SSO ARTISTS ENTERTAINMENT · SYSTEMS ENGINEERING

SSO System (ARDNDEX and NODE) Technical Reference v5.2

Carbon-system technical reference for the standards, validation, intelligence, orchestration, packaging, delivery, API, and operations layers that compose the SSO media infrastructure.

VERSION	DOMAIN A	DOMAIN B	DESIGN
5.2	ARDNDEX	NODE	IBM Carbon

REFERENCE MODEL

This v5.2 reference groups the source architecture into two top-level system planes. ARDNDEX contains standards, validation, metadata, and intelligence services. NODE contains distributed execution, orchestration, packaging, delivery, APIs, and operations.

Carbon domain color system

ARDNDEX	NODE	Shared	Control	Risk

00 / REFERENCE MAP

Technical reference map

The reference is organized by system responsibility rather than by end-user feature. ARDNDEX defines the rules, schemas, validation, and analytical gates. NODE executes those rules as distributed jobs, APIs, packages, deliveries, and operational workflows.

01 System architecture and design principles	13 Partner adapters and delivery profiles
02 ARDNDEX domain model	14 Authentication and API surface
03 Canonical ingestion and metadata normalization	15 Release, asset, and lyric APIs
04 DDEX engine: ERN and MEAD	16 Packaging, delivery, partner, and webhook APIs
05 Audio, ADM, and metadata parity validation	17 Reporting, intelligence, incident, and admin APIs
06 Lyrics, TTML, and localization	18 Security and identity
07 QC orchestration and policy packs	19 Error handling, retries, and resilience
08 Monte Carlo, QMC, and Bayesian detection	20 Schema crosswalk and core data fields
09 Catalog intelligence and feedback loops	21 Operator consoles and monitors
10 NODE domain model	22 Operations playbooks and SLOs
11 Event orchestration and deterministic jobs	23 End-to-end data flows
12 Packaging and partner export	24 Glossary and revision notes

SOURCE BASIS

The underlying architecture defines a modular, distributed, self-validating stack with versioned internal APIs, independently deployable nodes, event-driven orchestration, immutable release objects, reproducible builds, and audited transitions.

01 / SYSTEM ARCHITECTURE

Modular, self-validating media infrastructure

SSO System is modeled as a distributed media infrastructure with strict metadata parity across release manifests, enrichment metadata, immersive audio, and lyric formats. The system is designed around versioned contracts and audited state transitions rather than mutable ad-hoc handoffs.

Core architecture principles

- Internal APIs connect modules through versioned contracts so components can evolve independently.
- Event-driven orchestration coordinates transitions, retries, and back-pressure across ingestion, validation, packaging, and delivery.
- Release state is treated as reproducible and audit-traceable, with immutable artifacts after lock points.
- Telemetry is part of the control loop: logs, metrics, traces, and operator outcomes are used to refine policy.
- Validation is policy-driven. Structural gates block invalid states while advisory checks surface remediation without silently changing source truth.

System plane split

ARDNDEX

Standards, normalization, DDEX, validation, metadata parity, lyrics, localization, statistical quality gates, and catalog intelligence.

NODE

Distributed execution, event orchestration, package builds, partner transport, APIs, monitoring, security enforcement, and operational controls.

Shared catalog truth

Canonical release objects, identifiers, rights state, asset manifests, policy versions, and audit events bind both planes together.

Control loop

Delivery outcomes and operational evidence flow back into policy tuning, anomaly detection, and reliability decisions.

Primary lifecycle

Ingest	Validate	Master	Package	Deliver	Audit	Analyze

02 / ARDNDEX

Standards, validation, and intelligence plane

ARDNDEX is the reference domain for the rule-bearing portions of the system. It converts raw metadata and media into canonical, validated, partner-ready state while preserving traceability back to the source release object.

