Universal Manifest v0.1 Specification

Initial Draft

15 February 2026

This version: https://universalmanifest.net/spec/v0.1/

Latest published version: https://universalmanifest.net/spec/v0.3/

Next version: https://universalmanifest.net/spec/v0.2/

History: Changelog

Editors: Universal Manifest Working Group

Abstract

This specification defines a JSON-LD-based file format known as the **Universal Manifest**, a portable state capsule. Formulated as a hybrid synthesis of web publication metadata and web application parameters, this format provides developers and issuers with a standardized envelope to convey linked-data identity references, role permissions, device registrations, and pointers to canonical data sources.

The Universal Manifest is specifically designed for local-first environments (e.g., venue edges, public displays) where consumers must tolerate partial connectivity and rely on cached, verifiable state. Using this standard, user agents, smart displays, and network edges can securely interoperate without requiring a continuous cloud connection, facilitating seamless cross-context experiences.

Status of This Document

This section describes the status of this document at the time of its publication.

Universal Manifest v0.1 is the initial published version of the Universal Manifest specification. It defines the core JSON-LD envelope, lifecycle semantics, and caching model. The v0.1 format establishes the manifest structure with intentionally permissive signature semantics, allowing early adoption while a normative signature profile is developed.

Because Universal Manifest is in the v0.x line, minor-version bumps may include breaking changes when reflected by a new version folder. This version is **superseded by v0.2**, which introduces a normative signature profile (JCS + Ed25519), identity binding framework, and tiered trust model.

This document uses layout formats established by major web standard groups to present normative requirements cleanly.

For implementers: v0.1 adoption is discouraged for new projects. Please use the latest specification. If upgrading from v0.1, see the Migration Guide.

1. Universal Manifest

A **Universal Manifest** is a JSON-LD document acting as a cross-platform data envelope. It blends the semantic linkability of a Web Publication Manifest with the applied processing lifecycle of a Web App Manifest.

1.1. Examples

The following is an example of a minimal Universal Manifest:

Example 1: Minimal Universal Manifest payload

```
{
  "@context": [
    "https://universalmanifest.net/ns/v0.1"
  ],
  "@id": "urn:uuid:123e4567-e89b-12d3-a456-426614174000",
  "@type": ["um:Manifest"],
  "manifestVersion": "0.1",
  "subject": "did:example:123",
  "issuedAt": "2026-03-01T00:00:00Z",
  "expiresAt": "2026-03-02T00:00:00Z"
}
```

1.2. JSON-LD Core Members

1.2.1. `@context` member

The `@context` member establishes the semantic definitions of terms used within the manifest. It **MUST** include the Universal Manifest namespace for the specification version being implemented (e.g., `https://universalmanifest.net/ns/v0.1` for this version).

1.2.2. `@id` member

The `@id` member provides a primary identifier. Issuers **SHOULD** generate `@id` as an opaque, offline-safe identifier (e.g., `urn:uuid:<uuidv4>`).

1.2.3. `@type` member

The `@type` member indicates the document type classifying the resource. It **MUST** include the value `um:Manifest`.

1.3. Identities & Lifespans

1.3.1. `subject` member

The `subject` member specifies the primary entity (user, app, venue) the manifest describes. It **MUST** contain a stable identifier URI (e.g., a Decentralized Identifier / DID).

1.3.2. `issuedAt` and `expiresAt` members

The `issuedAt` and `expiresAt` members formulate the bounding constraints (TTL) for the manifest's validity. Both parameters are formatted as ISO 8601 / RFC 3339 date-time strings.

1.4. Structural State Members

The manifest structure relies on domain-specific members akin to Web Publication linkages.

1.4.1. `facets` member

The `facets` member organizes extended functional blocks (`um:Facet`), packaging specific verifiable capabilities, metadata subsets, or configuration modules.

1.4.2. `claims`, `consents`, `devices`, and `pointers`

These arrays group specific operational contexts representing permissions, deployed hardware targets, and external data reference pointers connected to the `subject`.

1.5. Signature Integrity

1.5.1. `signature` member

The `signature` member encapsulates cryptographic proofs of the manifest payload. In v0.1, its format is intentionally permissive. No canonicalization scheme, signature algorithm, or verification profile is defined.

> **Note:** A normative signature profile is introduced in v0.2. Implementers requiring tamper protection **MUST** migrate to v0.2 or later.

