RFC 0003 — USG Registry Architecture

A Standards-Track Specification for Canonical Rights Registries

Document Series: USG-RFC
Series Number: 0003
DOI: 10.5281/zenodo.17942833 **Updates:** RFC 0001, RFC 0002
Relation: isSupplementTo 10.5281/zenodo.17565793
Author: Scott Jellen (Independent Researcher)
Date: December 2025

Status: Standards-Track (Draft for Review and Implementation Feedback)

Status of This Memo

This document defines the **USG Registry Architecture**, a standards-track specification for canonical, machine-readable registries within the Universal Sports Graph (USG) ecosystem.

It updates RFC 0001 and RFC 0002 by normatively defining registry structure, canonical identifiers, object lifecycle semantics, federation rules, indexing formats, digest and integrity requirements, and key-registry integration used during entitlement validation and settlement operations.

This document is intended for public review and pilot implementation.
Distribution is unlimited.

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 RFC 2119.

1. Introduction

The Universal Sports Graph (USG) defines a protocol suite for interoperable representation of sports rights, event metadata, access control, and settlement.
RFC 0001 introduced the architectural model for rights-as-data and access-as-API.
RFC 0002 defined the Entitlement Token Profile used to grant, verify, and audit access to events across distributed platforms.

This document — RFC 0003 — defines the **Registry Architecture**, the canonical data layer that all USG implementations rely on. A USG registry provides authoritative, structured, machine-readable records for:

- leagues, teams, and venues
- broadcasters and distributors
- rights bundles
- events and schedules
- key registries and trust anchors
- index files and digest manifests used for validation

Without a formal registry architecture, USG components cannot interoperate reliably: event identifiers cannot be resolved, entitlements cannot be validated against canonical sources, rights constraints cannot be enforced, and clearinghouse operations cannot audit or reconstruct settlement behavior.

The Registry Architecture supplies these missing foundations by defining:

- object types and canonical identifiers
- versioning, supersession, and withdrawal semantics
- canonical JSON requirements and SHA-256 digests
- federation and authority rules
- index structures for efficient discovery
- key registry integration for signature verification
- validation requirements for implementers

RFC 0003 completes the minimum viable USG protocol stack:

1. **RFC 0001** — Architecture
2. **RFC 0002** — Entitlement / Access Control
3. **RFC 0003** — Registry / Canonical Data Layer

---

2. Scope

This RFC specifies the architecture, structure, and operational requirements for USG-compliant registries. Specifically, it defines:

- registry object types and required fields
- canonical identifier rules
- versioning, supersession, and withdrawal semantics
- canonical JSON and digest requirements
- registry directory and indexing formats
- federation, mirroring, and authority models
- key registry objects and trust-anchor publication
- validation requirements for consumers and downstream systems
- minimum expectations for read-only registry APIs

Out of scope for this document:

- business rules, pricing models, or commercial terms
- internal storage schemas or proprietary backend implementations
- non-sports generalizations of the registry model
- entitlement token details (covered by RFC 0002)
- clearinghouse settlement mechanics (covered in future RFCs)

Registries implementing this specification MUST comply with all normative requirements contained in Sections 4–13, as applicable to their role as registry operators or consumers.

---

3. Terminology

This section defines the terms used throughout this specification.
All normative language uses RFC 2119 keywords.

3.1 Registry Operator

The entity responsible for publishing and maintaining a USG-compliant registry.
A registry operator MAY be a league, federation, delegated service provider, or clearinghouse.

3.2 Authoritative Registry

A registry whose contents are considered canonical for a defined scope
(e.g., a league or competition).
Authoritative registries publish objects, indexes, and digest manifests that downstream systems rely upon for entitlement validation and settlement.

3.3 Mirror Registry

A read-only replica of an authoritative registry.
Mirrors:

- MUST NOT introduce new objects within the authoritative scope.
- MUST synchronize using digest comparison or equivalent mechanisms.
- MUST clearly identify themselves as mirrors.

3.4 Aggregator Registry

A registry that aggregates objects from multiple authoritative registries, typically to provide cross-league indexes or unified lookup layers.
Aggregators MUST preserve original identifiers and MUST NOT alter authoritative semantics.

3.5 Registry Object

Any JSON document within a USG registry representing a domain entity such as:

- `league`
- `team`
- `venue`
- `broadcaster` or `distributor`
- `rights_bundle`
- `event`
- `key` (public key material for issuers and operators)

Registry objects MUST follow canonical JSON rules and MUST include required metadata fields, including `id`, `type`, `version`, and `meta`.

3.6 Object Identifier (OID)

The stable, globally unique identifier for a registry object within the USG namespace.
OIDs:

- MUST be immutable once assigned,
- MUST NOT be reused for semantically different entities, and
- SHOULD be treated as opaque strings by consumers.

3.7 Supersession

A versioning action where a new object **replaces** a previous version.
Superseded objects remain available for audit purposes but MUST NOT be used for new entitlements or settlements.

3.8 Withdrawal

A terminal state for objects declared invalid (e.g., cancelled events, compromised keys).
Withdrawn objects MUST NOT be referenced by current indexes and MUST NOT be considered valid for any new operational use.

3.9 Digest

A SHA-256 hash computed over the canonical JSON representation of an object or index.
Digests provide integrity assurance and enable mirror synchronization and validation.

3.10 Digest Index

A registry-wide document mapping object paths or identifiers to their SHA-256 digests.
Digest indexes are REQUIRED and form the integrity backbone for registry consumption.

3.11 Canonical JSON

A deterministic JSON encoding profile used for all registry objects and indexes.
Canonical JSON ensures that identical logical objects produce identical byte sequences and therefore stable digests.