Capability	Technical responsibility
Normalization	Controlled vocabularies, encoding normalization, profile selection, canonical casing, and cross-field constraints.
DDEX engine	Schema-aware ERN and MEAD generation, XSD validation, controlled vocabularies, signatures, and diff-based resubmission.
Media validation	Audio, ADM/BWF, artwork, lyric timing, and technical-readiness checks with blocking and advisory thresholds.
Metadata parity	Two-way comparison between embedded asset metadata and canonical manifests to prevent drift.
Policy orchestration	Graph-based reduction of distributed validation results into a deterministic verdict with explainable reasons.
Statistical validation	Monte Carlo and quasi-Monte Carlo sampling for failure prediction plus Bayesian anomaly detection for low-frequency risks.
Catalog intelligence	SQL-backed analytics, forecasts, cohort analysis, and anomaly signals that feed policy tuning.

DESIGN RULE

ARDNDEX does not replace the canonical release record. It evaluates, transforms, and explains that record under pinned standards and policy versions.

03 / ARDNDEX

Canonical ingestion and metadata normalization

The ingestion layer is the front door for assets and metadata. Its job is to construct a canonical release package that downstream validators and NODE workers can reproduce deterministically.

Normalization responsibilities

- Normalize UTF-8/UTF-16 inputs and byte-order handling into a consistent internal representation.
- Apply controlled vocabularies and partner profile rules without destroying the original user-supplied values needed for audit.
- Validate cross-field dependencies, identifier shape, release-level versus track-level ownership, and required metadata presence.
- Inspect media basics such as sample rate, bit depth, channel configuration, artwork dimensions, and lyric timing sanity.
- Build version-pinned manifests that preserve ISRC, UPC, contributor, rights, and asset relationships.

Canonical package boundary

Component	Canonical state	Downstream use
Release object	Title, artists, identifiers, territories, dates, rights	DDEX, packaging, delivery
Asset manifest	Files, checksums, media type, technical metadata	QC, mastering, transport
Contributor graph	Names, roles, identifiers, publishing relationships	ERN/MEAD, credits, rights checks
Policy pin	Applicable standards and partner profile versions	Reproducible validation
Audit context	State transition, actor, time, previous version	Incident analysis and compliance

04 / ARDNDEX

DDEX engine: ERN and MEAD

The DDEX layer generates, validates, and versions partner-facing metadata payloads from the same canonical catalog truth. ERN carries release, resource, deal, and rights structures. MEAD carries enrichment data such as lyrics, contributor context, artwork, and supplemental descriptors.

Engine behavior

- XSD-driven validation is pinned to the relevant DDEX profile and schema version.
- Controlled vocabularies enforce role names, release types, rights semantics, and partner-readable naming.
- Character-set and normalization policies protect readability across international catalogs.
- Partner-specific rule layers sit above the base schema for destination quirks and conditional requirements.
- Diff-based resubmission regenerates only impacted payloads when a change does not require a full package rebuild.

Payload integrity model

DDEX PROCESS MODEL

```
canonical_release -> ERN profile -> XSD validation -> partner rules
canonical_enrichment -> MEAD profile -> XSD validation -> parity check
payload_version + policy_version + checksum -> audit record
```

PARITY GUARANTEE

ERN and MEAD outputs inherit identifiers and rights state from the same canonical model, reducing the risk that partner packages disagree about the release they describe.

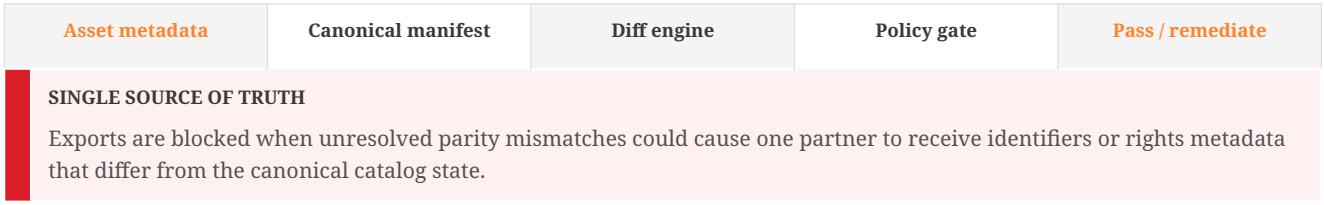
05 / ARDNDEX

Audio, ADM, and metadata parity validation

Media validation applies deterministic technical checks to stereo masters, Apple Digital Masters candidates, and immersive ADM/BWF assets. The same stage also compares embedded metadata with the canonical release manifest.

