NQUIST TECHNICAL WHITE PAPER
NQ-TR-001
Version 1.0e - 2026

Digital Audio Assets Protocol (DAAP)

A Systems-Level Framework for Verifiable Identity, Provenance, and Lineage in Distributed Music Assets

Vanessa N. Adams
Graduate:
Musicians Institute Audio Engineering Certification
California Institute of Applied Technology Computer Information Systems Networking Concentration

https://www.nquist.net

Abstract

Contemporary digital music distribution continues to treat rendered audio files (e.g., WAV, AIFF, FLAC, AAC, Ogg Vorbis) as the primary transferable representation of recorded works [1][2]. These containers preserve acoustic fidelity but do not reliably preserve the production context that defines modern recordings, including contributor roles, tool identifiers and versions, session topology, rights references, and derivation lineage [3][4]. In most professional workflows this contextual information resides within digital audio workstation (DAW) sessions, delivery metadata, and platform databases rather than in the distributed object itself. Consequently, asset identity and provenance are externalized across heterogeneous systems and message exchanges, increasing reconciliation overhead, reducing auditability, and weakening deterministic verification when assets are transcoded, repackaged, or versioned across platforms.

This paper specifies the Digital Audio Assets Protocol (DAAP) as an asset-definition framework that binds distributable audio payloads to a canonical manifest and integrity layer. The canonical manifest records asset identity, contributor attribution, lineage relationships, and declared production context, while the integrity layer provides cryptographic verification through payload hashing, manifest signing, and optional trusted timestamping [5][6][7]. In doing so, DAAP provides a deterministic verification substrate for modern music distribution infrastructure while remaining compatible with current rendering, distribution, and playback workflows

---

For the purposes of this paper, "layer" is used in two distinct senses. First, it refers to the internal logical organization of a DAAP asset such as payload layer, manifest layer, and integrity layer. Second, it may refer to the broader external systems stack in which DAAP operates, including container, delivery, registry, and reporting infrastructures. These meanings are related but not interchangeable and should not be interpreted as standardized external layers within existing audio container or distribution system architectures.

1. Introduction

Contemporary music production is fundamentally system-defined. A musical work is authored within a digital audio workstation (DAW) environment composed of multitrack arrangements, routing graphs, signal processors, automation states, iterative edits, and toolchain dependencies. These elements collectively define the sonic result and its production lineage.

By contrast, the distribution layer of the music industry remains file-defined. The delivered artifact is typically a rendered audio container, most commonly a linear PCM waveform encoded within a WAVE/BWF structure or a derivative compressed format accompanied by metadata exchanged through external messaging systems and platform-specific schemas. While such containers preserve acoustic fidelity and limited descriptive fields, they do not reliably encapsulate the production context that generated the recording [5][8].

This structural separation introduces a persistent identity discontinuity between the studio environment and the distributed asset. Attribution, tool identity, session topology, and rights information are frequently maintained in independent registries or database systems rather than traveling with the distributed object itself. As audio files are transcoded, repackaged, or ingested across distribution networks, the contextual data required for provenance verification and durable attribution becomes fragmented across heterogeneous infrastructures including archival metadata frameworks, distribution messaging systems, and registry databases [9][10].

The Digital Audio Assets Protocol (DAAP) is proposed as a systems-level response to this architectural gap. DAAP defines a distributable audio asset as a composite structure consisting of:

1. Audio payloads, typically rendered waveform representations (e.g., canonical PCM or derivative encodings);

2. A canonical manifest, expressing identity, attribution, session lineage, and external registry references in a portable machine-readable structure; and

3. Integrity material, including cryptographic hashes and signatures sufficient for deterministic verification of the asset and its manifest.

Rather than extending legacy metadata fields within existing containers, DAAP treats the distributed recording as a verifiable object whose identity and provenance are defined by the manifest layer. This model enables contextual information originating in the production environment to remain bound to the audio payload even when the payload itself undergoes format conversion, normalization, or repackaging during downstream distribution processes.

The architectural approach aligns with broader developments in digital media provenance, where manifest-based verification models and cryptographic trust frameworks have been adopted to provide tamper evidence and persistent identity across heterogeneous media supply chains [11][12]. Within the audio domain, DAAP similarly aims to establish a stable object reference that can interoperate with existing metadata standards, registry identifiers, and distribution messaging systems without requiring modification to current playback pipelines.

1.1 Scope and None-Goals

The Digital Audio Assets Protocol (DAAP) is scoped to authorized supply-chain environments, including creation, mastering, release packaging, archival preservation, and controlled distribution. The protocol is designed to interoperate with existing distribution standards and identifier registries, including DDEX messaging frameworks, without requiring modification to existing playback pipelines, audio codecs, or distribution infrastructure [13][14].

DAAP defines a portable asset structure that binds distributable audio payloads to a canonical manifest and associated integrity material. This structure enables an audio asset to carry verifiable identity, attribution, and lineage across heterogeneous distribution systems while remaining independent of the underlying audio container.

DAAP does not replace existing audio containers or distribution messaging frameworks. Instead, it introduces an asset-level identity layer that can interoperate with established infrastructure. DAAP does not:

1. define streaming payout mechanics or accounting policy;
2. require modification of existing DSP playback pipelines;
3. implement post-distribution access control (DAAP is not a DRM framework); or
4. guarantee the factual accuracy of asserted credits or rights claims.

The protocol provides tamper evidence and signer accountability rather than adjudication of rights. This means that the protocol operates above the container layer and below distribution messaging systems, providing a stable identity and verification substrate for audio assets.

1.2 Deterministic Verification, Canonicalization, and Cryptographic Baselines

Because DAAP asserts verifiable identity and provenance, the protocol must define deterministic signing and verification behavior rather than relying on informal metadata conventions. Asset integrity claims therefore require canonicalization procedures, structured signing envelopes, and algorithm baselines that can be independently implemented across heterogeneous verification environments.

Accordingly, DAAP adopts canonicalization mechanisms compatible with deterministic signing frameworks such as the JSON Canonicalization Scheme (JCS) [15], structured signing envelopes including CBOR Object Signing and Encryption (COSE) [16] and recognized cryptographic algorithm baselines for hashing and digital signatures consistent with NIST Secure Hash Standard and Digital Signature Standard guidance [6][7]. These mechanisms ensure that manifest structures produce stable cryptographic digests regardless of serialization environment while enabling verifiers to confirm asset integrity and signer identity through established public-key infrastructure practices.

The protocol further aligns its packaging and disclosure semantics with emerging provenance frameworks that support embedded, sidecar, and repository-based manifest stores for media assets

[11]. This allows DAAP assets to interoperate with existing provenance and archival verification models while maintaining cryptographic agility and schema evolution over time.

2. Systems Context

Digital music distribution operates across two distinct but interdependent technical layers. First, recorded works are rendered into deterministic digital signal representations suitable for storage and transmission. Second, these representations move through a distributed network of metadata, delivery, and reporting systems operated by multiple independent stakeholders. Understanding the architectural separation between signal representation and distribution infrastructure is necessary to explain why production context and provenance information frequently become externalized from the distributed audio object.

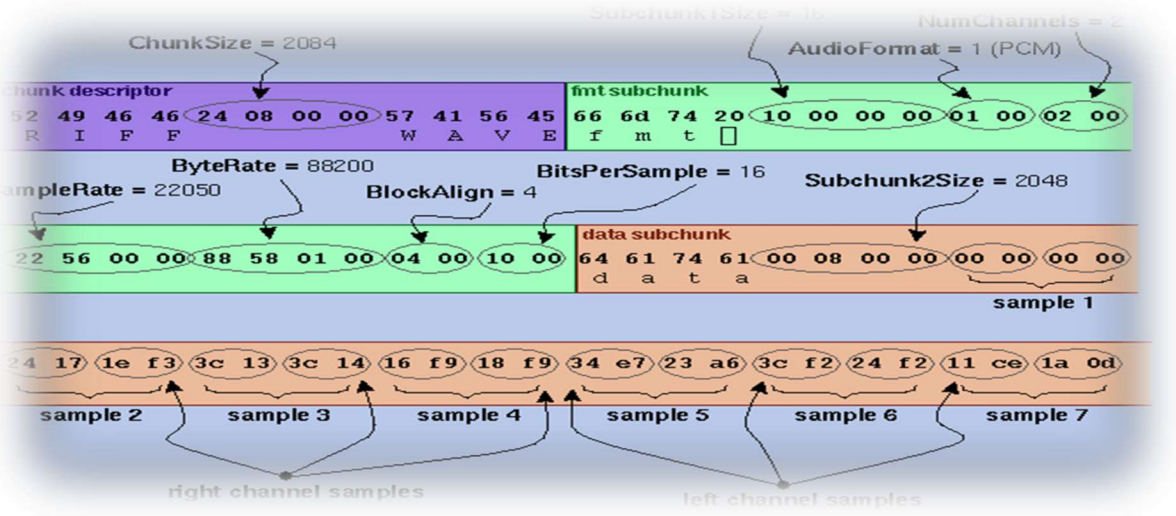
2.1 Canonical Audio Representation

The acoustic core of a DAAP asset remains a conventional digital audio container. DAAP does not define a new audio encoding format. Instead, it relies on a canonical waveform representation that serves as the deterministic acoustic anchor for the asset.

In most professional production environments this canonical representation is a linear Pulse Code Modulation (PCM) waveform encoded within a WAVE or Broadcast Wave Format (BWF) container [3][1][2]. PCM audio represents a sampled analog waveform by recording discrete amplitude values at uniform temporal intervals. Each sample is quantized to a fixed numerical range defined by the bit depth of the encoding system.

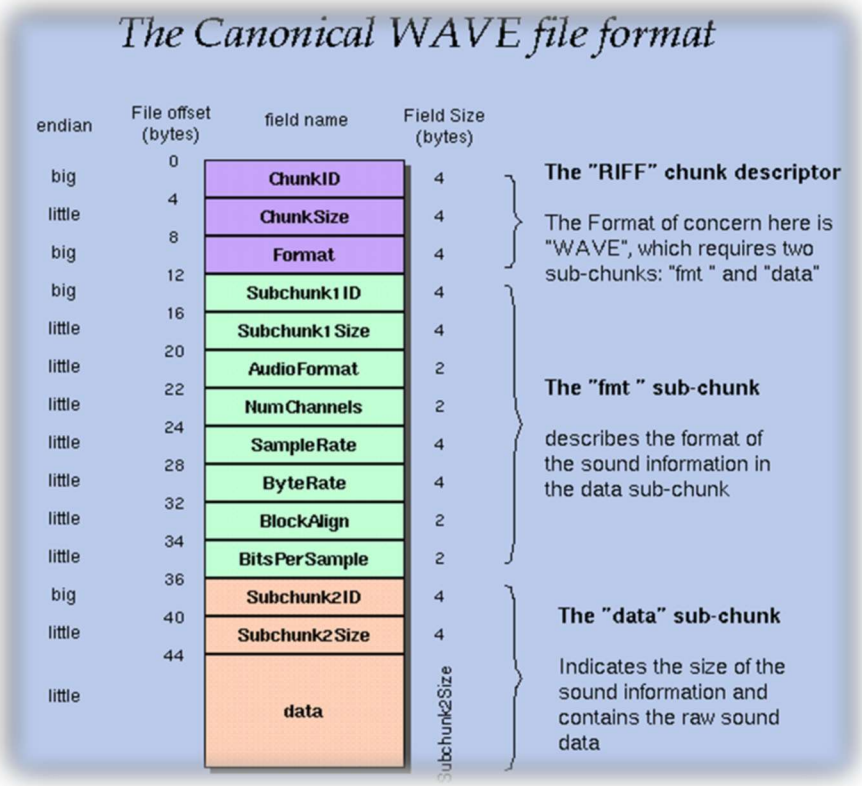
For example, a 24-bit PCM signal provides 2^24 discrete quantization levels per sample, enabling high dynamic range and extremely low quantization noise suitable for professional recording and mastering environments [6]. The waveform therefore functions as a deterministic digital representation of the acoustic outcome of a recording process.
Within RIFF/WAVE containers, the canonical waveform is defined by several structural parameters, including:

- Bit depth (commonly 16-bit or 24-bit)
- Sample rate (44.1 kHz, 48 kHz, 96 kHz, or higher)
- Channel configuration (mono, stereo, or multichannel)
- Interleaving structure within the RIFF data chunk
- Time reference and metadata extensions such as BEXT, iXML, or aXML



Illustrated by: craig@ccrma.stanford.edu http://www.ora.com/centers/gU/formats/micriU/index.htm

The Broadcast Wave Format specification extends the base WAVE container by introducing structured metadata chunks that support limited production and archival metadata including time references, originator information, and descriptive annotations [3][4]. These parameters collectively define the deterministic signal representation of the recording.



However, while the waveform accurately captures the acoustic outcome of a production process, it does not encode the production environment that generated it. The container structure therefore cannot reliably express in studio attribution contextual information. Even when auxiliary metadata chunks are used, most production context remains stored within DAW project files, studio asset databases, or external metadata registries, rather than traveling with the distributed audio object.

To illustrate how container-level transformations occur during routine production workflows, a reference waveform rendered from the Reaper DAW session was processed in Audacity for basic mastering adjustments and subsequently exported into multiple delivery formats. Examination of the resulting files at the byte level reveals that the acoustic payload (PCM sample data) may remain unchanged while the surrounding container structure and metadata headers are rewritten by the exporting application.

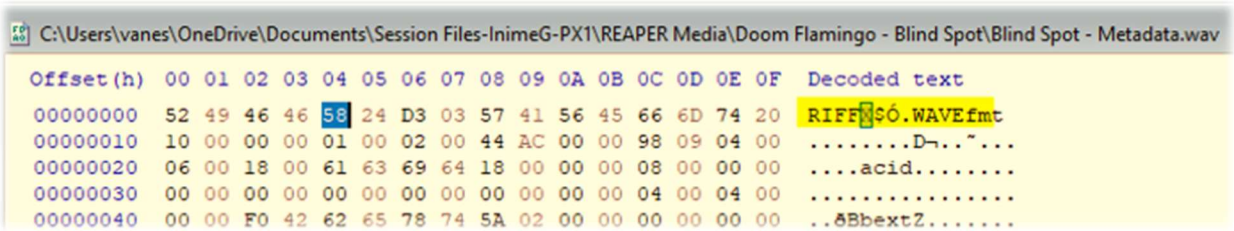
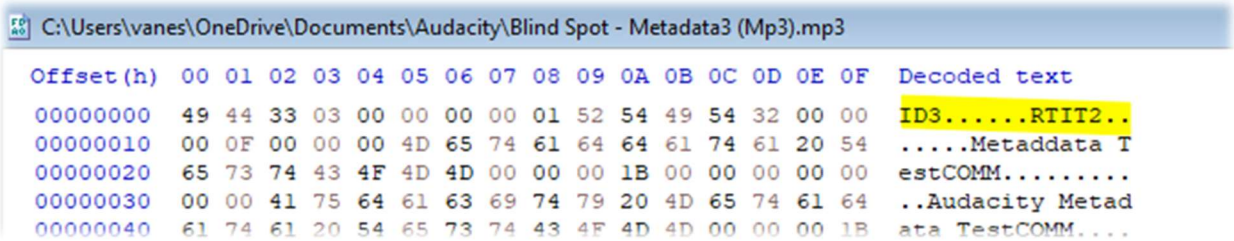


Illustration 2.1a – Original exported stereo wave file using the Reaper DAW.

Inspection of the example exported file in a hexadecimal viewer shows the insertion of an ID3, fLac and OggS for MP3, fLac and Ogg Vorbis respectively, metadata block preceding the RIFF/WAVE container header, indicating that application-level metadata has been embedded alongside the waveform payload. Such transformations are common across audio editing and mastering tools, which may add or modify container-level metadata fields during export operations. While these modifications do not alter the acoustic content of the recording, they change the binary representation of the file and may introduce additional metadata structures that were not present in the original render. As a result, the distributed file can differ structurally from the canonical studio output even when the underlying signal remains identical.

Consequently, as seen in these simple transcoding examples, the canonical waveform functions as the acoustic anchor of the recording, but not as a comprehensive representation of its production provenance.