3.12 Key Registry

A subset of registry objects that publishes public keys, rotation metadata, and status information for issuers and verifiers defined in RFC 0002.

3.13 Authority Scope

The set of entities (e.g., leagues, teams, events) for which a registry operator is responsible.
Authority scope determines which registry is considered canonical for a given identifier.

3.14 Conforming Registry

A registry that fully satisfies all REQUIRED and MUST-level requirements in this specification, including canonical JSON, object lifecycle semantics, digest publication, and indexing rules.

3.15 Registry Consumer

Any system that loads, validates, or resolves objects from a registry.
Consumers include platforms, distributors, verifiers, clearinghouses, and auditors.

3.16 Effective Snapshot

A consistent view of registry content at a specific point in time, defined by a matching set of digest values.
Snapshots allow deterministic entitlement validation and historical reconstruction.

---

4. Design Goals

The USG Registry Architecture is designed to ensure that all implementations operate from a shared, verifiable, and interoperable source of truth.
The following goals inform all normative requirements in this specification.

4.1 Determinism

Registry objects MUST be represented using canonical JSON so that:

- the same logical object always produces the same byte sequence;
- SHA-256 digests remain stable across builds and mirrors;
- downstream consumers can rely on deterministic integrity checks.

Determinism ensures consistent entitlement validation, settlement behavior, and auditability.

4.2 Identifier Stability

Object identifiers MUST remain stable once assigned.
Identifiers MAY be human-readable but MUST be treated as opaque by consumers.
Identifier stability enables predictable linking between registries, tokens (RFC 0002),
and clearinghouse workflows.

4.3 Auditability and Historical Reconstruction

Registries MUST preserve superseded and withdrawn objects to enable:

- reconstruction of historical entitlements;
- forensic analysis;
- regulatory or compliance review.

Digest indexes MUST allow consumers to validate snapshots of registry state at any time.

4.4 Integrity and Verification

All registry objects and index files MUST include or be referenced by SHA-256 digests.
Digest validation MUST allow consumers to detect:

- tampering,
- incomplete updates,
- drift between mirrors and authoritative registries.

Integrity is a prerequisite for entitlement validation and settlement correctness.

4.5 Minimal Operational Requirements

Registries MUST remain usable in environments with:

- simple file-based publication workflows,
- static hosting,
- minimal dependencies,
- offline or air-gapped audit systems.

No runtime service is required.
A file-based or static-registry approach MUST remain fully supported.

4.6 Federation and Delegation

The architecture MUST support:

- multiple authoritative registries (e.g., per league),
- mirror registries,

- aggregator registries.

Federation MUST NOT create ambiguity about which registry is authoritative for a given identifier.
Authority scope MUST be clearly declared.

4.7 Extensibility and Forward Compatibility

Registries MUST support the addition of:

- new object types,
- new fields,
- new indexes,
- new key-registry formats.

Extensibility MUST NOT break existing consumers.
Consumers MUST ignore unknown fields and object types unless forbidden by other USG RFCs.

4.8 Independence from Transport and Backend

The registry architecture MUST NOT assume:

- a specific database technology,
- a specific API protocol,
- a specific hosting model.

Registry content MUST be consumable through:

- direct file access,
- static hosting,
- content-addressed storage,
- minimal read-only APIs.

4.9 Alignment with Entitlement and Settlement Protocols

Registry rules MUST support the needs of:

- entitlement issuance and verification (RFC 0002),
- settlement and audit processes (future RFCs).

Registry semantics MUST ensure that rights, territories, and access windows can be enforced consistently across distributed systems.

4.10 Operational Transparency

Registries SHOULD include metadata that enables:

- inspection of version history,
- reasoning about object lifecycle changes,
- transparent understanding of authority and governance.

This supports regulatory trust, cross-platform interoperability, and public auditability.

5. Registry Object Model

A USG registry is composed of **registry objects**, each represented as a canonical JSON document.
This section defines the common structure, required fields, and normative semantics for all registry object types.

All objects MUST adhere to the canonical JSON and digest requirements defined in Section 6.

5.1 Core Object Structure

Every registry object MUST include the following top-level fields:

Field	Type	Description
`type`	string	The object's classification (e.g., `league`, `team`, `event`).
`id`	string	The object's canonical identifier (stable and immutable).
`version`	string	The object's version identifier.
`meta`	object	Metadata describing creation, updates, status, and integrity.

Objects MAY include additional fields that define their type-specific behavior.
Consumers MUST treat unknown fields as OPTIONAL and MUST NOT reject objects containing them unless otherwise specified by future RFCs.

5.2 Object Types

A conforming USG registry MUST represent the following object types:

- `league`
- `team`
- `venue`
- `broadcaster` (or `distributor`)
- `rights_bundle`
- `event`
- `key` (public-key registry object)

Registries MAY define additional object types for extended use cases but MUST document them and MUST NOT conflict with reserved types.

5.3 Object Identifier (OID) Requirements

The `id` field uniquely identifies the object within the registry's authority scope and the broader USG namespace.

Object identifiers:

- MUST be globally unique within the registry.
- MUST be stable once assigned.
- MUST NOT be reused for semantically distinct entities.
- SHOULD be treated as opaque by consumers.

- MAY include human-readable components (e.g., league code, team code, date).

Consumers MUST NOT infer semantics from identifier structure.

5.4 Version Field Requirements

The `version` field:

- MUST change whenever the canonical JSON encoding of the object changes.
- MUST follow a monotonically increasing ordering.
- SHOULD follow a semantic-versioning pattern (`MAJOR.MINOR.PATCH`), though registries MAY adopt alternative version schemes if well-documented.
- MUST be treated as an opaque string by consumers unless otherwise specified.

Only one version of an object MAY have the status `"active"` at any time.