Validation area	Examples
ADM/BWF integrity	Frame alignment, bed/object mapping sanity, channel structure, timing, and asset-readiness checks.
Loudness and peaks	LUFS ranges, true-peak limits, inter-sample peaks, headroom, and clipping detection.
Apple Digital Masters	Noise floor, headroom, dynamic behavior, and master-readiness checks.
Embedded metadata	ISRC, UPC, rights tags, and other embedded fields compared against ERN/MEAD manifests.
Checksums/signatures	Detect truncation, corruption, or unintended asset changes between validation and packaging.
Threshold classes	Blocking rules produce deterministic failure; advisory rules preserve operator visibility without blocking.

Parity decision



SINGLE SOURCE OF TRUTH

Exports are blocked when unresolved parity mismatches could cause one partner to receive identifiers or rights metadata that differ from the canonical catalog state.

06 / ARDNDEX

Lyrics, TTML, and localization

The lyric and localization layer turns timed text and contributor-language metadata into portable partner-ready representations while preserving timing structure, language tags, and role context.

Lyric transformation path

Source text / ASS	Timing normalize	BGV separation	Apple TTML	MEAD mapping

- Word-level timing and stanza structure are preserved through conversion.
- Background-vocal parentheticals can be separated into distinct lanes or role-tagged text.
- Negative, overlapping, or out-of-range intervals are rejected by timing policy.
- MEAD mapping carries language codes, role metadata, and enrichment context alongside the text payload.

Internationalization

- BCP-47 language tags are used across lyrics, titles, and contributor metadata where applicable.
- Canonical casing and punctuation policies standardize display names and version descriptors.
- Right-to-left scripts, diacritics, and transliteration require region-aware rendering and validation.
- Territory-specific policy packs can apply localized exceptions without changing the canonical record for other regions.

07 / ARDNDEX

QC orchestration and policy packs

The QC orchestrator composes many independent checks into one explainable decision. Structural failures stop the pipeline early; advisories are retained as evidence for operators and later policy tuning.

Decision graph

Layer	Behavior
Pre-flight	Checks audio, artwork, metadata, XML, rights completeness, and partner prerequisites before package build.
Reducer	Combines sub-check pass/fail/advisory states into one final verdict.
Explainability	Every failure carries machine-readable reasons and human-readable remediation guidance.
Retry semantics	Transient checks can retry with controlled backoff without erasing the original failure context.
Operator override	Overrides require justification and generate audit events rather than silently bypassing policy.
Policy packs	Partner and standards rules are pinned by version so the same release can be reproduced later.

Example policy model

POLICY EXAMPLE

```
loudness.target_lufs: -14
audio.max_true_peak_dbfs: -1.0
lyrics.allow_overlap: false
policy_pack: <partner_version>
result: blocking | advisory | pass
```

08 / ARDNDEX

Monte Carlo, QMC, and Bayesian detection

The statistical validator acts as a predictive layer above deterministic schema checks. It is intended to estimate failure probability, explore edge cases, and surface anomalies that static validation may not capture.

Monte Carlo and QMC

- Sample metadata drift, timing jitter, and partner-specific tolerances to estimate downstream reject risk.
- Use low-discrepancy Sobol or Halton sequences for more reproducible edge-case coverage than naive random sampling.
- Emit predicted rejection probability, severity, and candidate root causes for operator review.
- Refine thresholds over time using observed delivery outcomes and verified operator resolutions.

Bayesian anomaly model

- Update priors when new ingestion or delivery errors are observed for similar payloads.
- Use credible intervals to flag unusual stream, royalty, or delivery behavior that falls outside expected ranges.
- Localize anomalies to fields, assets, or partner adapters when evidence supports attribution.
- Feed confirmed operator actions back into the model to reduce repeat false positives.

STATISTICAL ROLE

Probabilistic output is an evidence layer, not a substitute for deterministic standards checks. Schema and rights gates remain authoritative for structural compliance.

09 / ARDNDEX

Catalog intelligence and feedback loops

Catalog Intelligence joins delivery, stream, royalty, and asset-health data into analytical views that can inform product, finance, and policy decisions. The same canonical identifiers used in delivery are used to reconcile analytics.