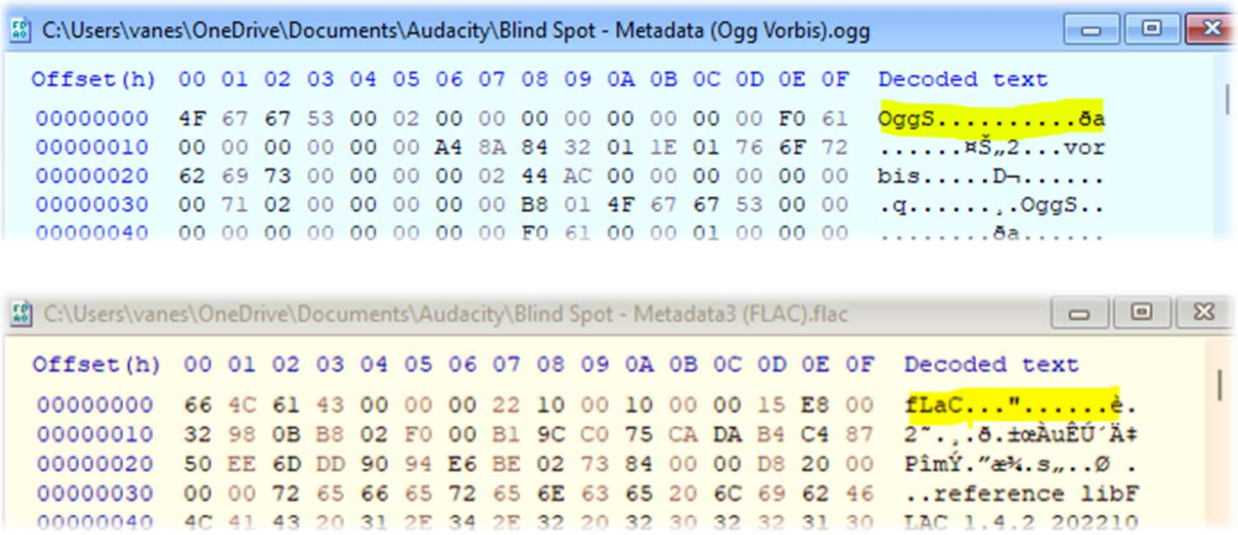


Illustration 2.1b - Wave metadata transcoded into MP3, FLAC & Vogg Orbis containers via the Audacity audio editor.

Lossy and consumer-oriented audio containers prioritize playback compatibility over structural preservation. When Broadcast Wave and XML metadata are transcoded into FLAC or AAC containers, structured provenance and session-level metadata are typically discarded, reducing the asset to flattened descriptive tags. Only RIFF-based containers natively support embedded, extensible, and inspectable production metadata suitable for enforceable attribution systems.

2.1 Structural Limitations of File-Centric Audio Distribution

Under contemporary file-based distribution architectures, no deterministic mechanism exists to bind a rendered audio container to the full set of contextual information required to establish the identity and operational meaning of the recording. Existing audio container standards were designed primarily to represent sampled acoustic signals with RIFF/WAVE and Broadcast Wave Format (BWF) define structures for encoding linear PCM audio and limited descriptive metadata, but they do not enforce deterministic propagation of contextual metadata across editing, rendering, or distribution transformations [3][1][2]

Archival and preservation guidance similarly recognizes that metadata persistence across production tools and export pipelines is inconsistent. FADGI guidance on Broadcast Wave metadata preservation notes that embedded metadata structures may be truncated, rewritten, or discarded during editing and transcoding operations unless applications explicitly preserve those structures [4].

Table 2.1a - Common Audio Metadata Formats and Typical Ingestion Contexts

Metadata Format	Embedded / Ingested In	Common Usage Environments
Common (Core Tags)	Container-native tags (varies by format)	GEN, DSP, LIB
APE (APEv2)	APE container, sometimes WAV	ARC, LIB
ASWG / AudioMD	Broadcast WAV preservation metadata	ARC, LIB
AXML	XML chunk embedded in BWF	BCAST, POST, ARC

BWF (bext)	Broadcast WAV metadata chunk	BCAST, POST, ARC
CAFINFO	CAF (Core Audio Format) metadata fields	APPLE, POST
CART	CART chunk in WAV/BWF	RADIO
CUE	RIFF cue chunk / markers	POST, MASTER
FLACPICT	FLAC picture block	DSP, LIB
ID3	MP3 tags / MP4 variants	DSP, CONSUMER
IFF	AIFF container chunks	POST, MUSIC
INFO	RIFF INFO chunk	LEGACY, WIN
XML (generic)	Embedded XML or sidecar	MAM, DIST
VORBIS	Vorbis Comment blocks	DSP, LIB
WAVEXT	WAVE extensible format header	POST, MASTER
XMP	Embedded XMP packet	MAM, MEDIA

GEN - General-purpose metadata, DSP - Digital Service Provider, LIB – Library, ARC – Archival, BCAST - Broadcast systems, POST - Post-production, APPLE - Apple/Core Audio, RADIO - Radio automation, MASTER – Mastering, CONSUMER - End-user, MUSIC - General music, LEGACY – Older, WIN - Windows-specific, MAM - Media Asset Management, DIST - Distribution systems, MEDIA - Cross-media environments

When identity and attribution are not intrinsically attached to the distributed asset, downstream systems must maintain contextual meaning through external databases and registry mappings. Distribution metadata frameworks such as DDEX Electronic Release Notification (ERN) and Digital Sales Reporting (DSR) therefore operate primarily as system-to-system communication protocols rather than as intrinsic asset identity mechanisms [13][17].

Under these conditions, identity continuity becomes dependent on cross-system reconciliation rather than deterministic verification.

Table 2.2a - Common Audio File Formats and Typical Usage Environments

Audio File Format	Encoding Type	Common Usage Environments
WAV (RIFF/WAVE)	Linear PCM	POST, MASTER, BCAST, DIST
BWF (Broadcast WAV)	Linear PCM with broadcast metadata	BCAST, POST, ARC
AIFF	Linear PCM	MUSIC, POST, APPLE
AIFF-C	PCM or compressed	POST, LEGACY
FLAC	Lossless compression	DSP, DIST, LIB
ALAC	Lossless compression	APPLE, DSP
MP3	Lossy compression	DSP, CONSUMER
AAC / M4A	Lossy compression	DSP, CONSUMER, MOBILE
Ogg Vorbis	Lossy compression	DSP, OPEN
Opus	Lossy compression	STREAM, WEB
CAF (Core Audio Format)	Flexible container (PCM or compressed)	APPLE, POST
WavPack	Hybrid lossless	ARC, LIB
Monkey's Audio (APE)	Lossless compression	ARC, LIB
DSD / DSF / DFF	Direct Stream Digital	AUDIOPHILE, MASTER
Dolby Atmos ADM-BWF	Object-based audio container	POST, FILM
MXF Audio	PCM within MXF container	BCAST, FILM
OGG/FLAC hybrids	Lossless	OPEN, LIB

POST - Post-Production; MASTER - Mastering / Final Delivery; BCAST – Broadcast; DIST - Distribution; MUSIC - Music Production; APPLE - Apple / Core Audio Ecosystem; LEGACY - Legacy Systems; DSP - Digital Service Providers; LIB - Library / Collection Management; CONSUMER - Consumer Playback; MOBILE - Mobile Devices; OPEN - Open / Open-Standard Systems; STREAM - Streaming Systems; WEB - Web-Based Environments; ARC - Archival / Preservation; AUDIOPHILE - High-Fidelity / Audiophile Systems; FILM - Film / Cinema Production

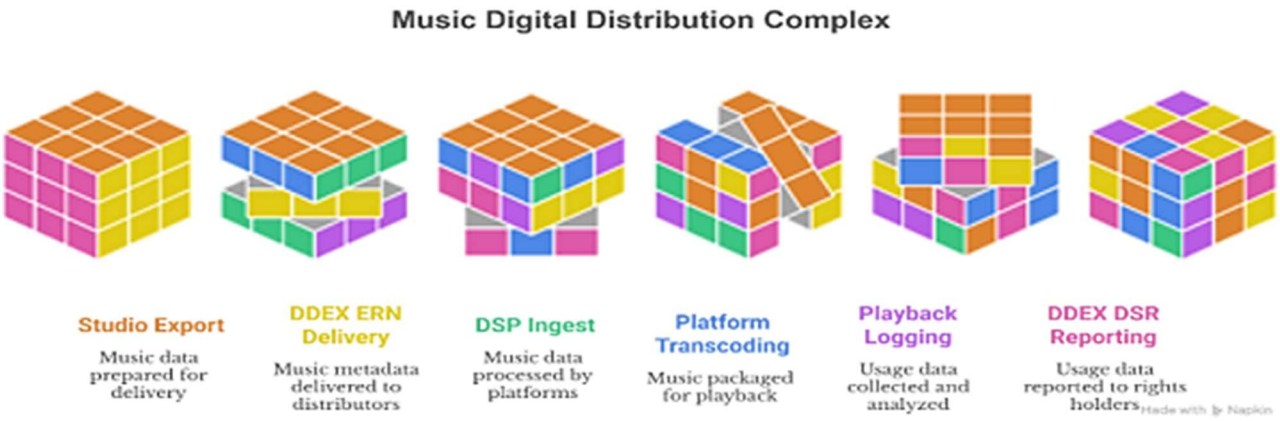
2.2 Distribution and Verification Infrastructure

Digital music distribution is best understood as a relay of independently operated systems rather than a single end-to-end workflow. Labels, distributors, digital service providers (DSPs), rights societies, publishers, and accounting platforms each maintain their own identifiers, validation rules, and database authorities. Interoperability between these systems is achieved through standardized metadata exchanges rather than through a persistent asset identity layer.

In the recorded-music pipeline, release delivery commonly begins with a distributor or rights holder transmitting release metadata and commercial terms to DSPs using the DDEX Electronic Release Notification (ERN) message suite [13]. Consumption events occurring on DSP platforms are subsequently communicated back to licensors through reporting standards such as the DDEX Digital Sales Reporting (DSR) message suite [13]. These message frameworks provide structured communication between systems, but they do not function as an authoritative identity mechanism for the underlying audio asset.

In practice, the same recording may be represented by multiple identifiers across the distribution chain, including distributor catalog identifiers, DSP-specific internal asset identifiers, International Standard Recording Codes (ISRC), composition identifiers such as ISWC, internal, and database primary keys within reporting systems.

Each recorded music digital distribution participant maintains its own internal identifiers, validation rules, and catalog representations [18][14][19]. As a result, the continuity of identity and attribution is achieved through reconciliation across systems rather than through verification of a single authoritative object [18][14][19].



2.3 Distribution Transformation: Transcoding, Repackaging, and System-Level Rebinding

Distribution pipelines for digital audio assets consist of heterogeneous system boundaries, each enforcing distinct schema constraints, ingestion protocols, and identifier frameworks. Metadata continuity across these systems is not guaranteed and depends on explicit cross-system alignment. EBU and related preservation guidance on broadcast and production metadata consistently note that metadata preservation is implementation-dependent and non-deterministic across tools and workflows [3][4][20][1].

DDEX specifications further formalize this separation by defining metadata exchange as message-based (e.g., ERN XML) rather than container-bound, explicitly decoupling metadata transport from the audio payload itself [18][21]. EBU guidance on file-based workflows, together with archival and preservation guidance for WAVE/BWF handling, indicates that such transformations are expected and systemic rather than exceptional [3][4][20][1].

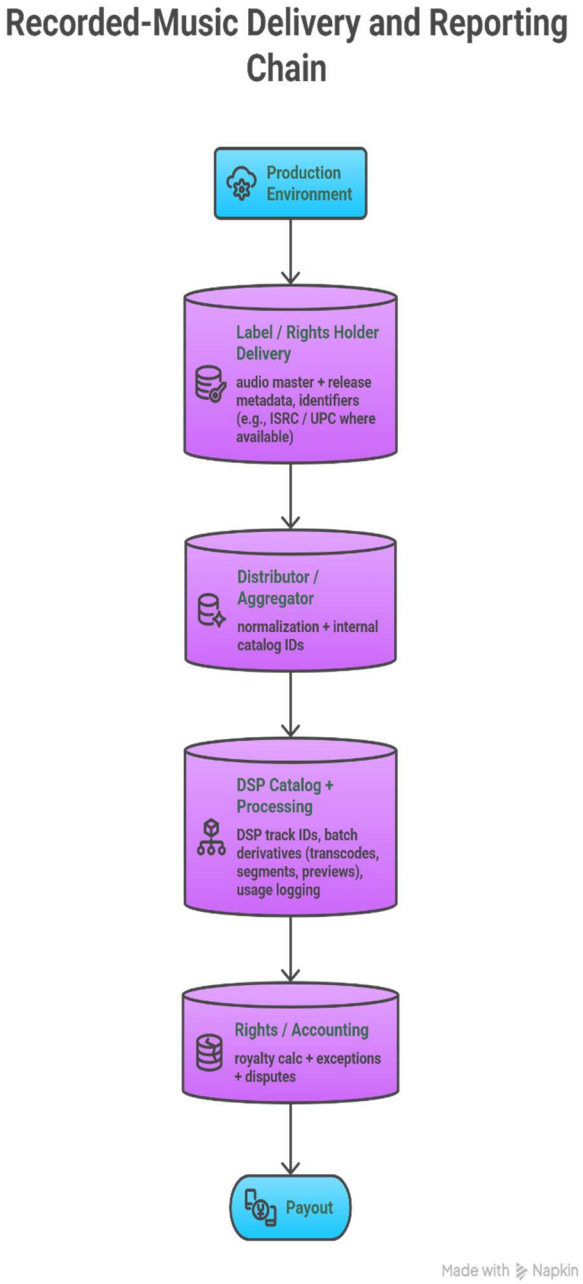
In the recorded music pipeline, a label or rights holder typically delivers a master audio representation and release metadata to a distributor. The distributor normalizes fields, assigns internal catalog identifiers, and transmits delivery packages to DSPs using structured delivery frameworks (commonly DDEX Electronic Release Notification messages) together with the audio payload [17][21].

DSPs ingest these deliveries, assign platform specific track identifiers, and generate internal derivatives for playback and platform operations through batch processing [18][14]. These derivatives commonly include codec transcodes, loudness normalization variants, preview clips, segmented streaming objects, and content recognition artifacts [14][22]. Usage events are logged within the DSP's environment and later emitted as sales/usage reports back to rightsholders (commonly using DDEX Digital Sales Reporting messages) [14][23]. Royalties are then calculated and paid downstream according to contract terms, territory rules, recoupment logic, and administrative splits [14][19].

Cross-platform verification typically begins with identifier matching. When identifiers are present and correct, systems can map a recording to an internal catalog entry with relatively low ambiguity. When identifiers are missing, inconsistent, or mis-assigned or when versions proliferate systems often rely on signal-based matching (fingerprinting) to determine whether two audio signals resemble each other, particularly in contexts involving re-uploads, re-encodes, or user-generated content.

This distinction matters because fingerprinting is designed primarily for similarity detection and enforcement workflows. It can answer whether two signals are likely the same or substantially similar. It does not reliably encode the higher-order semantics that distribution and licensing decisions often require, such as which version is authoritative, whether a derivative was generated under an authorized rule set, how the work relates to upstream session lineage, or which slice of rights is implicated.

In practice, platform truth becomes an approximation built from three mechanisms: identifier matching where possible, fingerprinting where necessary, and dispute processes where the first two cannot resolve nuance.




```
(1) Identifier matching (deterministic when
correct) ISRC / UPC / internal IDs align →
catalog linkage
```

```
(2) Fingerprinting (robust for similarity,
weak for rights nuance) "Does this resemble
that?" → detection / enforcement workflows
```

```
(3) Exceptions + disputes (manual
resolution) Version ambiguity / split
ambiguity / deal ambiguity → human
process
```

These mechanisms are not failures; they are reasonable responses to an architecture in which identity is external to the distributed audio object.

2.3.1 Non-Uniform Interpretation Across Tools and Platforms

The operational issue DAAP is concerned with is not whether metadata exists at any given hand off. The issue is that the "thing being reported on" is repeatedly re expressed as it moves downstream. Identity is not carried by a single authoritative asset object. Instead, identity is distributed across databases and message exchanges that are locally coherent but globally non- uniform and non-authoritative [18][14][24][19]. Where alignment fails or becomes ambiguous, downstream processes increasingly depend on manual reconciliation, exception handling, and dispute workflows [25][19].

In practical terms, this is where "data breaks" become "money leaks": payments are delayed, misallocated, or placed into exception queues when attribution, ownership, or version identity cannot be linked with sufficient confidence across systems [25][19].

Even standardized identifiers do not resolve this problem automatically. For example:

- an **ISRC** may be omitted from a delivery message, duplicated across alternate assets, or associated with the wrong recording variant during redelivery or catalog normalization.
- an **ISWC** may be absent altogether at the time of release delivery or remain only partially linked to the corresponding recording in downstream rights systems.

- distributor and DSP systems may assign their own proprietary catalog or asset identifiers, which become locally authoritative even when upstream standards identifiers are present [25][26][27][3][4].