5.5 Metadata (`meta`) Requirements

The `meta` field is REQUIRED for all registry objects.

It MUST contain:

Field	Type	Description
`created_at`	string	RFC 3339 timestamp of initial publication.
`last_updated_at`	string	RFC 3339 timestamp of the most recent update.
`status`	string	One of: `active`, `superseded`, `withdrawn`.

The following fields are OPTIONAL but RECOMMENDED:

Field	Type	Description
`digest`	object	SHA-256 digest metadata for integrity validation.
`supersedes`	object	Pointer to the prior object version (if applicable).
`withdrawal_reason`	string	Reason for withdrawal, if status is `withdrawn`.
`change_notes`	string	Human-readable summary of updates.

5.5.1 Digest Object Structure

If present, the `meta.digest` object MUST follow this structure:

```
{
  "alg": "sha-256",
  "value": "<hex-encoded-digest>"
}
```

5.6 Consistency Requirements

Registry operators MUST ensure the following consistency rules across all registry objects:

1. **Single Active Version**
   Only one version of a given object MAY have `"status": "active"` at any time.

2. **Preservation of Historical Objects**
   Superseded and withdrawn objects:

   - MUST remain available for audit and historical lookup;
   - MUST retain their original identifiers and version numbers;
   - MUST NOT be overwritten or deleted.

3. **Valid Object References**
   Inter-object references (e.g., `league_id`, `team_id`, `venue_id`, `rights_bundles`)

   - MUST resolve to existing registry objects;
   - MUST NOT reference withdrawn objects;
   - MUST reference `"active"` objects unless an operation explicitly requires historical context.

4. **Lifecycle Integrity**
   Objects MUST transition between lifecycle states (`active` → `superseded` → `withdrawn`) according to the rules in Section 7.

5. **Digest Consistency**
   The SHA-256 digest MUST correspond exactly to the canonical JSON encoding of the object.
   If the canonical encoding changes, the version **and** digest MUST change.

6. **Atomic Publication**
   Updates to objects, indexes, and digest files SHOULD be published atomically to avoid exposing mixed or partial registry states.

5.7 Example Registry Object (Event)

The following non-normative example illustrates a fully compliant `event` object:

```
{
  "type": "event",
  "id": "usg:event:nba:2025-12-25:LAL-BOS",
  "version": "1.0.0",
  "meta": {
    "created_at": "2025-12-01T10:00:00Z",
    "last_updated_at": "2025-12-01T10:00:00Z",
    "status": "active",
    "digest": {
      "alg": "sha-256",
      "value":
"a4f8c0c3c51f4b1a6f9c2f2e7d7a19a2c5f9473f6f2b0c1d8e9f0a1b2c3d4e5"
    }
```

```
  },
  "league_id": "usg:league:nba",
  "home_team_id": "usg:team:nba:LAL",
  "away_team_id": "usg:team:nba:BOS",
  "venue_id": "usg:venue:nba:CRYPTOCOM",
  "scheduled_start": "2025-12-25T20:00:00Z",
  "rights_bundles": ["usg:rb:nba:2025:INTL-OTT"],
  "labels": ["regular_season", "christmas_game"]
}
```

This example demonstrates:

- the REQUIRED fields (`type`, `id`, `version`, `meta.*`)
- canonical field ordering
- correct digest placement
- correct reference structure (`league_id`, `team_id`, etc.)

5.8 Invalid Object Examples

The following objects illustrate common violations of this specification and MUST be rejected by conforming validators.

5.8.1 Missing Required Metadata

```
{
  "type": "event",
  "id": "usg:event:nba:2025-12-25:LAL-BOS"
}
```

Violations:

- missing `version`
- missing `meta`
- missing required event fields (e.g., `league_id`, `scheduled_start`)
- invalid as a registry object

5.8.2 Reused Identifier for Different Semantics

```
{
  "type": "event",
  "id": "usg:event:nba:2025-12-25:LAL-BOS",
  "version": "1.0.0",
  "meta": {
    "created_at": "...",
    "last_updated_at": "...",
    "status": "active"
```

```
  },
  "league_id": "usg:league:wnba"
}
```

Violations:

- the same `id` is reused for conflicting semantics (NBA event vs. WNBA league)
- object identifiers MUST remain stable and MUST NOT change meaning

5.8.3 Reference to Withdrawn Object

```
{
  "type": "event",
  "id": "usg:event:nba:2025-12-30:LAL-NYK",
  "version": "1.0.0",
  "meta": {
    "created_at": "...",
    "last_updated_at": "...",
    "status": "active"
  },
  "league_id": "usg:league:nba",
  "venue_id": "usg:venue:nba:STAPLES",
  "rights_bundles": ["usg:rb:nba:2024:REVOKED"]
}
```

Violations:

- references a withdrawn rights bundle (`status: withdrawn`)
- withdrawn objects MUST NOT appear in live objects or live indexes
- invalid for publication

5.8.4 Digest Mismatch

```
{
  "type": "team",
  "id": "usg:team:nba:LAL",
  "version": "1.0.0",
  "meta": {
    "created_at": "...",
    "last_updated_at": "...",
    "status": "active",
    "digest": {
      "alg": "sha-256",
      "value":
"ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff"
    }
  },
```

```
  "name": "Los Angeles Lakers"
}
```

Violations:

- the published digest does not match the canonical JSON byte representation
- canonical JSON + SHA-256 digest mismatch indicates tampering or invalid publication
- MUST be rejected by validators

---

6. Canonical JSON and Hashing

A USG registry relies on deterministic, machine-verifiable object representations.
Canonical JSON ensures that identical logical objects always produce identical byte sequences, enabling integrity verification through SHA-256 digests.

