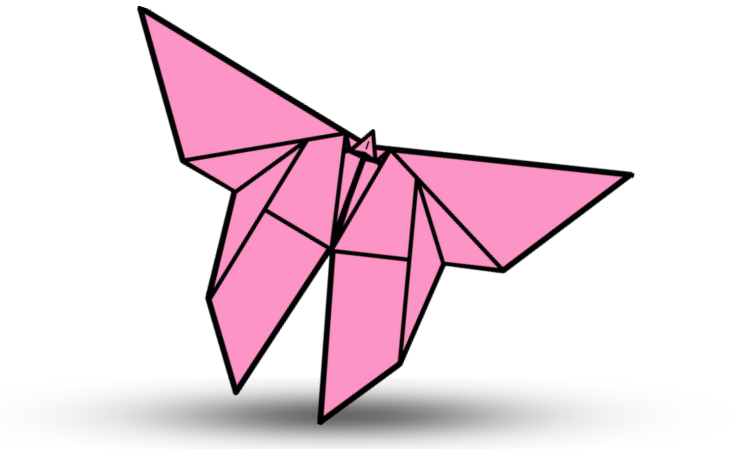
JATS Structural Integrity: The XML-First Framework

Architectural Foundations of JATS-Centric Editorial Systems: A Comprehensive Framework for XML-First Scholarly Production and Machine-Readable Knowledge



zeba.academy

Zeba Academy

First Published by Zeba Academy and Zeba Books.

Publication Year: 2026

Document Series: Zeba Academy Blueprints -- Sovereign Systems Technical Directives

Purpose: This series of blueprint directives is authored to combat the "enshittification" and unnecessary bloat of modern software. Our goal is to reclaim sovereign control over our systems by bridging the gap between deep academic theory and high-stakes industrial implementation. We believe that software should be fast, permanent, and most importantly, understandable to the person who owns and uses it.

Principal Architect: Sufyan bin Uzayr, Google Cloud-Certified Professional DevOps Engineer.

Core Stack: Linux, Rust, Zig, C++, Flutter, and PHP.

Email: hello@zeba.academy

JATS Structural Integrity: The XML-First Framework

Architectural Foundations of JATS-Centric Editorial Systems: A Comprehensive Framework for XML-First Scholarly Production and Machine-Readable Knowledge

The "Single Source of Truth" Philosophy

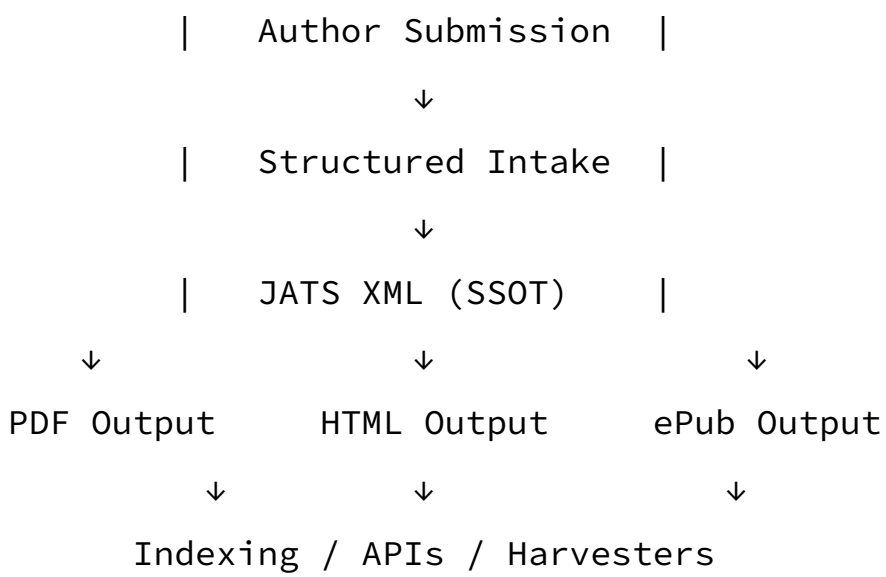
The process of developing infrastructure for scholarly publication has shown us that there is a fundamental problem with traditional publication workflows: there is no standardized, machine-readable way to display content. The traditional publishing pipeline was based around documents, especially PDF, as a source of information. This is a fundamentally flawed approach, as PDF is a presentation format, not a data repository.