In practical nomenclature, this means the same commercial object may simultaneously be represented as:

• **ISRC:** US-XXX-25-12345 or GB-ABC-24-54321 for the sound recording [28].

• **ISWC:** T-123.456.789-0 for the underlying musical work [29].

• **UPC/EAN:** release-level commerce identifier.

• **Distributor internal ID:** platform-specific catalog key such as release/resource database keys.

• **DSP internal track ID / release ID:** service-scoped identifiers not portable outside the platform environment [25][26][30][31][2][19].

Where identifiers remain consistent across systems, verification can occur deterministically through identifier matching [25][26]. When identifiers are absent, inconsistent, or drift across re- deliveries, platforms frequently rely on content-based audio fingerprinting systems to detect similarity between recordings [26][33].

Large-scale systems such as YouTube's Content ID demonstrate the effectiveness of fingerprinting for similarity detection and automated enforcement workflows [32]. However, fingerprinting systems are optimized to answer the question "does this audio resemble that audio?" rather than to represent production context, contributor attribution, or authorized derivative lineage [27][34][7][33].

From a systems perspective, this architecture externalizes the authoritative meaning of a recording into distributed databases and reconciliation processes rather than binding that meaning to the distributable audio object itself [25][64][27].

This structural separation between signal representation and identity continuity establishes the conditions under which provenance, attribution, and lineage information become fragmented across the distribution pipeline [25][26][34][35][7].

2.4 Scope of the Systems Limitation in the Post-Generative Context

The underlying failure mode is architectural rather than procedural. A rendered audio container cannot function as a complete, authoritative digital asset. It is a signal-bearing output representation, not a self-describing identity object [34][35][36][16].

Absent a structured mechanism that binds attribution, production context, lineage, and integrity assertions at the moment of asset formation, the digital audio ecosystem defaults to post hoc reconstruction rather than deterministic verification. Under this model, identity is inferred from external registries, delivery messages, platform databases, and reporting systems rather than validated from a single authoritative asset object [25][26][7].

This approach does not scale to contemporary production, distribution, archival, legal, and AI-mediated operating conditions [37][20][38]. Under these conditions, legacy best-effort metadata practice becomes operationally inadequate. AES and EBU metadata guidance already treats

metadata survivability across tools and transformations as a systems requirement rather than a cosmetic convenience [9][12]40][34][35].

DAAP is introduced as a protocol-level asset-layer topological response to this architectural deficiency by restoring identity, provenance, and attribution as intrinsic properties of the asset itself rather than downstream database artifacts [39][37][7].

2.4.1 Provenance Ambiguity and Rights-Enforcement Degradation

Where provenance is not bound at formation, downstream systems cannot deterministically establish:

- whether one asset is a derivative of another;
- which tools, models, or processing environments participated in formation;
- whether licensing constraints attach to the asset, the composition, the model, or only a downstream manifestation;
- whether a usage claim corresponds to the intended recording/work pair.

In the absence of asset-bound provenance, enforcement layers increasingly fall back to probabilistic content matching rather than authoritative metadata validation. Fingerprinting can identify acoustic similarity, but it does not encode role attribution, authorized derivative status, model lineage, or asset-scoped rights context [22]. As a result, disputes migrate from

deterministic verification into exception queues, manual reconciliation, and platform-specific policy workflows [26][2][19].

2.4.2 Structural Limitation of Treating Assets as Interchangeable Content

Without embedded identity and verifiable provenance, works that are structurally distinct become operationally indistinguishable once reduced to flattened payloads. Distinct contributor graphs, toolchains, derivation histories, and rights states collapse into files that may be acoustically similar yet semantically non-equivalent [34][35][36][16].

That collapse degrades attribution fidelity, distorts economic signaling, and weakens the informational substrate required for credits, ownership allocation, licensing decisions, compliance review, and compensation logic. In effect, the ecosystem prices and distributes signal while losing the structured evidence required to govern the asset.

2.5 Metadata Architectural Conclusion

The operative deficiency is not metadata sparsity but authority-surface non-unification. In the extant distribution topology, carrier-layer representations preserve signal state, exchange-layer schemas preserve transaction state, and platform-resident databases preserve locally scoped operational state. None of these strata provides a serialization-invariant, asset-scoped, cryptographically attestable identity substrate spanning production export, post-render transformation, delivery-message transport, platform ingest, derivative generation, reporting, and archival persistence [9][12][25][26][40][34][35][36][16].

The resulting architectural condition is asset-state fragmentation. Signal-bearing payloads remain distributable; identity-bearing semantics do not. Consequently, downstream economic, legal,

archival, and enforcement processes operate on reconciled surrogates rather than on a singular authoritative asset representation. Under high-throughput and generative production conditions, such reconciliation-centric topology becomes computationally brittle, operationally non-scalable, and evidentiary deficient [39][37][20][63][78].

The antecedent condition is not recent. Prior AES work had already formalized the metadata problem as a systems-modeling deficiency, not a file-tagging deficiency. At the AES Metadata Symposium (2003), Casey defined metadata as an information-systems construct whose operational validity depends on concurrent constraint across semantics, structure, encoding, applicability, storage model, and transport, and further distinguished wrapped and unwrapped metadata as separable but failure-prone coupling regimes [9]. Under that formulation, persistence failure is not reducible to omitted fields; it arises from representational dissociation across heterogeneous storage and exchange surfaces.

This position is reinforced by subsequent AES research on music information modeling. Abdallah, Raimond, and Sandler demonstrate that hierarchical or flat tagging regimes are structurally inadequate for domains in which signals, performances, agents, events, computational transforms, and derived artifacts must be modeled as formally related entities rather than as loosely attached descriptors [1]. In such environments, key-value metadata is not merely incomplete; it is ontologically insufficient. The required object space is relational, typed, and inference-bearing.

The historical implication is specific. The unresolved constraint is not the absence of identifiers, container fields, or descriptive schemas in isolation. The unresolved constraint is the absence of a portable, authoritative, machine-interpretable object in which identity, attribution, provenance, and derivational state are co-resident and verification-addressable. File-embedded metadata improves discoverability and partial interchange; it does not instantiate an asset-scoped authority model [9][1].

Accordingly, the deficiency is not appropriately characterized as metadata incompleteness. It is more precisely characterized as authority decomposition across non-isomorphic system boundaries of asset-state non-determinism under heterogeneous transformation. Earlier AES work identified this condition at a smaller operational scale; contemporary distribution volume, platform heterogeneity, and generative throughput amplify its impact but do not alter its underlying structure.

The next section therefore does not revisit the metadata problem descriptively. It addresses the protocol-design constraints implied by that problem: authoritative claims-object placement, packaging topology, canonicalization semantics, payload-binding semantics, namespace typing, derivation modeling, and verifier-state determinacy.

3. Asset Packaging, Manifest Authority, and Verification Semantics

DAAP v1 defines the distributable music asset as a compound verification object rather than a flat render plus external metadata. The asset is composed of:

(1) an authoritative manifest store, (2) one or more bound payloads, and (3) integrity material sufficient to make asset state reproducibly verifiable across heterogeneous environments. Authority

is therefore localized in the manifest store rather than in the waveform container. Payloads remain codec- and container-specific manifestations; identity, lineage, and verification semantics do not. This is the central packaging asymmetry of DAAP v1.

The normative packaging surface is constrained deliberately. The deep-dive specification defines one primary packaging model, the DAAP Bundle, and one optional compatibility bridge, the sidecar / repository manifest-store profile. The constraint is methodological: packaging variability is a verifier-divergence risk. A single primary model reduces ambiguity in asset discovery, content binding, active-manifest selection, and conformance testing.

3.1 Normative packaging objective

The packaging objective is not maximal metadata carriage. The packaging objective is portable authority preservation under ordinary media transformation. A conformant asset must therefore satisfy four conditions:

1. Durable portability - the asset remains interpretable and verifiable without dependence on a single downstream database.

2. Deterministic binding - the manifest store binds payload membership through cryptographic content bindings.

3. Canonical signing - manifest bytes are canonicalized prior to hashing and signature generation.

4. Verifier determinacy - independent implementations must derive the same verification inputs and the same result classes from the same asset state.

3.2 Primary model: DAAP Bundle Container

DAAP v1 SHALL define a bundle container as the normative transport object. The bundle contains exactly one authoritative manifest store, one or more audio payloads, optional auxiliary artifacts, and optional export mappings for legacy systems. Internal filesystem layout is non- authoritative; the manifest store is authoritative. Payload presence alone is non-normative unless the payload is referenced and bound from the manifest store.

Snippet 3-1. Bundle construction skeleton (Python)

```
from __future__ import annotations
from pathlib import Path
import shutil
from typing import Iterable
def build_daap_bundle(
    bundle_root: Path,
    manifest_store_bytes: bytes,
    audio_payloads: Iterable[Path],
    aux_payloads: Iterable[Path] = (),
    export_artifacts: Iterable[Path] = (),
) -> None:
    """
    Normative packaging skeleton for a DAAP Bundle.
    Authority is localized in /manifest_store/daap.manifeststore.
    Payload directories are transport layout only; they are not authoritative
    unless referenced by manifest bindings.
    """
    (bundle_root / "manifest_store").mkdir(parents=True, exist_ok=True)
    (bundle_root / "payloads" / "audio").mkdir(parents=True, exist_ok=True)
```

```
    (bundle_root / "payloads" / "aux").mkdir(parents=True, exist_ok=True)
    (bundle_root / "metadata").mkdir(parents=True, exist_ok=True)
    # Write the authoritative manifest store.
    (bundle_root / "manifest_store" / "daap.manifeststore").write_bytes(
        manifest_store_bytes
    )
    # Copy payloads into transport layout.
    for src in audio_payloads:
        shutil.copy2(src, bundle_root / "payloads" / "audio" / src.name)
    for src in aux_payloads:
        shutil.copy2(src, bundle_root / "payloads" / "aux" / src.name)
    # Optional bridge/export artifacts (e.g., DDEX mapping XML).
```

```
    for src in export_artifacts:
        shutil.copy2(src, bundle_root / "metadata" / src.name)
```

This snippet is written the way it is because the specification explicitly separates transport layout from authority layout: /manifest_store/daap.manifeststore is authoritative, while payload paths are non-authoritative and validated through manifest bindings.

3.3 Payload addressing and content binding

The normative bundle supports two payload addressing modes: embedded payloads and referenced payloads. However, regardless of locator mode, every referenced payload MUST carry at least one cryptographic binding, with SHA-256 as the minimum baseline. Locator strings are convenience surfaces, not identity surfaces; verifiers determine payload identity from digest agreement, not path strings. This distinction becomes critical under copy, mirroring, bundle relocation, or sidecar delivery.

Snippet 3-2. Payload binding insertion

```
from __future__ import annotations
from pathlib import Path
import base64
import hashlib
from typing import Any
def sha256_bytes(data: bytes) -> bytes:
    h = hashlib.sha256()
    h.update(data)
    return h.digest()
def sha256_file(path: Path) -> bytes:
    h = hashlib.sha256()
    with path.open("rb") as f:
        for chunk in iter(lambda: f.read(1024 * 1024), b""):
            h.update(chunk)
    return h.digest()
def attach_payload_bindings(manifest: dict[str, Any], payload_paths: list[Path]) -> None:
    """
    Inserts baseline content bindings into manifest['payloads'].
    DAAP baseline requires SHA-256 for every referenced payload.
    """
```

This implementation follows the deep-dive requirement that every payload referenced by the manifest store includes at least one cryptographic binding and that the baseline digest is SHA- 256. It also keeps locator as a transport hint rather than a trust anchor.

3.4 Canonical manifest and deterministic signing

A DAAP manifest is not signable in its source form unless canonicalization is fixed. The deep- dive specification resolves this by requiring JSON as the primary manifest encoding and RFC 8785 JCS as the canonicalization rule for any manifest that is hashed or signed. An optional CBOR+COSE compact profile is permitted, but it does not displace the JSON/JCS baseline. This hierarchy matters: the baseline interoperability anchor remains canonical JSON, not binary compactness.

Snippet 3-3. Canonicalization + manifest hash

```
from __future__ import annotations
import base64
import hashlib
from typing import Any
import jcs  # standards-compliant RFC 8785 implementation
def canonicalize_manifest(manifest_obj: dict[str, Any]) -> bytes:
    """
    Returns RFC 8785 JCS-canonical UTF-8 bytes.
    NOTE: json.dumps(sort_keys=True) is not a substitute for JCS.
    """
    return jcs.canonicalize(manifest_obj)
def compute_manifest_hash(manifest_obj: dict[str, Any]) -> dict[str, str]:
    canonical = canonicalize_manifest(manifest_obj)
    digest = hashlib.sha256(canonical).digest()
    return {
        "alg": "sha-256",
        "digest_b64": base64.b64encode(digest).decode("ascii"),
    }
```

The code is written this way because the specification explicitly states that semantically identical manifests must produce identical signing inputs across implementations, and that ordinary JSON serialization is insufficient for that property.

3.5 Optional bridge: sidecar / repository manifest store

DAAP v1 permits exactly one bridge profile for ecosystems that cannot yet ingest a bundle container. Under that profile, a conventional audio payload—e.g., master.wav—travels separately from master.daap.manifeststore, and determinacy is preserved by validating the manifest-store bindings against the payload bytes. This is not a fallback into arbitrary sidecar practice; it is a constrained bridge with the same authority model as the bundle.

Snippet 3-4. Sidecar verification skeleton

```
from __future__ import annotations
import base64
import hashlib
import json
from pathlib import Path
from typing import Any
import jcs
def verify_sidecar_payload(manifest_path: Path, payload_path: Path) -> bool:
    """
    Verifies that a sidecar manifest store binds the target payload bytes.
    Signature validation is omitted here; this snippet demonstrates content binding.
    """
    manifest = json.loads(manifest_path.read_text(encoding="utf-8"))
    expected_entries = manifest.get("payloads", [])
    payload_entry = next(
```

```
        (p for p in expected_entries if Path(p["locator"]).name == payload_path.name),
        None,
    )
    if payload_entry is None:
        return False
    supplied = next((h for h in payload_entry["hashes"] if h["alg"] == "sha-256"), None)
    if supplied is None:
        return False
    actual_digest = hashlib.sha256(payload_path.read_bytes()).digest()
    actual_b64 = base64.b64encode(actual_digest).decode("ascii")
    return actual_b64 == supplied["digest_b64"]
```

This skeleton enforces the bridge model correctly: verification attaches to digest agreement, not to co-location or filename convention alone.

3.6 Canonical manifest scope

The asset-architecture text is right to insist that the canonical manifest is the authoritative definition of the asset and that it carries identity, attribution, production topology, rights context, and schema versioning without collapsing into a DAW session format. It is also right to keep full executable DSP state, DAW edit constructs, and rights adjudication logic out of scope. The deep- dive synopsis model formalizes that boundary by constraining the portable production graph to typed nodes, typed edges, stable asset-scoped IDs, render bindings, and derivation relations-not plugin execution state.

3.7 First-pass architectural conclusion

At the end of this first pass, the specification surface is now strong enough to support implementation language. The bundle model gives you the transport object. JCS gives you deterministic claims bytes. SHA-256 gives you baseline content bindings. The sidecar bridge gives you transitional deployability. The synopsis graph gives you structural intelligibility without forcing DAW reconstitution. That is the correct direction for the paper: authority object first, payloads second, verification semantics explicit.

4. Canonical Manifest Format and Canonicalization for Signing

This section resolves the second reconstruction constraint: what the canonical manifest is, and how canonicalization is performed such that independent verifiers compute identical signing inputs from semantically identical manifest content. DAAP v1 therefore specifies a canonical manifest data model, one required on-the-wire encoding profile, and one optional compact profile.

4.1 Normative requirement: deterministic signing requires canonicalization

If DAAP assets are to support audit-grade integrity and interoperable verification, two verifiers operating in different environments SHALL compute the same signature-input bytes from the same manifest content. Common serialization formats do not guarantee this property by default. JSON, in particular, admits multiple equivalent textual representations of the same data model, including differences in whitespace, key ordering, and numeric formatting. DAAP v1 therefore treats canonicalization as a mandatory precondition for integrity, not as an implementation convenience.

4.2 Canonical manifest encoding: JSON + JCS

DAAP v1 SHALL use JSON as the primary canonical manifest encoding. For any manifest that is hashed or signed for integrity, DAAP v1 SHALL require the JSON Canonicalization Scheme (JCS) defined by RFC 8785. The purpose is twofold:

1. semantically identical manifests MUST produce identical canonical byte sequences; and
2. verification MUST be tool-agnostic and serializer-independent.