6.1 Canonical JSON Encoding

All registry objects and index files MUST be encoded using **canonical JSON**.
At minimum, canonical JSON in a USG registry MUST satisfy the following:

1. **UTF-8 Encoding**
   All documents MUST be encoded as UTF-8.

2. **Deterministic Key Ordering**
   Keys in JSON objects MUST be sorted lexicographically by Unicode code point.

3. **Minimal Whitespace**
   Whitespace MUST be limited to that required by JSON syntax:

   - no trailing spaces;
   - no tab (`\t`) characters;
   - no extra spaces around `:` or `,`.

4. **Stable Number Encoding**
   Numbers MUST NOT include leading zeros or alternative numeric forms.

5. **String Normalization**
   String values SHOULD be normalized to NFC (Normalization Form C).
   Deviations MUST be documented if used.

Canonicalization MUST occur prior to computing digests and prior to publication.

6.2 SHA-256 Digest Generation

A SHA-256 digest MUST be computed over the **canonical JSON byte representation** of each:

- registry object, and
- registry index file.

Digest values:

- MUST use the SHA-256 algorithm;
- MUST be lower-case hex;
- MUST remain stable unless the canonical JSON changes.

Digest metadata MAY appear inside the object's `meta` section but MUST also be included in the registry's digest indexes (Section 8.3).

Example digest block:

```
{
  "alg": "sha-256",
  "value":
"a4f8c0c3c51f4b1a6f9c2f2e7d7a19a2c5f9473f6f2b0c1d8e9f0a1b2c3d4e5"
}
```

6.3 Digest Validation

Registry consumers and validators:

- MUST compute SHA-256 over the canonical JSON form of each object or index;
- MUST compare the computed digest against:
  - the object's optional `meta.digest.value`, and
  - the digest index entry for that object.

If digest values differ:

- the object MUST be treated as **invalid** for entitlement validation or settlement;
- validators SHOULD produce a machine-readable error describing the mismatch;
- consumers MAY fall back to a prior known-good snapshot.

Digest mismatches MUST be treated as integrity failures, not versioning differences.

6.4 Digest Stability and Change Detection

Any change to a registry object's canonical JSON MUST cause the digest to change.

Registry operators SHOULD:

- treat digest changes as the authoritative signal of object modification;
- regenerate digests and indexes in a single atomic build step;
- ensure that objects and digest indexes are never published out of sync.

Registry consumers SHOULD:

- use digest indexes to determine whether local caches are current;
- treat missing or mismatched digests as indicators of possible corruption, partial updates, or tampering.

6.5 Example of Canonical JSON and Digest Alignment

```
{
  "type": "team",
  "id": "usg:team:nba:BOS",
  "version": "1.0.0",
  "meta": {
    "created_at": "2025-12-01T12:00:00Z",
    "last_updated_at": "2025-12-01T12:00:00Z",
    "status": "active",
    "digest": {
      "alg": "sha-256",
      "value":
"53de98f3c21319c3fbf33b1b8df5cd765d05b0ac28930f1d8817e2d968f5b322"
    }
  },
  "name": "Boston Celtics"
}
```

The digest MUST match the canonical JSON encoding (after key sorting, whitespace normalization, and UTF-8 serialization). Any modification to the object — including changes to timestamps, status, or formatting — MUST produce a different digest and therefore a new "version" value.

7. Versioning, Supersession, and Withdrawal

USG registries MUST support deterministic lifecycle transitions for all objects.
This section defines the required semantics for version changes, supersession, and withdrawal.

7.1 Versioning Model

Every registry object MUST include a `"version"` field.

The version:

- MUST change whenever the canonical JSON representation changes;
- MUST follow a monotonically increasing ordering;
- SHOULD use semantic-versioning (`MAJOR.MINOR.PATCH`), though the exact scheme MAY vary;
- MUST be treated as an opaque string by consumers;
- MUST NOT be reused for different object states.

Only one version of an object MAY have `"status": "active"` at any time.

7.2 Supersession Semantics

Supersession occurs when a new version of an object **replaces** a previous version.

A superseding object:

- MUST set `"meta.status"` to `"active"` unless a more specific status applies;
- MUST include a `"supersedes"` object referencing the prior object's `id` and `version`;

- MUST increment the `"version"` value;
- MUST NOT alter the meaning of the identifier (`id`).

The superseded object:

- MUST update `"meta.status"` to `"superseded"`;
- MUST remain available for audit and historical lookup;
- MUST NOT be used for new entitlements or settlements after the supersession takes effect.

Example (Superseding an Event):

```
{
  "type": "event",
  "id": "usg:event:nba:2025-12-25:LAL-BOS",
  "version": "1.1.0",
  "meta": {
    "created_at": "2025-12-01T10:00:00Z",
    "last_updated_at": "2025-12-10T09:00:00Z",
    "status": "active",
    "supersedes": {
      "id": "usg:event:nba:2025-12-25:LAL-BOS",
      "version": "1.0.0"
    }
  },
  "league_id": "usg:league:nba",
  "home_team_id": "usg:team:nba:LAL",
  "away_team_id": "usg:team:nba:BOS",
  "scheduled_start": "2025-12-25T20:30:00Z",
  "venue_id": "usg:venue:nba:CRYPTOCOM",
  "rights_bundles": ["usg:rb:nba:2025:INTL-OTT"]
}
```

7.3 Withdrawal Semantics

Withdrawal is a terminal state used when an object is no longer valid.

An object MUST be marked `"withdrawn"` when:

- an event is cancelled;
- a rights bundle is invalidated;
- a team, venue, or broadcaster record becomes factually incorrect;
- a key is compromised or retired;
- any integrity or governance issue requires explicit deactivation.