Capability	Reference behavior
Catalog reconciliation	Join statements and usage events back to ISRC/UPC and release relationships.
Forecasting	Use Monte Carlo or Bayesian forecasts for release timing, campaign windows, and expected outcomes.
Anomaly detection	Alert on sudden deltas in streams, royalties, or delivery behavior by territory or partner.
Cohort analysis	Compare releases by type, genre, campaign tag, or other catalog dimensions.
Evidence bundles	Attach contextual data to anomalies so operators can distinguish real issues from expected variance.
Policy feedback	Confirmed outcomes can adjust validator thresholds or partner-specific heuristics under change control.

Feedback loop

Delivery	Usage / royalty	Catalog Intelligence	Evidence	Policy tuning

10 / NODE

Distributed execution and delivery plane

NODE is the execution domain for independently deployable services and workers that move a release through validation, mastering, packaging, delivery, reporting, and operations. NODE consumes ARDNDEX policy and canonical catalog state through versioned contracts.

NODE responsibility	Technical model
Job execution	Idempotent, replay-safe jobs with deterministic inputs and outputs.
Event orchestration	Transitions, retries, back-pressure, and state changes coordinated through event-driven workflows.
Packaging	Build partner-specific artifacts from the canonical release and pinned policy packs.
Transport	Asynchronous delivery jobs, receipts, external IDs, and retry state.
API services	Authentication, releases, assets, lyrics, packaging, delivery, partners, reports, incidents, and admin surfaces.
Operational state	Logs, metrics, traces, SLA timers, audit events, and incident links.
Resilience	Versioned artifacts, geographic redundancy, backup/restore procedures, and bounded retry behavior.

NODE BOUNDARY

A node executes a defined contract. It should be independently deployable, observable, replay-safe where appropriate, and unable to silently rewrite canonical truth outside authorized state transitions.

11 / NODE

Event orchestration and deterministic jobs

Media-native CI/CD treats release state changes as events that can trigger validation, package builds, transport, reconciliation, or reporting. The system favors reproducibility and bounded retries over manual one-off execution.

Execution rules

- Idempotent job design allows safe replay after transient failures.
- Deterministic builds ensure the same canonical input and pinned policy produce the same package result.
- Per-environment configuration and protected secrets separate runtime concerns from release metadata.
- Canary delivery can test new policy or adapter behavior with limited blast radius before broad rollout.
- SLA/SLO targets are attached to stages so delays and error-budget burn are visible operationally.

Event classes

Class	Trigger	Typical NODE action
Catalog	Release create/update/lock	Validate and version state
Asset	Upload complete / checksum result	Extract technical metadata, queue validation
QC	Pass / fail / advisory	Advance, halt, or request remediation
Package	Build requested / retry	Render partner artifact and logs
Delivery	Transport success / failure	Archive receipt, map error, retry if eligible
Analytics	Usage / royalty update	Reconcile and emit intelligence jobs
Incident	SLO breach / repeated failure	Open incident with evidence

12 / NODE

Packaging and multi-partner export

Packaging transforms one canonical catalog model into partner-specific outputs. The export layer applies destination schemas, media constraints, rights windows, and policy packs while retaining traceability to the same source release.

Packaging behaviors

- Partner-specific XML, JSON, CSV, bundle, or manifest formats are versioned independently from the canonical catalog schema.
- ISRC/UPC parity and contributor-role mapping are checked before delivery.
- Territorial rights and release windows are transformed into partner-specific availability structures.
- Partial rebuilds or diff-based resubmission can limit changes to the affected payload or asset set.
- Delivery summaries and receipts are archived so external platform state can be reconciled to internal versions.

Example package path

Canonical release	ARDNDEX policy	NODE package build	Partner adapter	Transport receipt

TRANSPORT CONTRACT

Packaging and delivery are separate states. A valid package is not considered delivered until the partner transport stage returns or records a delivery outcome.

13 / NODE

Partner adapters and delivery profiles

The source architecture maintains destination-specific adapters and policy behavior for major and regional platforms. The same underlying catalog truth is reshaped per destination instead of maintaining disconnected catalog copies.

