending indefinitely or be addressed through separate specifications.

A.1 Learned-embedding profile

§6.1 specifies that 4D-State fields compose into a canonical pose vector suitable for vector databases. Modern spatial AI work — particularly joint-embedding predictive architectures and

related self-supervised approaches — produces additional learned semantic embeddings that capture what an entity is like, not just where it is. Implementations currently attach such embeddings to state records ad hoc; interoperability across implementations is not specified.

A future profile would specify how learned embeddings attach to 4D-State records alongside the canonical pose vector. Each learned embedding would be tagged with the model identifier that produced it, version itself independently of the entity (the model changes; the entity does not), and be explicitly non-comparable across models (different models produce different spaces; conflating them is a category error).

Informative note: spatial AI systems built on joint-embedding predictive architectures currently lack a standard substrate for grounding their learned predictions in real-world spatial identity. Each major effort is building its own. A learned-embedding profile would position 4D-ID as that grounding substrate, complementary to rather than competing with the spatial AI work itself.

A.2 Foveated streaming and per-entity tiering

§7.3 specifies tiered fan-out at the consumer level: premium consumers receive full-rate streams, standard consumers receive downsampled ones, with the tier declared at subscription time. AR and XR workloads need a finer-grained version: per-entity tiering within a single consumer's subscription, where entities in the central field of view are delivered at higher fidelity than peripheral entities, all within the same subscription.

A future revision would specify the interest expression for per-region fidelity tiers, the propagation layer's routing behavior for records crossing tier boundaries, and the consumer-side handling of smooth transitions when entities walk from the fovea to the periphery. The right design is an active area of XR research; this specification leaves room for it without committing prematurely. §6.3 already anticipates the visibility-query side of foveation; future work would connect the visibility-query and propagation-layer mechanisms into a single coherent design.

A.3 Multi-anchor formalism

§2.1 restricts the base specification's hierarchies to strict trees. Some use cases — virtual overlays anchored simultaneously to physical landmarks and to user session frames, entities co-located in multiple semantic hierarchies — require multi-anchor graphs. A future revision would specify a profile-level opt-in for multi-anchor support, with documented well-formedness conditions, traversal semantics, transform-composition disambiguation when multiple anchors disagree, and bounded fan-out to prevent multi-anchor graphs from becoming arbitrary networks.

Specifying this well requires concrete driving use cases. The base specification's strict-tree restriction is appropriate until such use cases are clear enough to design against.

A.4 Pseudonymization scheme

§5.1 mandates that implementations dealing with human-associated entities support pseudonymous identifiers as a profile-level capability. The specific pseudonymization scheme is not specified, and implementations currently interoperate at the pose level but not at the pseudonym level. A future revision would select one or more reference schemes for interoperability.

Candidate schemes include k-anonymity grouping (pseudonyms shared across cohorts of indistinguishable entities), rotating pseudonyms (each identifier valid for a documented window), and attribute-based credentials (pseudonyms cryptographically tied to credentials rather than to a registry). The most likely outcome is selection of one reference scheme with the others permitted as opt-in extensions. This work requires input from privacy-engineering practice; the choice has compliance implications in regulated jurisdictions that are not visible in purely technical review.

A.5 Disclosure policy language

§5.2 specifies that 4D-State records may carry pose at multiple resolutions disclosed to different consumers based on credentials, with the disclosure policy reference treated as opaque to this specification. A future revision would either define a normative policy language for scoped disclosure or select an existing language (XACML, OPA Rego, or a successor) as the reference.

Selecting an existing language is the more likely path. Policy languages are a hard design problem, existing tooling matters substantially for adoption, and standardization on a language already in production use is lower-risk than specifying a new one. The selection is a v1.1 or later decision.

A.6 DGGS variant selection

This specification is compatible with multiple DGGS variants — H3, S2, ISEA, and others as they emerge. v1.0 does not mandate one. Different variants have different properties: H3 has the widest operator ecosystem and well-defined neighbor semantics; S2 is more mathematically clean; ISEA is the original OGC reference. The right choice depends on what OGC standardizes on and on what implementing partners prefer.

A future revision will either mandate a specific variant at the profile level or specify a translation layer between variants. A translation layer is more flexible but adds complexity; mandation is simpler but more constraining. This decision is likely to follow standards-body engagement rather than precede it.

A.7 Standards-body engagement

v1.0 is shared as a working draft. A decision is pending on whether to pursue this specification as an OGC extension, as a joint OGC and Khronos working item, as an independent specification with informative references to OGC standards, or as a proprietary specification published under permissive license. Each path has implications for the scope and detail of subsequent revisions, for the licensing posture of implementations, and for the timeline of adoption.

This is the most consequential of the pending considerations because it shapes how all the others get resolved. A specification being ratified by a standards body progresses differently than one being maintained by a single author. The decision is expected before significant work on v1.1 begins.

A.8 Reference implementation

Literal serialization — JSON schemas, binary formats, API surface specifications — is deferred to the reference-implementation phase. The current draft describes data shape and semantics; concrete on-the-wire formats are a downstream concern. A reference implementation, when produced, will fix the serialization choices that this specification leaves to implementations and will provide a conformance test suite.

Whether the reference implementation is maintained by a standards body or by the specification's author depends on the outcome of A.7.

End of 4D-ID specification.