A withdrawn object:

- MUST set `"meta.status": "withdrawn"`;
- SHOULD include `"withdrawal_reason"` with a registry-defined code or explanation;
- MUST remain retrievable for historical and audit purposes;
- MUST NOT be referenced by newly created objects or included in live indexes.

Withdrawal does NOT require a superseding object.

Example (Withdrawn Event):

```
{
  "type": "event",
  "id": "usg:event:nba:2025-12-25:LAL-BOS",
  "version": "1.2.0",
  "meta": {
    "created_at": "2025-12-01T10:00:00Z",
    "last_updated_at": "2025-12-15T08:00:00Z",
    "status": "withdrawn",
    "withdrawal_reason": "cancelled_event"
  },
  "league_id": "usg:league:nba"
}
```

7.4 Change-Control Metadata

Every registry object MUST include lifecycle metadata inside `"meta"`:

- `"created_at"` — RFC 3339 timestamp of initial creation;
- `"last_updated_at"` — RFC 3339 timestamp of most recent modification;
- `"status"` — one of: `active`, `superseded`, `withdrawn`;
- `"supersedes"` — OPTIONAL pointer to the previous object version;
- `"withdrawal_reason"` — OPTIONAL explanation for withdrawn objects;
- `"change_notes"` — OPTIONAL free-text description of the update.

Change metadata MUST be updated whenever an object transitions to a new version.

7.5 Lifecycle Consistency Requirements

Registry operators MUST ensure:

- no two versions of the same object have `"status": "active"` simultaneously;
- superseded and withdrawn versions remain available for audit and historical use;
- `"active"` objects MUST NOT reference `"withdrawn"` objects;
- registry indexes MUST NOT include withdrawn objects;
- digests and version numbers MUST correctly reflect lifecycle state.

Registry consumers MUST:

- treat `"superseded"` and `"withdrawn"` objects as invalid for forward-looking use;
- retain historical versions when validating past entitlements or settlements;
- treat lifecycle inconsistencies as protocol errors.

---

8. Registry Layout and Indexing

A USG registry MUST provide a predictable and verifiable directory structure, along with index files and digest manifests that enable efficient discovery and integrity validation. This section defines the REQUIRED layout and indexing expectations for all conforming registries.

8.1 Directory Layout (File-Based Registries)

USG registries MAY be hosted as static files, object storage trees, content-addressed collections, or similar structures.
Regardless of hosting method, the logical directory layout MUST follow the structure defined here.

A RECOMMENDED layout is:

```
/registry
/leagues
  {league_id}.json
/teams
  {team_id}.json
/venues
  {venue_id}.json
/broadcasters
  {broadcaster_id}.json
/rights-bundles
  {rights_bundle_id}.json
/events
  {event_id}.json
/keys
  {key_id}.json
/indexes
leagues.json
events.json
events-by-date.json
events-by-league.json
/digests
objects.json
indexes.json
```

Registry operators MAY deviate from this layout but MUST:

- publish documentation describing the chosen structure;
- ensure equivalent index coverage;
- ensure all files adhere to canonical JSON and digest rules.

8.2 Index Requirements

Registry indexes provide searchable, structured views over registry objects.
All indexes MUST be valid registry objects encoded as canonical JSON.

A conforming registry MUST provide at least the following indexes:

1. **All Leagues Index**
   Lists all `league` objects.

2. **All Events Index**
   Lists all `event` objects.

3. **Events-by-Date Index**
   Enables chronological resolution of events.

4. **Events-by-League Index**
   Enables league-scoped discovery.

Indexes MUST NOT include withdrawn objects and SHOULD NOT include superseded objects.

8.2.1 Example: `events-by-date` Index

```
{
  "type": "index:events-by-date",
  "version": "1.0.0",
  "meta": {
    "generated_at": "2025-12-01T12:00:00Z",
    "status": "active"
  },
  "items": [
    {
      "event_id": "usg:event:nba:2025-12-25:LAL-BOS",
      "scheduled_start": "2025-12-25T20:00:00Z",
      "league_id": "usg:league:nba"
    }
  ]
}
```

8.3 Digest Indexes

Digest indexes provide registry-wide integrity guarantees by mapping registry artifacts to SHA-256 digest values.

A conforming registry MUST provide:

- an **objects digest index**, and
- an **indexes digest index**.

Digest indexes MUST be encoded as canonical JSON and MUST include:

- `type` (e.g., `"digest:objects"` or `"digest:indexes"`);
- `version`;
- `algorithm` (MUST be `"sha-256"`);
- a `meta` object containing a `generated_at` timestamp;
- an `items` array mapping object identifiers or paths to digest values.

8.3.1 Example: Objects Digest Index

```
{
  "type": "digest:objects",
  "version": "1.0.0",
  "algorithm": "sha-256",
  "meta": {
    "generated_at": "2025-12-01T12:00:00Z",
    "status": "active"
  },
  "items": [
    {
      "path": "events/usg:event:nba:2025-12-25:LAL-BOS.json",
      "digest":
"a4f8c0c3c51f4b1a6f9c2f2e7d7a19a2c5f9473f6f2b0c1d8e9f0a1b2c3d4e5"
    }
  ]
}
```

Digest indexes MUST be regenerated atomically with object and index publication (see Section 6.4).

8.4 Index Naming Conventions

Registry indexes SHOULD follow predictable naming conventions to ensure interoperability across implementations:

- `{object_type}s.json` — index of all objects of a given type
- `{object_type}s-by-{attribute}.json` — attribute-scoped index
- `events-by-league.json` — league-scoped event discovery
- `events-by-date.json` — chronological event discovery

Registry operators MAY define additional indexes but MUST NOT alter the semantics of REQUIRED indexes defined in this specification.

8.5 Required Index Fields

Index entries MUST include the minimum fields necessary to resolve authoritative objects.