Partner class	Adapter emphasis	Operational signals
Apple	Transporter-oriented package structure, enriched assets, Apple Digital Masters and metadata parity	Validation feedback, package receipts, asset acceptance
Spotify	Metadata mapping, territorial availability, artwork/audio profiles, ingestion error classification	Receipt status, availability, analytics reconciliation
TIDAL	High-fidelity media handling, retry policy, editorial and analytics context	Delivery errors, quality state, territory coverage
YouTube / CID	Music delivery, reference policy, Content ID rights controls	Reference conflicts, claims, match state
Amazon Music	Packaging, availability, voice/search metadata	Ingestion state, catalog availability
Deezer / Pandora	Profile mapping, schema quirks, retry and error taxonomy	Partner error class, partial resubmission state
SoundCloud / TikTok / Meta	Direct or social-music delivery, rights and monetization context	Rights conflicts, availability, usage signals
Regional DSPs	Localization, script handling, endpoint-specific policy	Schema drift, territory delivery, retry state

14 / NODE API

Authentication and API surface

The private API surface is organized around scoped authorization, short-lived credentials, signed event delivery, and explicit service boundaries. The source reference describes OAuth2/SAML federation, JWT validation, token rotation, and HMAC-verified webhooks.

Authorization model

- Scopes constrain access to release, asset, delivery, and administrative operations.
- Access tokens are short-lived and refreshed through protected channels.
- JWT audience and issuer checks prevent tokens from being accepted by the wrong service boundary.
- Webhooks are signed and protected against replay using timestamp or nonce validation.
- Rate limits and quotas can be applied per tenant or client to protect shared infrastructure.

AUTHENTICATION SHAPE

```
POST /v1/auth/token
grant_type=client_credentials
scope=releases:write assets:read
Authorization: <client authentication>
```

API families

Releases, Assets, Lyrics, Packaging, Delivery, Partners, Webhooks, Reports, Catalog Intelligence, Incidents, and Admin are treated as separate functional surfaces with shared authorization and audit expectations.

15 / NODE API

Release, asset, and lyric APIs

API	Primary operations	Control points
Releases	Create drafts, update pre-lock state, read/search catalog, seal artifacts	Version increments, lock semantics, state-change webhooks
Assets	Upload audio/art/lyric media, resume chunked transfers, extract technical metadata	Checksums, media specs, manifest linkage
Lyrics	Upload/validate TTML, convert ASS to TTML, map MEAD lyric metadata, preview	Timing overlap, span validity, background-vocal preservation

Release immutability

Once a release reaches an artifact lock point, downstream jobs should reference the locked version rather than a mutable working draft. New corrections create explicit versions or change records so a delivered package can always be traced to the exact source state that produced it.

Asset integrity

- Chunked or resumable transfer state is separate from the final object record.
- Checksums are verified before an asset becomes eligible for downstream validation.
- Extracted technical metadata is evidence, not a substitute for the canonical catalog metadata entered by authorized users.
- Assets are linked to releases through manifest identifiers rather than loose filenames.

16 / NODE API

Packaging, delivery, partner, and webhook APIs

API	Primary operations	Reference behavior
Packaging	Create partner builds, inspect logs/artifacts, retry partial failures	Idempotent build IDs and pinned policy versions
Delivery	Transport artifacts, archive receipts, retry by job ID	Async jobs, error mapping, external ID audit
Partners	Discover capabilities, media constraints, schema quirks, profile versions	Capability diff when upstream schemas change
Webhooks	Emit signed state, error, analytics, and delivery events	HMAC signatures, replay protection, capped retry on transient failures

Representative event classes

EVENT TAXONOMY

qc.failed
package.ready
delivery.succeeded
delivery.failed
royalty.anomaly
incident.opened

IDEMPOTENCY

Retries should reference the existing job, package, or event identity whenever possible. Duplicate execution must not create duplicate catalog state or duplicate partner submissions.