4.2.1 Canonicalization input and output

- The canonicalization procedure SHALL: take as input a UTF-8 JSON text representing the manifest data model;
- emit UTF-8 JSON in JCS-canonical form; and
- require verifiers to compute manifest hashes and signature inputs over the canonicalized UTF-8 byte sequence, not over source-form JSON bytes.

Snippet 4-1. Canonicalization and manifest digest

```
from __future__ import annotations
import base64
import hashlib
from typing import Any
import jcs  # RFC 8785 implementation
def canonicalize_manifest(manifest_obj: dict[str, Any]) -> bytes:
    """
    Returns RFC 8785 JCS-canonical UTF-8 bytes.
    """
    return jcs.canonicalize(manifest_obj)
def compute_manifest_hash(manifest_obj: dict[str, Any]) -> dict[str, str]:
    """
    DAAP v1 baseline manifest digest: SHA-256 over canonical bytes.
    """
    canonical_bytes = canonicalize_manifest(manifest_obj)
    digest = hashlib.sha256(canonical_bytes).digest()
    return {
        "alg": "sha-256",
        "digest_b64": base64.b64encode(digest).decode("ascii"),
    }
```

This implementation is correct because the specification requires that integrity be computed over canonical bytes, not over arbitrary serializer output. Ordinary json.dumps(..., sort_keys=True) is insufficient unless it is specification-equivalent to RFC 8785.

4.2.2 Canonical manifest data-model constraints

Canonicalization is insufficient if the data model itself remains ambiguous. To prevent signing gaps and interpretation drift, DAAP v1 SHALL:

- define a fixed set of top-level required objects and known extension points;
- forbid duplicate keys;
- require explicit schema-version fields (e.g., daap_version, schema_uri); and

- require stable, explicit typing for identifiers (e.g., strings for ISRC/ISWC and UUID-style strings for DAAP component IDs) rather than overloaded numeric forms.

Snippet 4-2. Minimal top-level schema-shape enforcement

```
from __future__ import annotations
from typing import Any
REQUIRED_TOP_LEVEL = {
    "daap_version",
    "schema_uri",
```

```
    "asset",
    "disclosure",
    "payloads",
    "integrity",
    "events",
}
```

```
def validate_manifest_shape(manifest_obj: dict[str, Any]) -> None:
    missing = REQUIRED_TOP_LEVEL - set(manifest_obj.keys())
    if missing:
        raise ValueError(f"Missing required top-level objects: {sorted(missing)}")
```

```
    if not isinstance(manifest_obj["daap_version"], str):
        raise TypeError("daap_version must be a string")
```

```
    if not isinstance(manifest_obj["schema_uri"], str):
        raise TypeError("schema_uri must be a string")
```

4.3 Optional compact profile: CBOR + COSE

DAAP v1 MAY support an optional compact profile for environments where binary efficiency, constrained transport, or embedded signing envelopes are operationally desirable. Under that profile, the canonical manifest MAY be encoded as CBOR and signatures SHALL be represented and verified using COSE, as defined in RFC 9052. The important boundary is normative: the CBOR/COSE profile is optional and does not replace the JSON/JCS baseline. JSON/JCS remains the required interoperability anchor. If CBOR is used, the signing input SHALL be the canonical CBOR byte representation defined by the chosen COSE message type, and verifiers SHALL validate that the COSE structure binds the same content claims as the equivalent JSON/JCS form. The v1 assumption is semantic bijection between the two forms to prevent profile drift.

Snippet 4-3. Semantic-equivalence boundary for optional CBOR profile

```
from __future__ import annotations
from typing import Any
def verify_semantic_equivalence(
    json_manifest: dict[str, Any],
    cbor_manifest: dict[str, Any],
) -> bool:
    """
```

```
    Placeholder semantic-equivalence check for DAAP's optional CBOR/COSE profile.
    The v1 assumption is semantic bijection.
    """
    return json_manifest == cbor_manifest
```

4.4 Manifest store object and signing envelope

DAAP v1 distinguishes between two object classes within the authority surface:

1. Manifest (claims) - the canonical data model describing identity, provenance, tool identifiers, lineage, and references;
2. Integrity material - hashes, signatures, and optional timestamps binding the manifest and referenced payloads.

The DAAP Manifest Store SHALL contain:

- exactly one active manifest (canonical JSON/JCS, or CBOR/COSE profile); and
- integrity material binding: of the manifest itself, of each referenced payload via content bindings, and of optional additional signed assertions.

The manifest store MAY contain historical manifests or countersignatures, but DAAP v1 SHALL define deterministic active-manifest selection semantics. This requirement prevents signature wrapping, manifest substitution, and "multiple truths" ambiguity.

Snippet 4-4. Logical DAAP manifest store object

```
from __future__ import annotations
from typing import Any
def build_manifest_store(
    active_manifest: dict[str, Any],
    content_bindings: list[dict[str, Any]],
    signatures: list[dict[str, Any]],
    timestamps: list[dict[str, Any]] | None = None,
) -> dict[str, Any]:
    return {
        "active_manifest": active_manifest,
        "content_bindings": content_bindings,
        "integrity": {
            "manifest_hash": None,
            "signatures": signatures,
            "timestamps": timestamps or [],
        },
    }
```

The logical separation is deliberate: claims and integrity are bound, but not conflated. The manifest store remains the authority object; signatures and timestamps remain the verification machinery.

5. Mandatory Algorithms, Crypto-Agility, and Verifier Result Semantics

This section resolves the third reconstruction constraint: which algorithms are mandatory, how algorithm agility is published without verifier divergence, and how verification outcomes are expressed in machine-actionable form. DAAP v1 therefore defines both a mandatory baseline and an explicit agility envelope.

5.1 Normative requirement: fixed baselines plus agility envelope

DAAP assets are intended to remain verifiable across time and across heterogeneous implementations. That requires:

1. a mandatory baseline that every compliant verifier can support; and
2. a crypto-agility mechanism that permits algorithm addition and deprecation without fragmenting interoperability.

DAAP v1 therefore defines a baseline set of mandatory algorithms together with explicit rules for algorithm identifiers, deprecation, and verifier behavior.

5.2 Hash algorithms

DAAP v1 verifiers SHALL implement SHA-256 as the minimum required hash algorithm. This baseline is derived from FIPS 180-4. Issuers MAY additionally include stronger digests such as SHA-384 or SHA-512 as parallel bindings, but SHALL include SHA-256 digests for all required bindings.

DAAP v1 SHALL use hashes in two primary locations:

1. Manifest hash - digest over canonical manifest bytes;
2. Content bindings - digest(s) over each referenced payload, or over defined byte ranges where range hashing is used.

5.2.1 Multi-hash rules

Where multiple hashes are present for the same object:
each digest SHALL carry an unambiguous algorithm identifier;
verifiers SHALL treat the binding as valid if at least one supported digest matches, provided baseline SHA-256 is present and matches; and
issuers SHALL NOT omit baseline SHA-256 even if stronger digests are included.

Snippet 5-1. Multi-hash binding set

```
from __future__ import annotations
import base64
import hashlib
from pathlib import Path
def compute_hash_set(path: Path) -> list[dict[str, str]]:
    data = path.read_bytes()
    return [
        {
            "alg": "sha-256",
            "digest_b64": base64.b64encode(hashlib.sha256(data).digest()).decode("ascii"),
        },
        {
            "alg": "sha-384",
            "digest_b64": base64.b64encode(hashlib.sha384(data).digest()).decode("ascii"),
        },
    ]
```

5.3 Signature algorithms

DAAP v1 verifiers SHALL implement digital-signature verification using algorithms conformant with FIPS 186-5. DAAP v1 SHALL specify at least one mandatory-to-implement signature profile for interoperability. The baseline profile in v1 is: ECDSA with P-256 and SHA-256.

Issuers MAY include alternative signatures, such as RSA or ECDSA P-384, but SHALL include the baseline profile for assets intended for broad interoperability. If the optional CBOR/COSE profile is used, signature

algorithm identifiers SHALL use COSE-standard labeling while remaining FIPS-conformant for the baseline set.

Snippet 5-2. Baseline signature envelope metadata

```
from __future__ import annotations
```

```
from typing import Any
```

```
def baseline_signature_envelope(signing_key_ref: str, signer_party_id: str) -> dict[str, Any]:
    return {
        "sig_id": "sig:001",
        "sig_type": "x509_ecdsa_p256_sha256",
        "signer_party_id": signer_party_id,
        "signing_key_ref": signing_key_ref,
        "signed_object": "manifest_hash_and_payload_bindings",
    }
```

This structure reflects the deep-dive example, which binds the signature not merely to a raw blob, but to the manifest hash and payload bindings as the integrity-bearing object.

5.4 Crypto-agility rules

```
DAAP v1 SHALL define explicit agility semantics to prevent silent verification divergence.
Specifically:
▯
DAAP SHALL define or adopt an algorithm identifier namespace for hashes and
signatures;
▯
for COSE-based signatures, DAAP SHALL use COSE algorithm identifiers;
▯
for JSON/JCS manifests, DAAP SHALL use a well-defined textual registry aligned with
IANA or COSE naming where feasible.
```

5.4.1 Deprecation and acceptance rules

DAAP v1 SHALL define verifier behavior under algorithm lifecycle change using four registry states: □ required (baseline) □ recommended (stronger alternatives) □ deprecated (discouraged; warning or policy soft-fail) □ prohibited (verification failure)

Verification policy SHALL expose at least the following classes: □ MUST-ACCEPT - baseline SHA-256 and baseline ECDSA P-256 / SHA-256, subject to trust validation; □ MAY-ACCEPT - additional supported algorithms not explicitly deprecated; □ MUST-REJECT - prohibited algorithms or algorithms outside the DAAP conformance envelope.

Snippet 5-3. Agility enforcement

```
from __future__ import annotations
```

```
MUST_ACCEPT_HASHES = {"sha-256"}
MAY_ACCEPT_HASHES = {"sha-384", "sha-512"}
PROHIBITED_HASHES = set()
```

```
MUST_ACCEPT_SIGS = {"x509_ecdsa_p256_sha256"}
MAY_ACCEPT_SIGS = {"x509_ecdsa_p384_sha384"}
PROHIBITED_SIGS = set()
```

```
def accept_algorithm(alg_id: str, domain: str) -> bool:
```

```
    if domain == "hash":
        if alg_id in PROHIBITED_HASHES:
            return False
        return alg_id in MUST_ACCEPT_HASHES or alg_id in MAY_ACCEPT_HASHES
```

```
    if domain == "sig":
        if alg_id in PROHIBITED_SIGS:
            return False
        return alg_id in MUST_ACCEPT_SIGS or alg_id in MAY_ACCEPT_SIGS
```

```
    raise ValueError("Unknown algorithm domain")
```

5.4.2 Verifier result codes

To support audit goals, DAAP v1 verifiers SHALL produce explicit verification outcomes:

- VALID_TRUSTED - signature and bindings verified; trust model satisfied
- INVALID - signature or binding mismatch; tampering or substitution detected
- UNVERIFIABLE - insufficient algorithm support, missing required binding, or missing trust chain
- REVOKED - trust credentials invalid under revocation policy TIMESTAMP_VALID / TIMESTAMP_INVALID / TIMESTAMP_ABSENT - evidentiary time-binding states, where timestamp policy is in scope.

This classification prevents implementations from treating “cannot verify” as “valid” by default.

Snippet 5-4. Verifier result classification

```
from __future__ import annotations
from enum import Enum
class VerificationResult(str, Enum):
    VALID_TRUSTED = "VALID_TRUSTED"
    INVALID = "INVALID"
    UNVERIFIABLE = "UNVERIFIABLE"
    REVOKED = "REVOKED"
```

```
class TimestampResult(str, Enum):
    TIMESTAMP_VALID = "TIMESTAMP_VALID"
    TIMESTAMP_INVALID = "TIMESTAMP_INVALID"
    TIMESTAMP_ABSENT = "TIMESTAMP_ABSENT"
```

```
def classify_verification(
    signature_ok: bool,
    bindings_ok: bool,
    trust_ok: bool,
    revoked: bool,
) -> VerificationResult:
    if revoked:
        return VerificationResult.REVOKED
    if not signature_ok or not bindings_ok:
        return VerificationResult.INVALID
    if not trust_ok:
        return VerificationResult.UNVERIFIABLE
    return VerificationResult.VALID_TRUSTED
```

5.5 Conformance implications

A conformant DAAP verifier SHALL:

- support SHA-256 for manifest and payload bindings;
- support ECDSA P-256 / SHA-256 for baseline signature verification;
- compute integrity inputs over canonicalized manifest bytes, not source serializer output;
- publish explicit acceptance/rejection behavior for algorithm classes; and
- emit non-ambiguous verifier result codes rather than silent success under missing support.

This matters because verifier divergence is not a usability defect; it is a trust-surface fracture. A protocol that allows two verifiers to accept different integrity states for the same asset does not provide portable verification.

5.6 Section conclusion

Sections 4 and 5 jointly establish the minimum cryptographic semantics required for DAAP to function as an asset-layer authority model. Section 4 fixes the claims surface and the canonical byte representation. Section 5 fixes the digest and signature baselines, the algorithm registry semantics, and the verifier result classes. With those constraints in place, subsequent sections can address trust anchors, revocation, timestamps, and disclosure tiers without ambiguity in the signed object or the verification envelope.

6. Trust Anchor Models and Verification Policies

Section 5 establishes cryptographic validity. Section 6 establishes trust semantics. DAAP v1 treats those as separable evaluation domains. A manifest store may be correctly signed yet operationally untrustworthy if the signing credential is unknown, outside an authorized supply chain, mis-issued, or no longer policy-valid. DAAP v1 therefore defines trust evaluation as a second-stage decision procedure layered on top of signature and binding verification. Cryptographic validity is necessary; it is not sufficient.

6.1 Normative requirement: trust evaluation is explicit, not implied

DAAP v1 SHALL require that every verifier evaluate signatures under a declared trust model. Trust SHALL NOT be inferred implicitly from signature presence alone. A conforming verifier SHALL distinguish at minimum: CRYPTO_VALID + TRUSTED CRYPTO_VALID + UNTRUSTED CRYPTO_INVALID UNVERIFIABLE (insufficient trust data to complete evaluation) This requirement prevents a common over-trust failure mode in which unknown signers are silently treated as valid merely because a signature verifies mathematically.

6.2 Trust Model A: WebPKI / X.509 chain validation

Model A uses conventional X.509 path validation against a configured trust-anchor set. Under this model, a verifier SHALL:

- validate the signer certificate chain to an accepted root or intermediate anchor;
- enforce RFC 5280 path-validation rules;
- enforce validity-period checks;

- enforce KeyUsage / ExtendedKeyUsage constraints where applicable; and
- apply any profile-specific certificate policy OIDs or name constraints defined by DAAP policy.

DAAP v1 SHOULD define a dedicated signing EKU or equivalent certificate policy OID so that generic transport certificates are not repurposed indiscriminately for asset signing. Operationally, Model A is appropriate where implementers prefer mature PKI tooling, enterprise CA governance, and standardized revocation pathways.

Snippet 6-1. Trust Model A evaluation boundary

```
from __future__ import annotations
from dataclasses import dataclass
from enum import Enum
class TrustResult(str, Enum):
    TRUSTED = "TRUSTED"
```

```
    UNTRUSTED = "UNTRUSTED"
    UNVERIFIABLE = "UNVERIFIABLE"
```

```
@dataclass
class CertificatePathResult:
    path_valid: bool
    eku_valid: bool
    time_valid: bool
```

```
def evaluate_trust_model_a(path_result: CertificatePathResult) -> TrustResult:
    """
    DAAP Model A: WebPKI / X.509 trust evaluation.
    Revocation is intentionally excluded here and applied in Section 7.
    """
    if not path_result.path_valid:
        return TrustResult.UNTRUSTED
    if not path_result.eku_valid:
        return TrustResult.UNTRUSTED
    if not path_result.time_valid:
        return TrustResult.UNTRUSTED
    return TrustResult.TRUSTED
```

This boundary is correct because Model A is defined as PKI-native trust by chain validation, not merely trust by key possession. Revocation is intentionally deferred to Section 7 because DAAP treats revocation as part of trusted verification rather than as an optional post-check.

6.3 Trust Model B: Consortium Trust List

Model B uses a consortium-managed trust list, conceptually aligned with C2PA trust-list semantics. In this model, a signer is trusted only if the signing certificate, or its issuing chain, appears on a published trust list recognized by the verifier. The trust list SHALL be versioned, retrievable, cacheable, and policy-governed. It MAY contain roots, intermediates, or end-entity certificates depending on deployment policy. The value of Model B is governance locality. It permits an industry-specific or consortium- specific closed-loop trust regime without requiring the entire ecosystem to rely exclusively on public WebPKI roots. It also creates an explicit governance surface for onboarding signers, constraining certificate issuance, and responding to compromise events.