For event-based indexes, entries MUST include:

Field	Description
`event_id`	Identifier of the referenced event object.
`league_id`	Identifier of the associated league.
`scheduled_start`	Event start time (RFC 3339).

Indexes covering heterogeneous object types SHOULD include an explicit `type` field.

Additional fields MAY be included for convenience or performance, but MUST be redundant with authoritative source objects.

8.6 Index Lifecycle Management

Indexes MUST:

- be regenerated whenever referenced objects change;
- include only objects with `"status": "active"`;
- exclude withdrawn objects;
- exclude superseded objects unless explicitly designated as historical;
- include a `version` field;
- include a `meta.generated_at` timestamp.

Consumers MUST treat indexes with digest mismatches or missing digest entries as invalid.

8.7 Path Structure and File Naming

Registry object filenames SHOULD follow this pattern:

```
{type}/{id}.json
```

Examples:

- `leagues/usg:league:nba.json`
- `teams/usg:team:nba:LAL.json`
- `events/usg:event:nba:2025-12-25:LAL-BOS.json`

If hosting or filesystem constraints prevent direct use of identifiers, registry operators MUST define a deterministic and reversible mapping and document it publicly.

8.8 Canonical vs. Non-Canonical Fields in Indexes

Index files are **derivative artifacts** and MUST NOT define authoritative state.

Accordingly:

- indexes MUST reference canonical object identifiers;
- indexes MUST NOT introduce fields that conflict with authoritative objects;
- indexes MAY include redundant fields for query efficiency;
- canonical truth always resides in the referenced registry objects.

Consumers MUST resolve authoritative data by dereferencing the referenced object, not by trusting index content alone.

9. Federation and Authority

USG registries are designed to operate in a federated environment where multiple independent registry operators may publish data under distinct authority scopes. This section defines how authority is declared, how federation operates, and how consumers determine which registry is authoritative for a given object.

9.1 Authority Declaration

Each registry MUST explicitly declare its authority information in top-level registry metadata. At minimum, this declaration MUST include:

- the identity of the registry operator;
- the authority scope covered by the registry (e.g., specific leagues or competitions);
- a reference to governance or policy documentation, if applicable.

Authority scope determines which registry is considered canonical for a given object identifier. For any identifier, at most one registry SHOULD be considered authoritative.

9.2 Authoritative Registries

An authoritative registry is the canonical source of truth for objects within its declared authority scope.

Authoritative registries:

- MAY create, update, supersede, and withdraw objects within scope;
- MUST publish canonical objects, indexes, and digest manifests;
- MUST ensure lifecycle and integrity rules defined in this specification are followed;
- SHOULD provide stable publication endpoints or file distributions.

Consumers MUST treat data from authoritative registries as canonical for the identifiers within their scope.

9.3 Mirror Registries

A mirror registry is a read-only replica of an authoritative registry.

Mirror registries:

- MUST NOT create, modify, supersede, or withdraw authoritative objects;
- MUST synchronize content using digest validation or equivalent integrity checks;
- MUST declare the upstream authoritative registry and last synchronization time;
- MUST NOT alter canonical object identifiers or semantics.

Mirror registries MAY provide improved availability, geographic distribution, or caching, but MUST preserve the integrity and authority of upstream data.

9.4 Aggregator Registries

Aggregator registries collect objects from multiple authoritative registries to provide cross-scope discovery and unified indexing.

Aggregator registries:

- MAY publish derived indexes that reference authoritative objects;
- MUST preserve original object identifiers and versioning;

- MUST NOT redefine or override authoritative semantics;
- MUST clearly identify which authoritative registry each object originates from.

Aggregators MAY generate synthetic identifiers for internal use, but MUST maintain a clear mapping to authoritative identifiers.

9.5 Conflict Resolution

If multiple registries claim authority over the same identifier:

- consumers MUST treat the situation as a protocol error;
- consumers SHOULD prefer registries whose authority scope explicitly covers the identifier namespace;
- governance or policy processes outside the scope of this RFC SHOULD be used to resolve conflicts.

This specification does not define automatic conflict resolution mechanisms.

9.6 Federation Discovery

Registries MAY publish discovery metadata to support federation, including:

- lists of upstream authoritative registries;
- trust anchors or key registry references;
- supported index sets.

Discovery mechanisms MAY be implemented via static metadata files or read-only APIs. A future RFC MAY define a standardized federation discovery format.

9.7 Trust Boundaries

Each registry represents a distinct trust boundary.

Consumers:

- MUST validate digests and lifecycle metadata for all registry content;
- MUST NOT assume trust based solely on hosting location or transport security;
- SHOULD scope trust decisions to declared authority boundaries.

Federation MUST NOT weaken integrity guarantees defined elsewhere in this specification.

---

10. Key Registries and Trust Anchors

USG registries MUST support publication of public-key material used to validate entitlement tokens, registry digests, and other signed artifacts defined in RFC 0002. This section defines the structure and lifecycle of **key registry objects** and the concept of **trust anchors**.

10.1 Key Registry Objects

A USG registry SHOULD include a **Key Registry**, implemented as a collection of registry objects of type `key`.

Each key object MUST represent a single cryptographic public key and its associated metadata.