17 / NODE API

Reporting, intelligence, incident, and admin APIs

API	Purpose	Operational constraint
Reports	Usage, royalty, and ad-hoc exports for BI workflows	Parameterized queries, output limits, generation SLA
Catalog Intelligence	Forecasts, anomalies, cohorts, and evidence bundles	Confidence intervals and exportable results
Incidents	Open, track, resolve, and attach evidence to operational events	Status transitions and SLA timers generate audit state
Admin	Manage roles, policy packs, environment controls, quotas, and audit export	Administrative actions require explicit authority and traceability

Reports and data products

- CSV/Parquet-style export surfaces support downstream analytics stacks without changing canonical catalog state.
- Prebuilt anomaly reports package the context needed for investigation rather than returning only a numeric alert.
- Incident APIs connect partner errors, release versions, jobs, operator notes, and evidence into one traceable record.

18 / CONTROL PLANE

Security and identity

Security is layered across identity, authorization, artifact integrity, environment isolation, and audit. The source architecture calls for least privilege, role-based control, protected secrets, checksums, and threat modeling across supply-chain and partner interfaces.

Control	Reference intent
RBAC and scoped access	Creators, validators, operators, and administrators receive only the authority needed for their functions.
Time-bounded elevation	Critical operational fixes can use temporary elevated access rather than permanent privilege.
Artifact integrity	Checksums and signing protect assets and packages at rest and in transit.
Secrets management	Credentials and environment configuration are stored separately from release data and application code.
PII minimization	Rights and royalty systems should retain only the personal information necessary for legitimate processing.
Audit trail	Privileged actions, overrides, state transitions, and configuration changes remain attributable.
Threat modeling	Supply-chain risk, partner endpoints, replay/spoofing, and insider risk are treated as distinct attack surfaces.

SECURITY WRAP

Authentication, authorization, audit, secret management, and artifact integrity surround the release lifecycle rather than appearing only at the delivery boundary.

19 / NODE OPERATIONS

Error handling, retries, and resilience

The architecture classifies failures by type so structural catalog defects are handled differently from transient transport problems or partner-specific warnings.

Error class	Expected behavior	Typical remediation
Structural	Block progression	Correct schema, field, or asset structure
Parity	Block affected export	Resolve identifier or metadata mismatch
Rights	Hold delivery	Provide or correct rights evidence
Transport	Retry if safe	Backoff, resume, or replay same job identity
Partner schema	Pin/update adapter	Apply tested policy/profile revision
Advisory	Log and expose	Operator decision or future policy tuning

Resilience model

- Dead-letter queues or equivalent holding states preserve irreconcilable payloads with diagnostic context.
- Runbook automation can open templated incidents for repeatable failure patterns.
- Versioned artifact stores and geographic redundancy support recovery and rollback.
- Restore drills validate that backup policy works in practice, not only on paper.
- RPO/RTO targets and SLO error budgets convert resilience goals into measurable operational limits.

20 / SHARED DATA

Schema crosswalk and core data fields

A schema crosswalk connects canonical catalog fields to ERN, MEAD, ADM/BWF, TTML/ASS, and partner-specific representations. The objective is to maintain one source of truth even when destination schemas encode the same concept differently.

Canonical field	Example constraint	Mapped surfaces
title	Required; bounded length and normalized display	ERN release, partner title fields
isrc	Recording identifier pattern and parity requirement	ERN resource, embedded tags, analytics
upc	Release-level product identifier	ERN release, package manifest, reporting
contributors	Role and identifier vocabulary	ERN/MEAD credits, partner role mapping
territories	Normalized rights graph	ERN deals, partner availability matrices
lyrics	Timed text and language metadata	TTML/ASS, MEAD enrichment
assets	Manifest ID, type, checksum, technical metadata	Packaging, delivery, audit
policy_version	Pinned validation/adapter version	QC, builds, incident replay

CROSSWALK PRINCIPLE

Partner adapters can reshape syntax and field names, but they should not invent a second truth for identifiers, rights, contributors, or asset identity.

21 / OPERATIONS

Operator consoles and monitors

The source reference defines a set of operator interfaces that expose validation state, package queues, partner delivery, royalty analytics, discrepancy resolution, and access control.