Snippet 6-2. Trust Model B list evaluation

```
from __future__ import annotations
from dataclasses import dataclass
from enum import Enum
class TrustListResult(str, Enum):
    TRUSTED = "TRUSTED"
    UNTRUSTED = "UNTRUSTED"
    UNVERIFIABLE = "UNVERIFIABLE"
```

```
@dataclass
class TrustListState:
    version: str
    stale: bool
    issuer_fingerprints: set[str]
    end_entity_fingerprints: set[str]
```

```
def evaluate_trust_model_b(
    signer_fingerprint: str,
    issuer_fingerprint: str,
    trust_list: TrustListState | None,
) -> TrustListResult:
    """
    DAAP Model B: consortium trust-list evaluation.
    """
    if trust_list is None:
        return TrustListResult.UNVERIFIABLE
    if trust_list.stale:
        return TrustListResult.UNVERIFIABLE
```

```
    if signer_fingerprint in trust_list.end_entity_fingerprints:
        return TrustListResult.TRUSTED
    if issuer_fingerprint in trust_list.issuer_fingerprints:
        return TrustListResult.TRUSTED
    return TrustListResult.UNTRUSTED
```

This structure reflects the specification's trust-list model: trust is not inferred from a generic signature; it is established by membership in a currently active trust list subject to freshness and retrieval policy.

6.4 Verifier policy profiles

DAAP v1 SHALL publish verifier policy profiles describing how trust models are applied. At minimum, two policy classes SHALL exist: Policy P1 (Strict / Audit-grade): signature MUST validate and signer MUST chain to an accepted trust anchor; revocation MUST be checked per Section 7; missing trust data yields UNVERIFIABLE. Policy P2 (Permissive / Informational): signature validity MAY be reported even if trust cannot be established, but the result MUST be labeled explicitly as SIGNED-BUT- UNTRUSTED or equivalent, not VALID.

Snippet 6-3. Trust policy classifier

```
from __future__ import annotations
from enum import Enum
class CryptoResult(str, Enum):
    CRYPTO_VALID = "CRYPTO_VALID"
    CRYPTO_INVALID = "CRYPTO_INVALID"
```

```
class TrustOutcome(str, Enum):
    VALID_TRUSTED = "VALID_TRUSTED"
    SIGNED_BUT_UNTRUSTED = "SIGNED_BUT_UNTRUSTED"
    INVALID = "INVALID"
    UNVERIFIABLE = "UNVERIFIABLE"
```

```
def combine_crypto_and_trust(
    crypto_valid: bool,
    trust_status: str,
    policy: str,
) -> TrustOutcome:
    if not crypto_valid:
        return TrustOutcome.INVALID
```

```
    if policy == "P1":
        if trust_status == "TRUSTED":
            return TrustOutcome.VALID_TRUSTED
        if trust_status == "UNTRUSTED":
            return TrustOutcome.UNVERIFIABLE
        return TrustOutcome.UNVERIFIABLE
```

```
    if policy == "P2":
        if trust_status == "TRUSTED":
            return TrustOutcome.VALID_TRUSTED
        return TrustOutcome.SIGNED_BUT_UNTRUSTED
```

```
    raise ValueError("Unknown policy")
```

6.5 Trust-list retrieval, caching, and update cadence

Because trust lists are operationally time-sensitive, DAAP v1 SHALL define: a minimum trust-list refresh interval, a maximum cache staleness window, and explicit behavior when retrieval fails.

Verifiers MAY continue using a cached list only within a defined TTL. Once the staleness window is exceeded, the verifier SHALL return UNVERIFIABLE, not VALID_TRUSTED. This requirement exists because trust-list divergence is itself an audit risk.

7. Revocation Workflow and Verifier Behavior

Section 6 defines who may be trusted. Section 7 defines when a previously trusted credential ceases to be trustworthy. DAAP v1 treats revocation as part of the trusted verification envelope, not as an optional post-processing enhancement. A signature may remain cryptographically valid after the signer credential has been compromised, withdrawn, or invalidated. Strict verification therefore requires revocation evaluation.

7.1 Normative requirement: revocation is part of trusted verification

For X.509-based profiles, DAAP v1 SHALL define revocation behavior using established PKI standards: certificate / CRL processing and path validation per RFC 5280 online status checking per RFC 6960 (OCSP).

Verifiers that skip revocation checks MUST NOT return a trusted-valid result under strict policy. If revocation status cannot be evaluated within policy constraints, the verifier SHALL return UNVERIFIABLE or an equivalent trust-incomplete result, not VALID_TRUSTED.

7.2 Supported revocation mechanisms

DAAP v1 SHALL support the following revocation mechanisms for X.509-based trust:

1. OCSP - primary online status mechanism
2. CRLs - offline or batch status mechanism.

Issuers SHALL publish revocation information using standard certificate mechanisms, including Authority Information Access for OCSP responders and CRL Distribution Points for CRLs, consistent with RFC 5280.

7.3 Revocation policy profiles

7.3.1 Policy P1 (Strict / Audit-grade)

```
Under P1, verifiers SHALL:

perform RFC 5280 certificate-path validation;

perform revocation checking for each certificate required by policy, at minimum the end-
entity signing certificate;

use OCSP when available;

fall back to CRL when OCSP is unavailable and CRL data is available; and

return UNVERIFIABLE if neither mechanism can determine status within policy
constraints.
```

7.3.2 Policy P2 (Permissive / Informational)

Under P2, a verifier MAY report cryptographic validity even when revocation data cannot be obtained, but MUST surface that uncertainty explicitly and MUST NOT collapse it into a trusted-valid outcome.

Snippet 7-1. Revocation evaluation skeleton

```
from __future__ import annotations
```

```
from enum import Enum
```

```
class RevocationState(str, Enum):
    GOOD = "GOOD"
    REVOKED = "REVOKED"
    UNKNOWN = "UNKNOWN"
```

```
class RevocationOutcome(str, Enum):
    VALID_TRUSTED = "VALID_TRUSTED"
    REVOKED = "REVOKED"
    INVALID = "INVALID"
    UNVERIFIABLE = "UNVERIFIABLE"
```

```
def evaluate_revocation(
    signature_ok: bool,
```

```
    chain_ok: bool,
    ocsp_state: RevocationState | None,
    crl_state: RevocationState | None,
    policy: str,
) -> RevocationOutcome:
    if not signature_ok or not chain_ok:
        return RevocationOutcome.INVALID
```

```
    if policy == "P1":
        if ocsp_state == RevocationState.REVOKED or crl_state == RevocationState.REVOKED:
            return RevocationOutcome.REVOKED
        if ocsp_state == RevocationState.GOOD or crl_state == RevocationState.GOOD:
            return RevocationOutcome.VALID_TRUSTED
        return RevocationOutcome.UNVERIFIABLE
```

```
    if policy == "P2":
        if ocsp_state == RevocationState.REVOKED or crl_state == RevocationState.REVOKED:
            return RevocationOutcome.REVOKED
        return RevocationOutcome.UNVERIFIABLE
```

```
    raise ValueError("Unknown policy")
```

7.4 Caching and retrieval failure behavior

DAAP v1 SHALL define explicit caching and fallback semantics for revocation data. These behaviors MUST be specified because revocation endpoints are operational dependencies and denial-of-service surfaces. Cache usage MAY be permitted within policy-defined freshness windows; stale revocation state beyond those windows SHALL result in UNVERIFIABLE, not silent success.

7.5 Failure modes and required outcomes

DAAP v1 SHALL define explicit, non-ambiguous verifier outcomes for revocation evaluation:

- VALID_TRUSTED - signature valid, chain valid, revocation checked and not revoked
- REVOKED - chain indicates revoked at any required point
- INVALID - signature mismatch or chain invalid
- UNVERIFIABLE - insufficient data to establish revocation/trust state.

A conforming verifier SHALL NOT downgrade “revocation unknown” to “valid trusted.” Any policy that permits continued operation under uncertainty MUST surface that uncertainty explicitly in result codes and logs.

7.6 Threat-model implication

The revocation model exists to address concrete adversarial conditions already identified in the deep-dive threat appendix: stolen signing keys, trust-anchor compromise, mis-issuance, and denial-of-service against verification dependencies. Revocation is therefore not ancillary; it is the mechanism by which trust can be withdrawn after cryptographic issuance has already occurred.

8. Trusted Timestamps and Evidentiary Events

Section 7 determines whether a signer is still trusted. Section 8 determines when a given manifest state can be shown to have existed. DAAP v1 uses trusted timestamps to attach evidentiary time to manifest state, acceptance state, or distribution state. This is necessary for audit-grade chain-of-custody reasoning, compromise-window analysis, and replay/equivocation detection.

8.1 Normative requirement: timestamping for evidentiary/audit goals

DAAP v1 supports two equivalent timestamping mechanisms:

- Option T: RFC 3161 Time-Stamp Tokens issued by a Time-Stamping Authority (TSA)
- Option L: transparency-log receipts and inclusion proofs in a CT-style architecture.

DAAP v1 defines RFC 3161 as the normative baseline because it is widely implemented and aligns naturally with X.509-based trust models already used for signatures. Transparency logs are optional but permitted where a consortium or public append-only log is preferred over TSA infrastructure.

8.2 Required timestamped events

At minimum, the events[] list SHALL include a creation event. Under an audit-grade profile, the manifest SHALL include timestamp evidence for:

- creation acceptance / authorization (if the policy includes an authorization event) distribution / publication (if policy requires distribution evidence).

This requirement exists because DAAP does not merely verify signatures; it verifies claims about asset state and sequence.

8.3 RFC 3161 profile (Option T)

Under Option T: □ timestamp evidence SHALL be RFC 3161-compliant;

- the timestamp request SHALL include a hash imprint of the target artifact using an explicitly identified collision-resistant hash function;
- verifiers SHALL validate the timestamp token signature; and
- verifiers SHALL confirm that the included imprint matches the locally computed digest of the target artifact.

Crucially, timestamps SHALL bind to a stable imprint, i.e., canonical manifest bytes or a defined manifest digest, not to mutable container bytes. This prevents timestamp validity from depending on transport-layer rewrites.

Snippet 8-1. RFC 3161 timestamp request boundary

```
from __future__ import annotations
import hashlib
from dataclasses import dataclass
@dataclass
class TimestampRequest:
    alg: str
    imprint_hex: str
def build_timestamp_request(canonical_manifest_bytes: bytes) -> TimestampRequest:
    digest = hashlib.sha256(canonical_manifest_bytes).hexdigest()
    return TimestampRequest(
```

```
        alg="sha-256",
        imprint_hex=digest,
    )
```

8.3.1 TSA trust evaluation

TSA trust SHALL be evaluated using the same two trust models defined in Section 6:

- Model A: WebPKI / X.509 validation
- Model B: consortium trust list, where TSAs are explicitly listed.

This keeps time evidence inside the same overall trust-governance framework as signing evidence.

8.4 Transparency-log profile (Option L)

Option L provides an equivalent evidentiary function through an append-only transparency log design. A log issues a signed timestamp receipt upon submission and later provides a proof of inclusion. DAAP v1 MAY define a consortium log under a published trust policy, provided the log supports:

- signed submission receipts,
- inclusion proofs verifiable against published log state, and
- monitoring sufficient to detect log misbehavior.

This profile is particularly useful for detecting equivocation or “two truths” conditions in which inconsistent manifest histories are emitted for the same logical asset.

Snippet 8-2. Timestamp result classification

```
from __future__ import annotations
from enum import Enum
class TimestampResult(str, Enum):
    TIME_VALID = "TIME_VALID"
    TIME_MISSING = "TIME_MISSING"
    TIME_INVALID = "TIME_INVALID"
    TIME_UNVERIFIABLE = "TIME_UNVERIFIABLE"
def classify_timestamp(
    token_present: bool,
    signature_ok: bool,
    imprint_matches: bool,
    trust_ok: bool,
) -> TimestampResult:
    if not token_present:
        return TimestampResult.TIME_MISSING
    if not signature_ok or not imprint_matches:
        return TimestampResult.TIME_INVALID
    if not trust_ok:
        return TimestampResult.TIME_UNVERIFIABLE
    return TimestampResult.TIME_VALID
```

8.5 Verifier behavior, caching, and failure semantics

DAAP v1 defines the following timestamp result classes:

- TIME_VALID - token or receipt verifies and binds to the correct artifact digest
- TIME_MISSING - required timestamp absent under the active policy
- TIME_INVALID - token invalid, mismatched imprint, or proof failure
- TIME_UNVERIFIABLE - timestamp cannot be validated due to missing trust material, expired trust chain, or inability to obtain required inclusion proof.

RFC 3161 tokens MAY be cached indefinitely as evidence objects because they are detached signed assertions. For transparency logs, verifiers MAY cache signed receipts, but SHALL enforce a policy for delayed proof retrieval; if inclusion proof remains unobtainable after the allowed window, the result SHALL be TIME_UNVERIFIABLE. Under strict Policy P1, TIME_MISSING, TIME_INVALID, or TIME_UNVERIFIABLE SHALL prevent the asset from being treated as audit-grade verified, even if the underlying signatures validate.

8.6 Section conclusion

Sections 6–8 complete the DAAP verification stack. Section 6 defines who may be trusted. Section 7 defines how trust is withdrawn or becomes indeterminate. Section 8 defines when a manifest state can be established evidentially. Together they convert the manifest store from a signed metadata object into an audit-grade claims surface capable of expressing trust, revocation, and temporal evidence explicitly rather than by implication. The strongest next move is Section 9 and Section 10 together:

9. Synopsis Schema: Portable Production Graph Model and Explicit Out-of-Scope Boundary

Section 9 resolves the seventh reconstruction constraint: what structural production information DAAP carries, and what it explicitly does not carry. DAAP v1 does not attempt to serialize a DAW session. It defines a portable abstract synopsis graph whose function is limited to structural intelligibility, lineage anchoring, and payload-to-render traceability under ordinary distribution, archival, and verification conditions. The synopsis is therefore a graph- bounded authority adjunct, not an execution substrate.

9.1 Normative purpose: graph synopsis, not session recreation

Modern production environments are graph-shaped. Tracks, buses, processing chains, renders, alternates, and packaged payloads are related by dependency edges rather than by simple file lists. DAAP v1 captures only the minimum graph necessary to preserve:

1. render-to-payload mapping,
2. derivation and version lineage, and
3. stable structural references for attribution and provenance reasoning.

DAAP v1 SHALL NOT attempt to recreate an editable DAW session, encode executable DSP state, or duplicate the full studio metadata capture domain already addressed by DDEX RIN. RIN remains the upstream machine-to-machine capture standard for contributors, roles, technical session metadata, and recording-process detail; DAAP’s synopsis complements RIN by preserving a smaller, asset-scoped structural graph at distribution time.

9.2 Normative model: typed nodes, typed edges, stable internal identifiers

DAAP v1 SHALL define a synopsis schema consisting of typed nodes, typed edges, and stable internal identifiers. At minimum, the following node classes SHALL exist:

- Session - abstract root context
- Track - logical track identity, not audio bytes
- Bus / Group - routing aggregation
- ProcessorChain - logical processing-chain identity, without requiring parameter disclosure
- Render - produced audio artifact (e.g., master, stem, alt_mix)
- Payload - packaged or externally referenced bytes bound via Section 3 content bindings
- PartyRef - contributor/tool/vendor references by identifier only. At minimum, the following edge classes SHALL exist:
- routes_to - Track/Bus routing
- feeds - Track/Bus contribution to a Render
- processed_by - Track/Bus → ProcessorChain
- derives_from - Render → Render (version / derivation relation)
- binds_to - Render → Payload.

All nodes and edges SHALL carry stable, asset-scoped identifiers. These identifiers are not required to be globally resolvable. Global references to people, works, recordings, vendors, or registries are carried via PartyRef objects and optional mapping references.

9.3 Canonical synopsis fields

DAAP v1 SHALL require the following minimal synopsis fields: synopsis.version nodes[] with id, type, and optional label, refs[] edges[] with id, type, from, to renders[] or equivalent render-node form, including:

- render role (master, stem, alt, etc.),
- optional non-authoritative format hints,
- explicit binding to one or more payload identifiers.

The minimum conformance objective is not full production replay. It is sufficient graph structure to let a verifier reason about:

- which payload corresponds to which render role,
- which renders derive from which earlier renders, and
- which logical components contributed to an exported artifact.