2. Entities and Facets

2.1. um:Facet Module

Reusable composable sub-documents that can be attached to a manifest. Common fields:

- `@type` — **MUST** include `um:Facet`.
- `name` — human-readable label (optional).
- `ref` — URI to canonical source (optional).
- `entity` — embedded or referenced entity (`um:Entity` or JSON-LD node; optional).

2.2. um:Entity Base

Base class for embedded entities. Common fields:

- `@id` — URI.
- `@type` — entity type(s).

3. Manifest Lifecycle and Caching

3.1. ID & Caching Guidance

v0.1 recommendation: Issuers generate `@id` as `urn:uuid:<uuidv4>` (opaque, globally unique, offline-safe).

Consumers cache the full manifest only while it is actively needed for rendering/interaction. Logs should store only the manifest `@id` (and optionally a content hash) to keep telemetry small and enable future recovery workflows.

4. Conformance

There are two primary implementation targets: **Consumer** (receives and processes manifests) and **Issuer** (produces manifests for a subject).

4.1. Consumer Behavior

A consumer **MUST** treat a value as a v0.1 Universal Manifest when it is a JSON object containing all required fields and `@type` includes `um:Manifest`.

A consumer **MUST** ignore unknown fields safely (unknown top-level properties, unknown facet entity shapes, unknown array item shapes).

Consumers **MUST** enforce TTL: if `now > expiresAt`, the manifest **MUST NOT** be used to grant permissions or render state.

4.2. Issuer Behavior

Issuers MUST set `@id` to a globally unique URI. Issuers MUST set `subject` to a stable identifier URI. Issuers SHOULD use short TTLs for safety.

4.3. Standalone Conformance Suite

To claim conformance, an implementation MUST accept all valid fixtures and reject all invalid fixtures. The suite is language-neutral and packages fixtures plus expected outcomes.

5. Security Considerations

v0.1 is a draft specification with intentionally permissive signature semantics. It MUST NOT be used in production security-critical contexts.

5.1. Signature Limitations

The v0.1 `signature` field is intentionally permissive and not interoperable across implementations. There is no defined canonicalization scheme, signature algorithm, or verification profile. This means:

- Manifests cannot be reliably verified for integrity or authenticity.
- Tampering attacks are not cryptographically prevented.
- Cross-implementation signature verification is not guaranteed.

For production use cases requiring tamper protection, implementers MUST migrate to v0.2.

5.2. TTL Enforcement

All manifest consumers MUST enforce time-to-live (TTL) validity:

- **REQUIRED:** Reject manifests where `now > expiresAt`.
- **RECOMMENDED:** Validate `issuedAt <= expiresAt` to detect malformed manifests.

TTL enforcement provides a basic defense against replay attacks (presenting expired manifests).

5.3. Resource Limits

To prevent denial-of-service attacks, consumers SHOULD enforce limits on manifest size and structure:

- Maximum total JSON size: 1 MB
- Maximum JSON nesting depth: 10 levels
- Maximum array sizes: 1,000 elements per array

6. Privacy Considerations

- **Opaque identifiers:** Issuers SHOULD use `urn:uuid:<uuidv4>` for `@id` to prevent correlation across contexts. Rotate `@id` per issuance.
- **Minimal disclosure:** Include only claims/consents necessary for the immediate use case.
- **Subject privacy:** Use pseudonymous or pairwise DIDs in `subject` field to prevent tracking.

7. References

7.1. Normative References

[JSON-LD]
Manu Sporny; Gregg Kellogg; Markus Lanthaler. JSON-LD 1.1. W3C.

[RFC3339]
G. Klyne; C. Newman. Date and Time on the Internet: Timestamps. IETF.

[RFC2119]
S. Bradner. Key words for use in RFCs to Indicate Requirement Levels. IETF.

7.2. Informative References

[DID-CORE]
Manu Sporny; Dave Longley; Markus Sabadello; Drummond Reed; Orie Steele; Christopher Allen. Decentralized Identifiers (DIDs) v1.0. W3C.

[WEB-APP-MANIFEST]
Marcos Caceres; Kenneth Christiansen; Mounir Lamouri; Anssi Kostiainen; Matt Giuca. Web Application Manifest. W3C.

[WEB-PUBLICATION]

Matt Garrish; Ivan Herman. Publication Manifest. W3C.