Console	Operational purpose
Release Wizard	Stepwise asset, metadata, lyric, artwork, QC, and package preparation with inline validation.
Lyrics Sync Editor	Word-level timing, background-vocal lanes, TTML export, and preview playback.
QC Dashboard	Filter pass/fail/advisory state, inspect field-level diffs, retry with context, retain audit history.
Packaging Queue Monitor	Watch partner package builds, SLA timers, idempotent rebuilds, and transport receipts.
Partner Delivery Monitor	Track cross-partner IDs, territory drift, sampling cadence, and delivery alerts.
Royalty Analytics Dashboard	Reconcile statements, identify outliers, project campaigns, and export analysis.
Discrepancy Resolver	Connect parity or delivery mismatches to source fields, assets, and partner state.
Access Control Admin	Manage authorized identities, roles, policy-level access, and audit-sensitive configuration.

22 / OPERATIONS

Operations playbooks and SLOs

Operational reliability is documented as repeatable playbooks rather than tacit knowledge. Incident response, disaster recovery, change management, SLA/SLO policy, and alert routing are all explicit parts of the system design.

Playbook	Key controls	Evidence
Incident response	Severity matrix, owner, rollback, postmortem	MTTA, MTTR, recurrence
DMCA / rights response	Evidence pack, notices, platform timers, retention	Hashes, manifests, timestamps
Disaster recovery	Restore drills, failover, contacts, RPO/RTO	Restore result and recovery time
Change management	RFC, blast radius, canary, abort criteria, rollback	Changelog and policy version
SLA/SLO	Stage objectives, error budgets, burn alerts	Weekly reports and escalations
Monitoring	Metrics, logs, traces, dedupe, suppression, paging	Alert event and linked runbook

Change rule

Changes to policy, partner adapters, or validation thresholds should be versioned and testable. Riskier changes can be canaried against a limited release set before broader deployment.

23 / END-TO-END FLOWS

Reference data flows

Release lifecycle

Creator	Ingestion	ARDNDEX	Mastering	NODE package	Delivery	Audit

The canonical release and asset manifests move through validation and packaging without losing version identity. Delivery receipts and partner IDs return to the same audit graph.

Lyrics lifecycle

Source text	ASS / timing	ARDNDEX lyric QC	TTML / MEAD	NODE package	DSP

Analytics feedback

Delivery	Streams / royalties	Catalog Intelligence	Anomaly evidence	Policy review

Security wrap

AuthN	AuthZ	Audit	Secret mgmt	Artifact integrity

CLOSED-LOOP SYSTEM

The architecture is intentionally cyclical. Delivery is not the end of the pipeline: operational outcomes, analytics, and incidents become evidence for future policy and reliability improvements.

24 / REFERENCE

Glossary and revision notes

Core terms

Term	Meaning in this reference	System area
ADM / BWF	Immersive audio deliverable and container concepts used in technical validation	ARDNDEX
DDEX	Digital data-exchange standards used for release and enrichment metadata	ARDNDEX
ERN	Electronic Release Notification payload family	ARDNDEX
MEAD	Media Enrichment and Description metadata family	ARDNDEX
TTML / ASS	Timed-text representations used in lyric workflows	ARDNDEX
QMC	Quasi-Monte Carlo sampling for structured edge-case coverage	ARDNDEX
Policy pack	Versioned set of validation or partner-specific rules	Shared
Canonical release	Authoritative catalog object from which partner packages are derived	Shared
NODE job	Independently executed unit of orchestration, packaging, delivery, or operations	NODE
Idempotency	Property allowing safe replay without duplicating system effects	NODE
SLO	Service-level objective used to measure operational reliability	Control
Audit event	Traceable record of state, privilege, override, or configuration change	Control

Version 5.2 organization

Version 5.2 reorganizes the legacy infrastructure whitepaper into a two-plane SSO System reference. The technical behaviors retained here come from the source architecture, while the ARDNDEX/NODE split is the organizing model used by this edition.

- ARDNDEX groups standards, validation, normalization, metadata, lyrics, statistical quality, and intelligence functions.
- NODE groups distributed execution, API, packaging, delivery, monitoring, and operational functions.
- Shared canonical state, policy versioning, security, and audit connect both domains.
- The visual system uses IBM Carbon design tokens, grid hierarchy, restrained color coding, and Carbon-style line iconography throughout.

REFERENCE STATUS

This document is a technical reference, not a partner contract. Destination requirements and operational policy can change independently and should be verified against the active system configuration when executing a delivery.