Snippet 9-1. Minimal synopsis object

```
from __future__ import annotations
from typing import Any
```

```
def minimal_synopsis() -> dict[str, Any]:
    return {
        "synopsis": {
            "version": "1.0",
            "nodes": [],
            "edges": [],
        }
```

```
    }
Snippet 9-2. Adding typed nodes and edges
from __future__ import annotations

from typing import Any

def add_node(synopsis: dict[str, Any], node_id: str, node_type: str, label: str | None = None)
-> None:
    synopsis["synopsis"]["nodes"].append(
        {
            "id": node_id,
            "type": node_type,
            "label": label,
        }
    )

def add_edge(synopsis: dict[str, Any], edge_id: str, edge_type: str, src: str, dst: str) ->
None:
    synopsis["synopsis"]["edges"].append(
        {
            "id": edge_id,
            "type": edge_type,
            "from": src,
            "to": dst,
        }
    )
```

Snippet 9-3. Render-to-payload binding example

```
from __future__ import annotations
from typing import Any
def bind_render_to_payload(synopsis: dict[str, Any], render_id: str, payload_id: str) -> None:
    add_edge(
        synopsis=synopsis,
        edge_id=f"edge:{render_id}:binds_to:{payload_id}",
        edge_type="binds_to",
        src=render_id,
        dst=payload_id,
    )
```

These examples are intentionally austere. The implementation objective is stable structural intelligibility, not DAW execution-state serialization.

9.4 Relationship to DDEX RIN: complement, not duplication

DAAP v1 SHALL support RIN alignment without re-encoding full RIN inside the DAAP manifest store. Specifically, DAAP MAY:

- include a RIN_reference pointer (URI or bundle-local artifact reference), and/or
- link RIN-derived session or contributor identifiers through PartyRef objects. DAAP v1 SHALL NOT require implementers to embed full RIN messages inside the manifest store. This preserves separation of concerns:
  - RIN - studio/session metadata capture and communication
  - ERN/DSR - delivery and reporting
  - MEAD - DSP-oriented enrichment
  - DAAP - verifiable asset object with portable structural synopsis and integrity.

9.5 Explicit out-of-scope boundary

To prevent DAAP from collapsing into a DAW format or a studio metadata monolith, the synopsis schema SHALL declare the following out of scope:

1. executable DSP state and plugin parameters;
2. DAW-specific edit constructs (automation curves, clip boundaries, comp lanes, playlists);
3. human-facing narrative credit text beyond structured contributor references;
4. rights adjudication logic and split computation;
5. distribution reporting schemas themselves.

This boundary is required for portability. DAAP is intended to preserve asset-state structure, not tool-state replay.

9.6 Conformance implication

For Tier 1+ assets, DAAP v1 SHALL include at least enough synopsis structure to map deliverable renders to payloads. For Tier 2+ assets, DAAP SHOULD include the portable abstract graph. DAAP SHALL NOT require synopsis disclosure beyond the declared disclosure tier.

9.7 Section conclusion

Section 9 constrains the synopsis to the minimum graph required for render identity, lineage, and structural provenance. It preserves the distinction between distribution-grade asset structure and studio-grade metadata capture, thereby allowing DAAP to remain asset-scoped, verifier- friendly, and interoperable with RIN rather than competitive with it.

10. Redaction and Privacy While Preserving Verifiability

Section 10 resolves the eighth reconstruction constraint: how DAAP supports privacy and redaction without breaking verification. DAAP v1 treats privacy as a first-class conformance domain. The protocol therefore does not assume that all parties receive the same manifest view. Instead, it defines disclosure tiers and redaction primitives such that selective disclosure remains cryptographically coherent, policy-declared, and verifier-visible.

10.1 Normative requirement: privacy is first-class

Music production metadata often contains commercially sensitive or personally sensitive information: embargoed contributors, internal workflow IDs, proprietary chain labels, session notes, studio identifiers, or negotiated credits. In ordinary metadata systems, removing such fields breaks signatures or silently destroys verification. DAAP v1 prohibits that failure mode by requiring privacy transformations to be explicit, typed, and verification-preserving. DAAP v1 therefore imposes three top-level constraints:

1. verifiability SHALL survive across disclosure levels;
2. redaction actions SHALL themselves be representable as verifiable transformations;
3. selective disclosure SHALL NOT allow silent manipulation that appears authentic.

10.2 Disclosure tiers

DAAP v1 SHALL define at least four disclosure tiers:

- Tier 0 - Public: minimal identity and provenance assertions intended for broad circulation
- Tier 1 - Industry: contributor IDs, rights references, non-sensitive tool identifiers
- Tier 2 - Partner: richer synopsis structure and partner-visible structural references, subject to redaction
- Tier 3 - Private/Audit: full internal manifest for rights-holder, legal, or audit workflows.

Each manifest SHALL declare its disclosure tier, and verifiers SHALL treat the tier as part of the claims context rather than as out-of-band metadata.

Snippet 10-1. Disclosure declaration

```
from __future__ import annotations
from typing import Any
def apply_disclosure_tier(manifest: dict[str, Any], tier: int) -> None:
    manifest.setdefault("disclosure", {})
    manifest["disclosure"]["tier"] = tier
    manifest["disclosure"].setdefault("redaction_summary", [])
```

10.3 Redaction primitives

DAAP v1 SHALL support three privacy transformation classes.

10.3.1 Omission

A field MAY be omitted only if: □ the schema marks it optional, and □ the manifest declares the omission reason (e.g., redacted_for_privacy, contractual_confidentiality, security_sensitivity). Omission is valid only if the resulting manifest still satisfies the required field set for its declared tier.

Snippet 10-2. Omission with declared reason

```
from __future__ import annotations
from typing import Any
def omit_field(manifest: dict[str, Any], field_path: str, reason: str) -> None:
    manifest["disclosure"]["redaction_summary"].append(
        {
            "mode": "omit",
            "field_path": field_path,
            "reason": reason,
        }
    )
```

10.3.2 Hash commitment

For fields whose values must remain private but whose existence or stability should remain verifiable, DAAP v1 SHALL permit hash commitments. In this mode: □ the plaintext value is replaced by hash(value) plus algorithm identifier; □ the verifier can confirm cross-version stability without accessing the plaintext. This primitive is appropriate for embargoed contributor names, internal legal identifiers, proprietary chain labels, or internal workflow IDs.

Snippet 10-3. Hash commitment

```
from __future__ import annotations
import base64
import hashlib
```

```
def hash_commit(value: str, alg: str = "sha-256") -> dict[str, str]:
    if alg != "sha-256":
        raise ValueError("Only sha-256 shown in this baseline example")
    digest = hashlib.sha256(value.encode("utf-8")).digest()
    return {
        "commitment_alg": alg,
        "commitment_b64": base64.b64encode(digest).decode("ascii"),
    }
```

10.3.3 Encryption

For fields that must remain confidential but may require later authorized disclosure, DAAP v1 MAY support subtree encryption. Encrypted field blocks MUST carry:

- encryption algorithm identifiers,
- key-wrapping / recipient-mechanism references, and
- integrity binding such that ciphertext is itself signed and cannot be swapped. DAAP v1 SHALL NOT require verifiers to decrypt in order to validate the asset. Validation attaches to the manifest store, content bindings, and signed ciphertext presence, not to plaintext access.

10.4 Redaction as a verifiable transformation

A redacted manifest intended to circulate as an authentic view of an underlying private manifest SHALL be represented as one of:

1. a new manifest version with derives_from linkage and explicit redaction annotations; and/or
2. a countersigned derived manifest in which an authorized entity signs the redacted representation. DAAP v1 therefore requires a redaction record indicating:
    - which fields were omitted, committed, or encrypted;
    - why;
    - by whom;
    - and, where strict policy applies, at what time.

Snippet 10-4. Redaction record

```
from __future__ import annotations
from typing import Any
def add_redaction_record(
    manifest: dict[str, Any],
    field_path: str,
    mode: str,
    reason: str,
    signer_id: str,
```

```
) -> None:
    manifest["disclosure"]["redaction_summary"].append(
        {
            "field_path": field_path,
            "mode": mode,            # omit | commit | encrypt
            "reason": reason,
            "by": signer_id,
```

```
        }
    )
```

10.5 Verifier behavior under redaction

DAAP v1 verifiers SHALL evaluate:

1. cryptographic validity of the manifest store and content bindings;
2. completeness relative to the required field set for the declared tier;
3. redaction integrity - whether redactions are declared, derivationally linked, and covered

by signatures. Required outcomes:

- if required fields for the declared tier are missing without valid omission semantics, the verifier SHALL return UNVERIFIABLE or policy failure;
- if content bindings validate but redaction rules are violated by silent removal, the verifier SHALL return INVALID or POLICY_FAIL according to taxonomy;
- if hash commitments are present, the verifier SHALL validate commitment format, algorithm acceptance, and signature coverage.

Snippet 10-5. Redaction-aware verifier result

```
from __future__ import annotations
from enum import Enum
class RedactionResult(str, Enum):
    VALID_DISCLOSED = "VALID_DISCLOSED"
    UNVERIFIABLE = "UNVERIFIABLE"
    INVALID = "INVALID"
    POLICY_FAIL = "POLICY_FAIL"
def classify_redaction_state(
    cryptographically_valid: bool,
    required_fields_present: bool,
    declared_redactions_valid: bool,
) -> RedactionResult:
    if not cryptographically_valid:
        return RedactionResult.INVALID
    if not required_fields_present:
        return RedactionResult.UNVERIFIABLE
    if not declared_redactions_valid:
        return RedactionResult.POLICY_FAIL
    return RedactionResult.VALID_DISCLOSED
```

10.6 Threat-model implication

Redaction support is not only a privacy feature. It is a control against a distinct trust-surface failure: silent omission masquerading as authenticity. Without redaction semantics, any downstream party can strip fields and redistribute an apparently valid but semantically degraded representation. DAAP prevents that by requiring tier declaration, redaction declaration, derivational traceability, and signature coverage over commitments or ciphertext.

10.7 Section conclusion

Sections 9 and 10 complete the structural and disclosure surfaces of the DAAP authority model. Section 9 defines the minimum portable graph required to preserve render identity and derivation without collapsing into a DAW format. Section 10 defines how that graph and the rest of the manifest can be disclosed, committed, or encrypted without sacrificing verification. Together, they

give DAAP an authority object that is not only signable and trust-evaluable, but also structurally portable and privacy-governable.

Appendix A.

DAAP v1 Manifest Schema (Normative) + Example Manifest

This appendix defines the normative DAAP v1 manifest data model and the minimum required fields for conformance. The canonical encoding is JSON, with JCS canonicalization required for signing. An optional CBOR/COSE profile is permitted. Hash and signature baselines are defined by the protocol baseline established earlier in Sections 4 and 5.

A.1 Normative object model (top-level) A DAAP v1 Manifest Store SHALL contain exactly one Active Manifest object conforming to the following top-level structure:

- daap_version -protocol version identifier
- schema_uri -stable schema identifier
- asset -asset identity block
- disclosure -tier and redaction semantics
- integrity -manifest hash, payload bindings, signature references
- events -provenance / audit events
- synopsis -portable production graph synopsis, or explicit null/empty synopsis for Tier 0
- mappings -optional export mappings to external systems
- extensions -optional explicitly namespaced extension object.

A.2 Normative field definitions A.2.1 asset (identity block) Required

- asset_id (string): stable DAAP asset identifier
- asset_type (string): e.g. recording_asset, session_derived_asset
- asset_version (string): semantic or monotonic version
- parent (object|null): required if not the first version If parent is present
- parent.asset_id
- parent.asset_version
- parent.relationship (derives_from, revision_of, etc.) Recommended
- title
- primary_artist
- identifiers: o isrc o iswc o upc. A.2.2 disclosure (tier + redaction) Required
- tier: one of o T0_PUBLIC o T1_INDUSTRY o T2_PARTNER o T3_PRIVATE

• redaction_summary: MAY be empty, MUST enumerate redactions if present Each redaction_summary[] entry SHALL contain:

• path

• method = omitted | hash_commitment | encrypted

• reason = controlled vocabulary, e.g. privacy, contractual_confidentiality, security_sensitivity.

A.2.3 parties (contributors, rightsholders, tool vendors) Optional at Tier 0; required at Tier 1+. Each PartyRef SHOULD support:

• party_id

• party_type = person | org | tool_vendor | publisher | label

• name

• identifiers

• roles If name is omitted for privacy, a commitment form MAY be supplied instead.

A.2.4 tools (tool identifiers without execution state) Optional at Tier 0; recommended Tier 1+; required Tier 2+ where tool attribution is claimed. Each tool object SHOULD support:

• tool_id

• tool_type = plugin | instrument | hardware_ref | software_service

• vendor_party_id

• name

• version

• external_ids Explicitly out of scope: executable DSP state, parameters, presets, and DAW recreation semantics.

A.2.5 synopsis (portable production graph) Tier rules:

• Tier 0: MAY be absent or level = "none"

• Tier 1: SHOULD include render-to-payload bindings

• Tier 2+: SHOULD include abstract graph nodes and edges Required when present:

• level = none | render_binding | abstract_graph

• nodes[]

• edges[] nodes[] fields:

• id

• type

- label (optional)
- refs[] (optional) edges[] fields:
- id
- type = routes_to | feeds | processed_by | derives_from | binds_to
- from
- to.

A.2.6 payloads (distributable audio objects) Required Tier 0+. Each payload SHALL contain:

- payload_id
- role = master | stem | alt_mix | preview | other
- media_type
- locator
- hashes[] Each hashes[] entry SHALL contain:
- alg -MUST include sha-256
- digest_b64

.

A.2.7 integrity (signing and verification pointers) Required.

- canonicalization.format = "json"
- canonicalization.scheme = "RFC8785_JCS"
- manifest_hash.alg = "sha-256" minimum
- manifest_hash.digest_b64
- signatures[] Each signature entry SHOULD contain:
- sig_id
- sig_type (baseline: x509_ecdsa_p256_sha256)
- signer_party_id
- signed_object
- timestamp (optional, RFC 3161 token reference or embedded object).

A.2.8 events (provenance / audit events) Required Tier 0+.

- events[] MUST include at least one creation event Each event SHOULD contain:
- event_id
- type = creation | acceptance | distribution | derivation

• actor_party_id

• time

• evidence: o rfc3161_tst_b64 or tst_ref o imprint_alg o imprint_digest_b64 If strict audit policy is claimed, required events MUST carry RFC 3161 evidence.

If strict audit policy is claimed, required events MUST carry RFC 3161 evidence.

A.3 Example DAAP v1 Active Manifest (Tier 1 “Industry”)

```
{
  "daap_version": "1.0",
  "schema_uri": "https://daap.example.org/schema/daap-manifest-1.0",
  "asset": {
    "asset_id": "daap:uuid:7f3d8a3e-2a9a-4f1d-9d7b-9c6b0f9d2a10",
    "asset_type": "session_derived_asset",
    "asset_version": "1.0.0",
    "title": "Dont Play That Song for Me",
    "primary_artist": "Aretha Franklin",
    "identifiers": {
      "isrc": "USAT2XXXXX01",
      "iswc": "T-123.456.789-0",
      "upc": "012345678901"
    },
    "parent": null
  },
  "disclosure": {
    "tier": "T1_INDUSTRY",
    "redaction_summary": []
  },
  "parties": [
    {
      "party_id": "party:label:001",
      "party_type": "org",
      "name": "Example Label LLC",
      "roles": ["label", "rights_holder"]
    },
    {
      "party_id": "party:artist:001",
      "party_type": "person",
      "name": "Aretha Franklin",
      "roles": ["artist"]
    },
    {
      "party_id": "party:vendor:ni",
      "party_type": "tool_vendor",
      "name": "Native Instruments",
      "roles": ["tool_vendor"]
    }
  ],
  "tools": [
    {
      "tool_id": "tool:plugin:001",
      "tool_type": "plugin",
      "vendor_party_id": "party:vendor:ni",
      "name": "Example Compressor",
      "version": "3.2.1"
    }
  ],
  "payloads": [
    {
      "payload_id": "payload:master:001",
      "role": "master",
```

```
      "media_type": "audio/wav",
      "locator": "bundle:/payloads/audio/master.wav",
      "hashes": [
        {
          "alg": "sha-256",
          "digest_b64": "bW9ja19kaWdlc3RfYmFzZTY0X2V4YW1wbGU="
        }
```

```
      ]
    },
    {
      "payload_id": "payload:stem:kick:001",
      "role": "stem",
      "media_type": "audio/wav",
      "locator": "bundle:/payloads/audio/stems/kick.wav",
      "hashes": [
        {
          "alg": "sha-256",
          "digest_b64": "bW9ja19kaWdlc3Rfa2ljaw=="
        }
      ]
    }
  ],
  "synopsis": {
    "level": "render_binding",
    "nodes": [
      { "id": "n:render:master", "type": "Render", "label": "Master Render" },
      { "id": "n:payload:master", "type": "Payload", "label": "master.wav" },
      { "id": "n:render:kick", "type": "Render", "label": "Kick Stem Render" },
      { "id": "n:payload:kick", "type": "Payload", "label": "kick.wav" }
    ],
    "edges": [
      { "id": "e:bind:master", "type": "binds_to", "from": "n:render:master", "to":
"n:payload:master" },
      { "id": "e:bind:kick", "type": "binds_to", "from": "n:render:kick", "to":
"n:payload:kick" }
    ]
  },
  "integrity": {
    "canonicalization": { "format": "json", "scheme": "RFC8785_JCS" },
    "manifest_hash": { "alg": "sha-256", "digest_b64": "bW9ja19tYW5pZmVzdF9oYXNo" },
    "signatures": [
      {
        "sig_id": "sig:001",
        "sig_type": "x509_ecdsa_p256_sha256",
        "signer_party_id": "party:label:001",
        "signed_object": "manifest_hash_and_payload_bindings",
        "timestamp": {
          "rfc3161_tst_b64": "bW9ja190aW1lc3RhbXBfdG9rZW4="
        }
      }
    ]
  },
  "events": [
    {
      "event_id": "evt:create:001",
      "type": "creation",
      "actor_party_id": "party:label:001",
      "time": "2026-02-18T00:00:00Z",
      "evidence": {
        "rfc3161_tst_b64": "bW9ja190aW1lc3RhbXBfdG9rZW4=",
        "imprint_alg": "sha-256",
        "imprint_digest_b64": "bW9ja19pbXByaW50"
      }
```

```
    }
  ],
  "extensions": {}
}
```

The example is intentionally minimal. Digest fields are placeholders; real assets MUST compute SHA-256 over canonical manifest bytes and payload bytes. The example uses Tier 1 plus a minimal render-binding synopsis; Tier 2+ would extend that into the abstract graph form.