A key registry object MUST include:

- `type` — MUST be `"key"`
- `id` — canonical identifier for the key
- `version` — key object version
- `meta` — lifecycle metadata (Section 7)
- `issuer_id` or `operator_id` — identifier of the entity that controls the key
- `public_key` — public key material (e.g., JWK or equivalent)
- `alg` — cryptographic algorithm identifier

Key objects MAY include:

- `kid` — key identifier used in token headers
- `valid_from` — RFC 3339 timestamp
- `valid_until` — RFC 3339 timestamp
- `usage` — intended usage (e.g., `entitlement-signing`, `registry-signing`)

10.2 Example Key Registry Object

```
{
  "type": "key",
  "id": "usg:key:nba:issuer:2025-01",
  "version": "1.0.0",
  "meta": {
    "created_at": "2025-01-01T00:00:00Z",
    "last_updated_at": "2025-01-01T00:00:00Z",
    "status": "active"
  },
  "issuer_id": "usg:league:nba",
  "kid": "nba-issuer-2025-01",
  "alg": "ES256",
  "usage": "entitlement-signing",
  "public_key": {
    "kty": "EC",
    "crv": "P-256",
    "x": "...",
    "y": "..."
  }
}
```

10.3 Key Status and Lifecycle

Key registry objects MUST follow the lifecycle rules defined in Section 7.

Key status values:

- `"active"` — the key MAY be used for signing or verification;

- `"superseded"` — the key has been replaced by a newer key and MUST NOT be used for new signing operations;
- `"withdrawn"` — the key MUST NOT be used for any purpose.

Key rotation MUST be represented via **supersession**, not deletion.

Revoked or compromised keys MUST be marked `"withdrawn"` and SHOULD include a `withdrawal_reason` indicating the cause of revocation.

Superseded and withdrawn keys MUST remain retrievable for audit and historical validation.

10.4 Key Resolution During Validation

During entitlement validation (as defined in RFC 0002), verifiers:

- MUST resolve the issuer or operator identifier contained in the token;
- MUST retrieve the corresponding key object from the authoritative registry;
- MUST verify that the resolved key has `"status": "active"`;
- MUST reject tokens signed by superseded or withdrawn keys.

When `valid_from` and `valid_until` fields are present, verifiers SHOULD ensure the current validation time falls within the declared validity window.

Failure to resolve a valid, active key MUST result in token rejection.

10.5 Trust Anchors

A **trust anchor** defines the root of trust for a registry or registry federation.

Each registry SHOULD publish trust anchor metadata identifying:

- the entity controlling the registry;
- the authoritative key registry used for validation;
- the keys used to sign registry digests or API responses, if applicable.

Trust anchor metadata MAY be represented as:

- a dedicated registry object; or
- a top-level metadata document referenced by the registry.

10.6 Trust Boundary Enforcement

Consumers:

- MUST treat each registry as a distinct trust boundary;
- MUST NOT trust keys or registry artifacts outside the declared authority scope;
- MUST validate key registry objects using the same canonical JSON, digest, and lifecycle rules applied to other registry objects.

Trust anchors MUST NOT bypass digest validation or lifecycle enforcement requirements.

10.7 Relationship to RFC 0002

Key registries defined in this section provide the discovery and validation substrate required by RFC 0002.

Specifically:

- entitlement token issuers referenced in tokens MUST resolve to active key registry objects;
- key rotation and revocation MUST be represented through registry lifecycle semantics;
- verification logic MUST rely on registry-published keys rather than hard-coded trust material.

11. Validation Requirements

Implementations that consume or process USG registries MUST perform validation to ensure integrity, consistency, and compliance with this specification. Validation MAY be performed at ingest time, build time, or during runtime resolution.

11.1 Required Validation Checks

A conforming validator MUST perform the following checks:

1. **Canonical JSON Compliance**
    - All registry objects and index files MUST conform to canonical JSON requirements defined in Section 6.
    - Non-canonical encodings MUST be rejected.
2. **Digest Verification**
    - SHA-256 digests MUST be computed over canonical JSON.
    - Computed digests MUST match values published in digest indexes.
    - Objects or indexes with missing or mismatched digests MUST be treated as invalid.
3. **Identifier Resolution**
    - All referenced identifiers (e.g., `league_id`, `team_id`, `venue_id`, `rights_bundles`, `issuer_id`) MUST resolve to existing registry objects.
    - References to withdrawn objects MUST be rejected.
4. **Lifecycle Consistency**
    - At most one version of a given object MAY have `"status": "active"`.
    - Superseded and withdrawn objects MUST follow lifecycle rules defined in Section 7.
    - Active objects MUST NOT reference withdrawn objects.
5. **Index Integrity**
    - Index entries MUST reference existing, active registry objects.
    - Indexes MUST exclude withdrawn objects.
    - Index digests MUST match canonical JSON representations.
6. **Key Validation**

- Key registry objects MUST be validated according to Section 10.
- Entitlement tokens MUST be rejected if key resolution fails or if keys are not active.

11.2 Validation Failure Handling

When validation failures occur, implementations:

- MUST treat affected objects or indexes as invalid;
- SHOULD emit machine-readable error reports describing the failure;
- MAY fall back to a previous known-good registry snapshot;
- MUST NOT silently ignore validation errors.

Validation failures involving integrity or lifecycle violations SHOULD be treated as security-relevant events.

11.3 Snapshot Validation

Validators SHOULD support validation of complete registry snapshots.

A snapshot is valid if:

- all objects and indexes pass required validation checks;
- digest indexes match computed digests;
- object and index versions form a consistent lifecycle state.

Snapshots allow deterministic entitlement validation and historical reconstruction.

11.4 Continuous Validation

Registry operators SHOULD implement continuous validation within build and deployment pipelines to ensure that:

- canonical JSON encoding is enforced prior to publication;
- digests and indexes are regenerated atomically;
- lifecycle transitions are correctly applied;
- invalid registry states are never published.

11.5 Validation Output

Validation tools SHOULD produce outputs suitable for:

- automated CI pipelines;
- audit and compliance review;
- operational monitoring and alerting.

Outputs MAY include structured logs, JSON reports, or signed attestations.

---

12. Security Considerations

USG registries form a critical trust surface for entitlement validation, rights enforcement, and settlement. Compromise or misuse of registry data can result in unauthorized access, revenue leakage, or audit failure.