Appendix B.

Conformance Checklist (What "DAAP-Compliant" Means)

This appendix defines the verifier-oriented conformance criteria for DAAP v1. Conformance is evaluated at three levels:

- B.1 Packaging Conformance
- B.2 Manifest Conformance
- B.3 Verification Conformance.

B.1 Packaging Conformance

B.1.1 Normative container

- ☐ The asset SHALL implement the DAAP Manifest Store as the authoritative location for the active manifest.
- ☐ The container SHALL include all referenced payloads, or stable external references under the optional bridge profile.

B.1.2 Payload reference integrity

- ☐ Every referenced payload SHALL contain at least one content-binding hash entry.
- ☐ Every required payload hash set SHALL include SHA-256.
- ☐ Payload locators SHALL be deterministic and non-ambiguous.

B.1.3 Manifest store determinism

- ☐ There SHALL be exactly one active manifest selected deterministically.
- ☐ Historical manifests, if present, SHALL preserve ordering/selection semantics without ambiguity.

B.2 Manifest Conformance

B.2.1 Required top-level fields

- ☐ daap_version SHALL be present and equal "1.0".
- ☐ schema_uri SHALL be present.
- ☐ asset, disclosure, payloads, integrity, and events SHALL be present.
- ☐ synopsis SHALL be present for Tier 1+ assets. B.2.2 Disclosure tier declaration
- ☐ disclosure.tier SHALL be one of T0_PUBLIC, T1_INDUSTRY, T2_PARTNER, T3_PRIVATE.
- ☐ Any privacy transformation SHALL be declared in disclosure.redaction_summary.

B.2.3 Identifier constraints

- ☐ Identifiers SHALL be typed deterministically.
- ☐ Duplicate keys SHALL NOT appear.
- ☐ Ambiguous numeric encodings SHALL NOT be used where canonicalization would be unstable.

B.3 Canonicalization Conformance

B.3.1 Canonical JSON baseline

- ☐ JSON manifests SHALL be canonicalized using RFC 8785 JCS.
- ☐ Hash and signature inputs SHALL be computed over canonicalized UTF-8 bytes.

B.3.2 Optional CBOR profile

- ☐ If CBOR is used, the asset SHALL declare the CBOR/COSE profile.
- ☐ CBOR profile manifests SHALL remain semantically equivalent to the normative JSON model.

B.4 Cryptographic Baseline Conformance

B.4.1 Hash baseline

- ☐ Verifiers SHALL support SHA-256.
- ☐ Issuers SHALL include SHA-256 for manifest hash and all required payload bindings.

B.4.2 Signature baseline

- ☐ Verifiers SHALL support the mandatory signature suite: ECDSA P-256 with SHA-256.
- ☐ Issuers targeting broad interoperability SHALL include a baseline signature using that suite.

B.4.3 Algorithm agility

- ☐ Additional algorithms SHALL be explicitly labeled.
- ☐ Prohibited algorithms SHALL be rejected.

B.5 Trust Model Conformance

- ☐ The manifest store SHALL declare which trust model(s) apply.
- ☐ Model A verifiers SHALL validate certificate chains per RFC 5280.
- ☐ Model B verifiers SHALL evaluate signer trust against the consortium trust list.
- ☐ A verifier SHALL NOT return VALID_TRUSTED unless cryptographic verification and trust evaluation both pass.

B.6 Revocation Conformance

- ☐ Under strict policy, verifiers SHALL perform revocation checking using OCSP and/or CRLs.
- ☐ If revocation status cannot be determined under policy constraints, verifiers SHALL return UNVERIFIABLE.
- ☐ If status is revoked, verifiers SHALL return REVOKED.

B.7 Timestamp / Evidentiary Conformance

- ☐ events[] SHALL include at least a creation event.
- ☐ For audit-grade profile, the manifest SHALL include trusted timestamp evidence for creation, acceptance (if present), and distribution (if present).
- ☐ If Option T is used, timestamp evidence SHALL be RFC 3161 compliant.
- ☐ Timestamps SHALL bind to a stable imprint, not mutable container bytes.
- ☐ Missing or invalid required timestamps SHALL prevent VALID_TRUSTED under strict policy.

B.8 Synopsis Schema Conformance

- ☐ Tier 1+ assets SHALL include at least render→payload binding edges sufficient to map deliverables to payloads.
- ☐ Tier 2+ assets SHOULD include a portable abstract graph with stable internal IDs.
- ☐ The synopsis SHALL NOT include DAW-specific edit state or executable DSP parameters.

B.9 Redaction / Privacy Conformance

- ☐ Any privacy transformation SHALL be declared via disclosure.redaction_summary.
- ☐ A redacted manifest intended to represent an authentic derived view SHALL be represented as: o a new version with derives_from linkage and/or o a countersigned derived manifest.
- ☐ Hash commitments and encrypted blocks SHALL remain covered by signature and integrity checks.

B.10 Verifier Output Conformance

A conforming DAAP verifier SHALL produce explicit, non-ambiguous outcomes, at minimum:

- VALID_TRUSTED
- INVALID
- UNVERIFIABLE
- REVOKED And where timestamps are used:
- TIME_VALID

- TIME_MISSING
- TIME_INVALID
- TIME_UNVERIFIABLE.

.

Appendix C.

Mapping Tables (DAAP → DDEX ERN/DSR; DAAP → BWF/iXML where applicable)

This appendix is implementation-neutral. It does not attempt to reproduce proprietary XSD field names in full. Instead, it provides mapping intent: which DAAP concepts correspond to which external standards surfaces.

C.1 DAAP → DDEX ERN (Release Delivery)

DAAP v1 field (concept)	ERN surface (concept)	Mapping note
asset.asset_id, asset.asset_version	Message-level internal references / resource references	DAAP IDs function as upstream authority IDs; ERN still requires its own message references.
asset.identifiers.isrc	Sound-recording resource identifier	ISRC remains the canonical recording identifier for DSP delivery where available.
asset.identifiers.upc	Release/product identifier	Used at release/product commerce level.
payloads[]	Resource list within release	ERN communicates release-contained recording resources.
events[] of type distribution	Release-delivery publication semantics / availability windows	DAAP distribution event can function as an audit anchor for the delivered state.
parties[]	Party/role attribution	ERN can carry a subset of DAAP's richer contributor graph.
tools[]	No direct analogue	Tool attribution generally remains DAAP-native.
synopsis	No direct analogue	Synopsis remains DAAP-native for verification/portability, not standard DSP ingest.
integrity	No ERN analogue	ERN is not a cryptographic provenance envelope.

Implementation note: treat ERN as the delivery message and DAAP as the authoritative asset object. ERN may include DAAP identifiers or pointers, but ERN remains a distribution interface, not the canonical provenance container.

C.2 DAAP → DDEX DSR (Sales and Usage Reporting)

DAAP v1 field (concept)	DSR surface (concept)	Mapping note
asset.identifiers.isrc	Recording identifier in usage lines	Primary linkage key when present and consistent.
asset.identifiers.upc	Release/product identifier	Used for release-level aggregation where applicable.
asset.asset_id	Additional/private identifier	Improves deterministic linking across catalog, delivery, and reporting layers.
events[] (distribution, acceptance)	Reporting-period alignment	DAAP events help anchor which version was distributed.
integrity	No DSR analogue	DSR is reporting; DAAP integrity supports chain-of-custody evidence outside DSR.
parties[], rights pointers	Rights-owner reconciliation	Helps reconcile DSR blocks to authoritative asset identity without relying solely on platform IDs.

Implementation note: DSR is a downstream accounting/reporting interface; DAAP improves reconciliation by supplying a stable asset identity + version that can be referenced alongside ISRC/UPC and platform internal IDs.

C.3 DAAP → BWF/bext and WAV chunk metadata

DAAP treats WAV/BWF embedded metadata as bridge surfaces, not as the authoritative identity layer. BWF/iXML are useful export targets when audio must travel alone, but they remain fragile under transformation.

DAAP v1 concept	BWF/WAV chunk surface	Export feasibility
asset.asset_id, asset.asset_version	bext.Description, INFO tags, or custom iXML fields	Possible, but limited in bext; better in iXML/aXML where supported
events.creation.time	bext.OriginationDate, bext.OriginationTime	Directly mappable for basic timing metadata
Time reference / sync anchors	bext.TimeReference	Applicable where meaningful
Minimal credits pointers	iXML	Suitable for structured export subsets
tools[]	iXML / aXML / XMP chunks	Technically possible, interoperability varies
integrity	No standard BWF field	May be placed in XML chunks as opaque data, but not natively verifiable by most audio tools

C.4 DAAP → iXML (when applicable)

DAAP v1 concept	iXML carriage approach	Notes
asset.asset_id, asset.asset_version	custom iXML element under DAAP namespace	Pointer back to DAAP authority when audio travels alone
parties[] subset	iXML elements for contributor subset	Keep minimal; iXML is not a complete rights graph
events.creation	iXML or bext date/time	Useful for traceability, not cryptographic proof
integrity.manifest_hash	optional iXML opaque field	Lightweight check only; not full DAAP verification

C.5 Summary

- ERN is the outward-facing delivery interface to DSPs.
- DSR is the inward-facing reporting interface back to rightsholders.
- BWF/iXML are useful embedded metadata bridges when audio must travel alone.
- DAAP remains the authoritative asset object and verification layer.

Appendix D.

Threat Model and Security Considerations

This appendix defines the DAAP v1 security posture in standards-style form: attacker goals, mitigations, and residual risk. It is aligned with the normative choices in Sections 3–10 and Appendices A–C, including canonicalization, hash/signature baselines, trust models, revocation, timestamps, and disclosure tiers.

D.1 Security objectives DAAP v1 is designed to provide cryptographically verifiable bindings between:

1. asset identity + version,
2. payload bytes,
3. canonical manifest claims, and
4. provenance / audit event chain. Primary security objectives:

- Integrity - detect tampering of manifest and payload bindings
- Authenticity- identify signer(s) and authorization basis
- Auditability - support evidence that a manifest state existed at or before time T
- Non-equivocation (bounded) - reduce probability of undetected “forked truths” through versioning, derivation, and optionally transparency logs
- Privacy-aware disclosure- allow redaction without destroying verification.

DAAP is not intended to prevent copying of audio bytes. Its function is to make copied or redistributed assets remain legible in terms of identity, provenance, and policy pointers.

D.2 Assets under protection DAAP v1 protects the following security-relevant objects:

- active manifest bytes
- payload bindings
- signature envelopes
- trust anchors
- revocation-status evidence
- timestamp evidence
- derivation chain.

D.3 Adversary model DAAP v1 assumes the following attacker classes:

- A1 - opportunistic modifier editing files in transit or at rest
- A2 - malicious distributor/aggregator remapping IDs, swapping payloads, or stripping context
- A3 -malicious uploader claiming authorship or legitimacy for unauthorized content

- A4- malicious implementer emitting non-conformant manifests that appear valid to lax verifiers
- A5 - credential attacker using stolen keys or compromised certificates
- A6 - privacy adversary inferring confidential production details from manifests
- A7 - denial-of-service actor targeting verification dependencies.

.

D.4 Attack surfaces and threats

D.4.1 Payload substitution Threat: replace master.wav or a stem while leaving the manifest unchanged. Mitigation: mandatory SHA-256 payload bindings plus signature coverage over payload bindings. Residual risk: lazy verifiers that fail to re-hash payload bytes may miss substitution.

D.4.2 Manifest tampering Threat: alter contributors, rights pointers, or synopsis edges while keeping superficial appearance. Mitigation: canonicalization + manifest hash + signature over canonical form. Residual risk: issuers signing non-canonical bytes or verifiers accepting non-canonical input create exploitable implementation defects.

D.4.3 Signature wrapping / substitution / mix-and-match Threat: copy a valid signature from one manifest and attach it to another, or confuse active- manifest selection. Mitigation: signature binds to a stable manifest hash; manifest store requires deterministic active-manifest selection. Residual risk: multi-manifest containers increase verifier complexity.

D.4.4 Trust-anchor compromise or mis-issuance Threat: trusted CA mis-issues certificates, or a consortium trust list is poisoned. Mitigation: dual trust models, explicit verification policy, revocation requirements. Residual risk: WebPKI is broad; consortium lists concentrate governance risk.

D.4.5 Stolen signing keys Threat: attacker signs counterfeit assets with stolen credentials. Mitigation: revocation checking + timestamping; timestamps establish signing-time relation to compromise. Residual risk: soft-fail revocation policies may still admit counterfeit assets.

D.4.6 Replay / cloning Threat: copy a legitimate DAAP asset and redistribute it as newly authorized. Mitigation: DAAP preserves chain-of-custody anchors and version evidence; timestamps help detect false “newly packaged” claims. Residual risk: business-rule enforcement remains outside DAAP.

D.4.7 Equivocation (“two truths” problem) Threat: issuer signs two different manifests for the same asset_id and version. Mitigation: monotonic versioning, derivation chain, optional transparency-log profile. Residual risk: without shared log or registry, global equivocation detection remains difficult.

D.4.8 Privacy leakage Threat: manifest exposes sensitive contributors, toolchains, or production structure. Mitigation: disclosure tiers + omission / commitment / encryption + signed redaction transforms. Residual risk: even minimal graph structures may leak inference patterns (e.g., number of stems).

D.4.9 Downgrade / algorithm agility abuse Threat: attacker forces weaker or unknown algorithms that a verifier accepts. Mitigation: mandatory baseline suites, explicit registry semantics, reject-unknown policy for security-critical fields. Residual risk: poor verifier implementations remain exploitable.

D.4.10 Denial-of-service against verification Threat: attacker blocks OCSP, CRL, trust-list, or timestamp endpoints. Mitigation: explicit caching policy and defined failure outcomes. Residual risk: online dependencies remain attack surfaces; strict audit verification may require live checks.

D.5 Security considerations for implementers DAAP v1 implementers SHOULD:

1. fail closed for audit-grade verification when revocation or timestamp requirements are unmet;
2. isolate parsing from verification; canonicalize before trust decisions;
3. treat locators as untrusted, with identity determined by hashes;
4. enforce schema constraints and reject unknown critical fields;
5. log verification outcomes deterministically for audit and dispute workflows.

D.6 Residual-risk statement DAAP v1 improves integrity and provenance verifiability, but it does not eliminate:

- rights disputes over ownership/splits,
- marketplace enforcement challenges,
- copying/redistribution of audio bytes,
- global equivocation detection without shared logs/registries, or
- operational risk in CA ecosystems, consortium governance, or endpoint availability. These risks are addressed through published trust/timestamp policies and explicit conformance behavior, not by implied guarantees.

Appendix E.

Glossary and Identifier References

```
This appendix provides:

(1) DAAP-specific terminology used normatively in this document, and
(2) a reference overview of core music-industry identifiers and adjacent exchange
standards thatappear in DAAP manifests, mappings, and interoperability discussions.
The purpose is terminological precision. This appendix does not redefine the external
standards themselves; it locates them within the DAAP authority model.
```

E.1 DAAP Glossary (Normative Terms)

Active Manifest The single manifest instance selected by deterministic rules as the authoritative claims object for an asset. Historical manifests, countersignatures, or derived manifest views MAY exist, but only one manifest is authoritative at a time for verification purposes. Active-manifest selection semantics are part of conformance.

```
Asset (DAAP Asset)

A packaged, versioned, verification-addressable digital object consisting of:
(1) one or more payload bindings,
(2) a canonical manifest, and
(3) integrity material.
In DAAP, an asset is not identical to a single file instance. It is the authority-
bearing object to which identity, lineage, and provenance attach.
```

Asset ID (asset_id): The stable identifier for the DAAP asset across versions. asset_id identifies the asset lineage, not merely one serialized payload instance. Multiple payload manifestations MAY correspond to the same asset_id across versions or delivery contexts.

Asset Version (asset_version) A monotonic or semantically versioned label identifying one specific manifest state. Versions are connected through explicit parent pointers and/or derivation events. asset_version identifies a state of the asset, not a generic release family.

Authority Surface The location within the architecture where authoritative identity and verification semantics reside. In DAAP v1, authority is localized in the Manifest Store, not in container headers, filenames, or downstream database rows.

Bundle / Container The physical packaging form that carries the Manifest Store and, optionally, the referenced payloads. DAAP v1 defines a normative bundle model and one optional bridge profile for sidecar / repository manifest-store deployment. The container is a transport object; authority is still determined by manifest and binding semantics.

Canonical Manifest The manifest encoded in a protocol-defined form and canonicalization scheme such that all compliant implementations derive identical bytes for hashing and signing. In DAAP v1, the baseline canonical manifest is JSON canonicalized via RFC 8785 (JCS).

Canonicalization The process of transforming a manifest into a deterministic byte representation for hashing and signing. In DAAP v1, canonicalization is mandatory for integrity and is not treated as an implementation convenience. For JSON, the baseline scheme is RFC 8785 JCS.

Claims Object The structured manifest content expressing identity, attribution, lineage, references, disclosure state, and event metadata. The claims object is distinct from the payload and distinct from the cryptographic envelope Content Binding / Payload Binding A cryptographic association from a manifest to payload bytes using hashes and, typically, signatures. Payload identity is determined by digest agreement, not by filename, path, or container header. DAAP v1 requires baseline SHA-256 bindings.

DAAP Manifest Store The authoritative location within the DAAP package containing the active manifest and associated integrity material such as signatures and timestamp tokens, plus any historical manifest objects if present. The Manifest Store is the primary authority surface of the DAAP asset model.

Derivation Chain The sequence of versions and/or derived assets linked by explicit parent pointers, derives_from edges, and event records. The derivation chain is used to formalize alternates, remasters, stems, edits, redacted variants, and downstream transforms as related asset states rather than unrelated files.

Disclosure Tier A declared policy level that determines which classes of fields are present, omitted, committed, or encrypted in a given manifest view. DAAP v1 defines four tiers: T0_PUBLIC, T1_INDUSTRY, T2_PARTNER, and T3_PRIVATE. Disclosure tier is part of the claims context and therefore part of verification semantics.

Hash Commitment A privacy-preserving representation in which the manifest includes hash(value) and algorithm identifier rather than plaintext. Hash commitments support selective disclosure while preserving cross-version stability checks.

Integrity Material The set of cryptographic objects used to validate manifest and payload state, including manifest hashes, content bindings, signatures, revocation-dependent trust evaluation, and optional trusted timestamps. Locator A transport-level pointer to payload location (e.g., bundle path or URI-like reference). A locator is not an identity surface. Payload identity remains determined by content bindings.

PartyRef A typed reference to a person, organization, tool vendor, label, publisher, or other actor associated with the asset. PartyRef carries identity references and roles without requiring a full rights graph inside the manifest.

Portable Synopsis Graph The graph-bounded structural representation used by DAAP to preserve render identity, lineage, and payload mapping without attempting DAW session recreation. The synopsis is intentionally smaller in scope than a studio metadata model such as RIN.

Redaction Record A structured record indicating that a manifest field was omitted, hash-committed, or encrypted, together with reason, actor, and, where policy requires, temporal evidence. A redaction record is part of the provenance chain, not an editorial note.

Trusted Verification Verification in which cryptographic validity, trust-anchor status, revocation policy, and where applicable timestamp policy have all been evaluated and satisfied. A signature that verifies mathematically but chains to an unknown or revoked signer is not "trusted verification."

Verifier Determinacy The requirement that independent conformant verifiers derive the same validation inputs and the same result classes from the same asset state. Determinacy applies to canonicalization, signature inputs, active-manifest selection, trust evaluation, revocation, and redaction handling.

E.2 Identifier References (Core External Identifiers)

ISRC - International Standard Recording Code A standardized identifier for a sound recording. ISRC identifies the recording-level resource, not the release as a commercial package and not the musical work as a composition. In DAAP, ISRC is carried as an external recording identifier reference inside the asset identity block when available.

ISWC - International Standard Musical Work Code A standardized identifier for a musical work / composition. ISWC identifies the underlying work, not a specific recording. In DAAP, ISWC is carried as a work-level reference and is distinct from ISRC by type and scope.

```
GRid - Global Release Identifier

A release-level identifier historically introduced for identifying releases /
products as distinct from individual recordings. GRid occupies a different entity
scope than ISRC: it identifies a release package rather than a track-level recording.
In DAAP, GRid, when present, is treated as an optional release-level external
identifier, not as the asset's authoritative internal ID. Casey's AES metadata paper
listed GRid explicitly among the metadata-era identifier systems and distinguished it
from ISRC.
```

UPC / EAN Commercial product identifiers typically used at the release/package level in distribution systems. In practice these often function as release-level commerce identifiers in place of or alongside GRid.

IPI / IPN / ISNI Contributor or interested-party identifier systems used in rights, collection-society, or broader cross-domain identity environments. These are not DAAP-native IDs; they are external identity references that MAY populate PartyRef.identifiers when available.

E.3 DDEX and Adjacent Exchange References

RIN - Recording Information Notification A DDEX standard for structured collection and exchange of metadata during and following recording creation. RIN focuses on contributor roles, technical session information, and studio- process metadata. In the DAAP architecture, RIN is

complementary, not duplicative: RIN captures studio/session metadata upstream; DAAP preserves an asset-scoped authority object downstream.

RIN FAQ The explanatory DDEX layer describing the purpose, scope, and workflow intent of RIN. It is useful for operational understanding but is not itself the normative schema.

RIN Implementer Portal The DDEX implementation surface containing message structures, schema resources, and implementer guidance for RIN deployments. It is the appropriate operational reference when mapping DAAP fields to RIN-facing artifacts.

ERN - Electronic Release Notification A DDEX release-delivery standard used to communicate release metadata and commercial terms to DSPs. In the DAAP model, ERN is a delivery interface, not the asset authority surface. DAAP MAY map manifest data outward to ERN, but ERN does not replace the DAAP Manifest Store.

DSR - Digital Sales Reporting A DDEX reporting standard used to communicate sales and usage information back to licensors and rightsholders. DSR is a downstream reporting interface and therefore sits outside. DAAP's authority layer, though DAAP asset IDs and version IDs may improve reconciliation when used alongside ISRC/UPC and platform-local IDs.

MEAD A DDEX enrichment standard used to communicate non-core descriptive metadata to DSPs and related downstream systems. MEAD supplements discovery and presentation workflows; it does not provide DAAP-style cryptographic identity or provenance guarantees.

E.4 Identifier-Scope Distinctions (Normative Clarification)

The following distinctions SHALL be preserved in DAAP implementations:

- ISRC identifies a recording
- ISWC identifies a musical work / composition
- GRid / UPC / EAN identify a release / product package
- IPI / IPN / ISNI identify persons or parties
- DAAP asset_id identifies the DAAP asset lineage
- DAAP asset_version identifies a specific asset state. A conformant DAAP implementation SHALL NOT collapse these scopes into a single overloaded identifier field. Namespace ambiguity at the identifier layer directly degrades verifier determinacy and reconciliation performance.

E.5 Operational Summary

This appendix establishes the terminological boundary that underlies the rest of the protocol:

- DAAP-native terms define the authority model and verification model.
- External identifiers define the entity references that DAAP may carry.

• DDEX standards define exchange surfaces, not the DAAP authority surface itself.

• The DAAP Manifest Store remains the authoritative object through which these external identifiers, structural synopsis references, and integrity semantics become co-resident and verification-addressable.

References

[1] International Organization for Standardization. (n.d.). About ISO. ISO. https://www.iso.org
[2] Ivors Academy. (2021). Data issues are at the heart of half a billion pounds a year of unallocated or misallocated streaming royalties for songwriters and rightsholders.
[3] DDEX RIN. (2026). Recording Information Notification-Implementer portal. DDEX. https://ddex.net
[4] DDEX RIN. (2026). Proprietary identifiers. DDEX. https://ddex.net
[5] Adams, C., Cain, P., Pinkas, D., & Zuccherato, R. (2001). Internet X.509 public key infrastructure time-stamp protocol (TSP) (RFC 3161). Internet Engineering Task Force. https://www.rfc-editor.org/rfc/rfc3161
[6] AES Standards Committee Working Group. (n.d.). AES-X098B: Best practices for audio metadata (working draft). Audio Engineering Society.
[7] Apple Inc. (2023). AAC encoding guidelines. Apple Developer Documentation.
[8] Audio Engineering Society. (2014). AES Technology Report: TC-NAS-981 - Network Audio Systems Task Group Report. Audio Engineering Society.
[9] Audio Engineering Society. (2018). AES-R5-2018: AES standard for audio metadata-Framework for communication. Audio Engineering Society.
[10] Coalition for Content Provenance and Authenticity. (2024). C2PA specification v2.2. Linux Foundation. https://spec.c2pa.org
[11] England, P., Malvar, H. S., Horvitz, E., Stokes, J. W., Fournet, C., Burke-Aguero, R., & Zaman, A. (2020). AMP: Authentication of media via provenance. arXiv. https://arxiv.org/abs/2001.07886
[12] Content Authenticity Initiative. (n.d.). Secure mode enabled. https://contentauthenticity.org
[13] Content Authenticity Initiative. (2025). Understanding manifests: Manifest store and content bindings. https://opensource.contentauthenticity.org/docs/manifest/understanding- manifest/
[14] C2PA. (2025). Implementation guidance: Manifest repositories and external manifest stores. https://spec.c2pa.org/specifications/specifications/1.3/guidance/Guidance.html
[15] Rundgren, A., Jordan, B., & Erdtman, S. (2020). JSON Canonicalization Scheme (JCS) (RFC 8785). Internet Engineering Task Force. https://www.rfc-editor.org/rfc/rfc8785
[16] Schaad, J. (2022). CBOR Object Signing and Encryption (COSE): Structures and Process (RFC 9052). Internet Engineering Task Force. https://www.rfc-editor.org/rfc/rfc9052
[17] National Institute of Standards and Technology. (2015). Secure Hash Standard (SHS) (FIPS PUB 180-4). U.S. Department of Commerce.
[18] National Institute of Standards and Technology. (2023). Digital Signature Standard (DSS) (FIPS PUB 186-5). U.S. Department of Commerce.
[19] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., & Polk, W. (2008). Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile (RFC 5280). Internet Engineering Task Force. https://www.rfc-editor.org/rfc/rfc5280
[20] C2PA. (2025). C2PA technical specification v2.2: Trust model and manifest processing. https://spec.c2pa.org/specifications/specifications/2.2/specs/C2PA_Specification.html
[21] C2PA. (2025–2026). Conformance / Trust Lists. https://c2pa.org/conformance/

[22] Content Authenticity Initiative. (2025). C2PA conformance overview and trust list implementation. https://opensource.contentauthenticity.org/docs/conformance/
[23] Myers, M., Ankney, R., Malpani, A., Galperin, S., & Adams, C. (2013). X.509 Internet Public Key Infrastructure Online Certificate Status Protocol (OCSP) (RFC 6960). Internet Engineering Task Force. https://www.rfc-editor.org/rfc/rfc6960
[24] Digital Data Exchange, LLC. (2022). DDEX standards for recording, release, and rights information. DDEX. https://ddex.net
[25] Digital Data Exchange, LLC. (2026). Electronic Release Notification (ERN) message suite. DDEX. https://ddex.net
[26] Digital Data Exchange, LLC. (2026). Digital sales reporting (DSR) message suite. DDEX. https://ddex.net
[27] Digital Data Exchange, LLC. (2026). Recording Information Notification (RIN): Overview and implementation resources. DDEX. https://ddex.net
[28] Digital Data Exchange, LLC. (2026). MEAD metadata enrichment for digital service providers. DDEX. https://ddex.net
[29] DDEX Knowledge Base. (2026). ERN 4 structure: NewReleaseMessage sections, releases, resources, and deals. DDEX Knowledge Base. https://kb.ddex.net
[30] DDEX Knowledge Base. (2026). RIN introduction: Machine-to-machine studio metadata capture across iterative production stages. DDEX Knowledge Base. https://kb.ddex.net
[31] DDEX Knowledge Base. (2026). RIN FAQ: Structured collection and exchange of metadata during and following recording creation. DDEX Knowledge Base. https://kb.ddex.net
[32] DDEX Knowledge Base. (2026). MEAD explained: Non-core enrichment metadata for DSP experience. DDEX Knowledge Base. https://kb.ddex.net
[33] International Federation of the Phonographic Industry. (2023). Global music report 2023. IFPI.
[34] International Federation of the Phonographic Industry. (n.d.). ISRC handbook / ISRC overview documentation. IFPI. https://isrc.ifpi.org
[35] CISAC. (n.d.). ISWC. https://iswc.org
[36] Kahn, R., & Wilensky, R. (2006). A framework for distributed digital object services. Corporation for National Research Initiatives. https://www.cnri.reston.va.us
[37] European Broadcasting Union. (2017). Specification of the Broadcast Wave Format (BWF) (EBU Tech 3285). https://tech.ebu.ch/publications/tech3285
[38] Federal Agencies Digital Guidelines Initiative. (2021). Embedded metadata in Broadcast WAVE files (BWF), version 3. FADGI. https://www.digitizationguidelines.gov
[39] Library of Congress. (2023). WAVE audio file format description (FDD000016). https://www.loc.gov/preservation/digital/formats/fdd/fdd000016.shtml
[40] Microsoft. (1991). Multimedia programming interface and data specifications 1.0: Resource Interchange File Format (RIFF). Microsoft Corporation.
[41] European Broadcasting Union. (2011). EBU Tech 3293: EBU core metadata set for audio objects, production and reporting. EBU. https://tech.ebu.ch
[42] UK Intellectual Property Office. (2021). Music 2025: The music data dilemma. UK Intellectual Property Office. https://www.gov.uk/government/publications/music-2025-the- music-data-dilemma
[43] Casey, M. (2003). Demystifying audio metadata. Journal of the Audio Engineering Society, 51(7/8), 744–751.

[44] Abdallah, S. A., Raimond, Y., & Sandler, M. (2006). An ontology-based approach to information management for music analysis systems. AES 120th Convention, Paris, France. Audio Engineering Society.
[45] Deezer. (2025). Deezer reveals 18% of all new music uploaded to streaming is fully AI-generated. Deezer Newsroom.
[46] DDEX Knowledge Base. (2026). Digital sales reporting (DSR) overview. DDEX Knowledge Base. https://kb.ddex.net
[47] DDEX. (2024). DSR overview (record/block structure). DDEX.
[48] Library of Congress. (n.d.). Broadcast WAVE Audio File Format, Version 2 (format description). Library of Congress.
[49] Gallery Software. (n.d.). Introduction to iXML. iXML.
[50] MediaArea. (n.d.). BWF MetaEdit: XML chunks (iXML, aXML, and XMP) technical view. MediaArea.
[51] Content Authenticity Initiative. (2025). Working with manifests: Time-stamps (RFC 3161 recommendation). https://opensource.contentauthenticity.org
[52] C2PA. (2025). Implementation guidance: Time-stamp manifest guidance.
[53] Laurie, B., Langley, A., & Kasper, E. (2013). Certificate Transparency (RFC 6962). Internet Engineering Task Force.
[54] Truskovsky, B., Fitterer, E., & Stradling, A. (2021). Certificate Transparency Version 2.0 (RFC 9162). Internet Engineering Task Force.