This section outlines key security considerations for registry operators and consumers.

12.1 Registry Integrity

Registry consumers MUST NOT trust registry content unless:

- canonical JSON validation succeeds;
- SHA-256 digests match published digest indexes;
- lifecycle rules are satisfied.

Failure to validate integrity MUST result in rejection of affected objects or snapshots.

12.2 Partial Update and Rollback Attacks

Attackers may attempt to induce inconsistent registry states by exposing consumers to partial updates (e.g., updated objects without updated digest indexes).

Registry operators SHOULD:

- publish objects, indexes, and digest files atomically;
- version and timestamp all registry artifacts;
- prevent public access to intermediate build states.

Consumers SHOULD detect and reject snapshots with mismatched digests or incomplete index coverage.

12.3 Key Compromise and Rotation

Compromise of registry signing keys or entitlement-signing keys can undermine trust in the entire system.

Registry operators MUST:

- represent key rotation and revocation using lifecycle semantics defined in Section 10;
- mark compromised keys as `"withdrawn"` immediately;
- publish updated key registry objects promptly.

Consumers MUST reject tokens or registry artifacts validated using withdrawn keys.

12.4 Authority Spoofing

Malicious actors may attempt to publish fraudulent registries claiming authority over identifiers they do not control.

Consumers MUST:

- verify registry authority scope declarations;
- ensure that object identifiers fall within the declared authority scope;
- treat conflicting authority claims as protocol errors.

12.5 Mirror and Aggregator Risks

Mirrors and aggregators increase availability but introduce additional trust boundaries.

Operators of mirror and aggregator registries MUST:

- preserve canonical identifiers and semantics;
- clearly identify upstream authoritative registries;
- prevent unauthorized modification of authoritative content.

Consumers SHOULD prefer authoritative registries when resolving conflicts.

12.6 Denial of Service and Resource Exhaustion

Large or malformed registries may be used to exhaust validator resources.

Implementations SHOULD:

- enforce reasonable size limits;
- validate incrementally using digest indexes;
- apply timeouts and rate limits for API-based registries.

12.7 Audit and Logging

Registry operators SHOULD maintain audit logs of:

- registry publication events;
- lifecycle transitions;
- key rotations and withdrawals.

Audit logs SHOULD be protected against tampering and retained according to operational and regulatory requirements.

---

13. Privacy Considerations

USG registries are designed to publish **institutional and event-level metadata**, not end-user or viewer-level information. This specification intentionally limits the scope of registry content to minimize privacy risk.

13.1 Absence of Personal Data

Registry objects MUST NOT include:

- end-user identifiers;
- viewer or subscriber information;
- device identifiers;
- personally identifiable information (PII).

Any registry that includes such data is **non-conforming** with this specification.

13.2 Metadata Sensitivity

While registry content is primarily institutional, certain metadata (e.g., event timing, distribution arrangements, or key material identifiers) may be operationally sensitive.

Registry operators SHOULD:

- publish only metadata necessary for interoperability and validation;
- avoid embedding confidential business logic or contractual terms;
- treat registry content as public or semi-public by default.

13.3 Key Material Privacy

Key registry objects publish public keys and associated metadata.

Registry operators MUST ensure:

- only public key material is included;
- private keys are never published;
- key identifiers and metadata do not encode sensitive internal information.

13.4 Aggregation and Correlation Risk

Federated or aggregated registries may increase the risk of inference through correlation.

Operators of aggregator registries SHOULD:

- document the sources and scope of aggregated data;
- avoid introducing new linkages that could expose sensitive relationships;
- ensure aggregation does not weaken the privacy posture of authoritative registries.

13.5 Compliance Considerations

Because USG registries do not process personal data, they are generally out of scope for privacy regulations such as GDPR and CCPA.

However, registry operators remain responsible for compliance with applicable laws and policies governing:

- publication of institutional data;
- cryptographic material;
- public infrastructure metadata.

---

14. IANA Considerations

This specification does not define any namespaces, protocol parameters, media types, or registries that require registration with the Internet Assigned Numbers Authority (IANA).

Future USG specifications MAY define protocol-level namespaces or identifier governance models that warrant external registration or coordination. Any such requirements will be addressed in the corresponding documents.

---

15. References

15.1 Normative References

- **RFC 2119**
  Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels”,

BCP 14, RFC 2119, DOI 10.17487/RFC2119.

- **RFC 3339**
  Klyne, G. and C. Newman, “Date and Time on the Internet: Timestamps”,
  RFC 3339, DOI 10.17487/RFC3339.

- **RFC 0001 — The Universal Sports Graph**
  Jellen, S., “The Universal Sports Graph: A Specification for Rights, Reach, and Real-Time Access”,
  USG-RFC 0001, DOI 10.5281/zenodo.17565793.

- **RFC 0002 — USG Entitlement Token Profile**
  Jellen, S., “USG Entitlement Token Profile: A Standards-Track Specification for Tokenized Sports Access”,
  USG-RFC 0002, DOI 10.5281/zenodo.17781619.

15.2 Informative References

- **JSON**
  ECMA-404, “The JSON Data Interchange Syntax”.

- **SHA-256**
  National Institute of Standards and Technology (NIST),
  “Secure Hash Standard (SHS)”, FIPS PUB 180-4.

- **JWK (JSON Web Key)**
  Jones, M., “JSON Web Key (JWK)”, RFC 7517, DOI 10.17487/RFC7517.

---

16. Revision History

- **v1.0.0 (Draft)** — Initial release of RFC 0003 defining the USG Registry Architecture, including registry object models, canonical JSON requirements, digest and integrity rules, indexing and directory layout, federation and authority semantics, key registries, and validation requirements.