The Single Source of Truth (SSOT) concept addresses this issue by making JATS XML (NISO Z39.96) the official format for displaying all scholarly information. All updates, distributions, and integrations originate from this single, well-organized source.

Conceptual Model

In SSOT-based systems, all downstream outputs, such as PDF, HTML, ePub, and indexing feeds, are generated from a single XML instance. This eliminates inconsistencies and ensures consistent results.[1]

[1] Conceptual Model (SSOT / Single-Source Publishing) - https://en.wikipedia.org/wiki/Single-source_publishing (en.wikipedia.org) - Accessed: 19 March 2026



Operational Implications

1. **Elimination of Redundant Transformations**

   There is no need to repeatedly change formats. You only need to create XML once and reuse it.

2. **Deterministic Rendering Pipelines**

   Rendering engines always produce the same results since the input is always the same.

3. **Metadata Centralization**

   All metadata, including author affiliations, references, and financing, is stored in a single schema.[2]

4. **Auditability and Traceability**

   Version control can be implemented at the XML level, making it simple to trace changes.

[2] Operational Implications & XML Schema Enforcement - https://www.w3.org/TR/xmlschema-1/ - Accessed: 19 March 2026

SSOT Implementation Requirements

To successfully deploy SSOT, we must:

- Enforce XML production in the earliest intake step.
- Disallow PDF as an authorized source.
- Implement schema validation checkpoints.
- Ensure bidirectional sync between editorial systems and XML.

Design Constraints

- XML must precisely comply with the JATS DTD or XSD.
- All editorial activities should be schema-aware.
- Rendering layers should be stateless and idempotent.

The XML Imperative: Why PDF-Centric Workflows Fail Modern Scholarly Discovery

People continue to use PDF for historical purposes rather than for its technological superiority. The primary purpose of the PDF is to create attractive documents; however, PDFs do not perform well in today's digital world, especially when it comes to machine learning, semantic indexing and exchanging data via APIs.[3]

Structural Limitations of PDFs

PDF content is represented as follows:

- Positioned glyphs

[3] XML vs PDF in Scholarly Publishing (JATS & Machine-Readable Advantage) - https://jats.nlm.nih.gov/ - Accessed: 19 March 2026

- Layout instructions.
- embedded fonts

Content in PDFs does not include:

- Logical structure (sections and paragraphs)
- Semantic meaning (author roles, references)
- Machine-interpretable metadata

Extraction Complexity

Probabilistic models are used by tools like GROBID to infer the structure of PDFs. But:

- Accuracy varies with the layout's complexity.
- Tables and formulae typically get worse over time.
- Parsing references might lead to mistakes.

This makes things non-deterministic, which is not acceptable in production-grade pipelines.

Metadata Fragmentation

Metadata is found at various levels in traditional workflows:

Layer	Source
Submission system	User input
PDF	Embedded or inferred
Indexing feeds	Reconstructed

This leads to:

- Names of authors that do not match
- The ORCID identifiers are missing.
- Links to non-working citations.

JATS XML as a Solution

JATS (Journal Article Tag Suite) addresses these issues by ensuring:

- Explicit semantic tags
- Structure with levels.
- Standardized fields for metadata

Example: Semantic Richness

```
<contrib contrib-type="author">
  <name>
    <surname>Smith</surname>
    <given-names>Jane</given-names>
  </name>
  <contrib-id
contrib-id-type="orcid">0000-0002-1825-0097</contrib-id>
</contrib>
```

This makes it possible to:

- ORCID integration
- Author disambiguation
- Automated indexing

Discovery and Indexing Advantages

JATS XML can be ingested directly into:

- Crossref DOI services
- PubMed Central repositories
- Semantic search engines

Machine Learning Readiness

Structured XML supports:

- Named Entity Recognition (NER)
- Construction of Citation Graphs
- Integration with Knowledge Graphs

Core Technical Stack

OJS 3.x as the Orchestration Layer

Open Journal Systems (OJS 3.x) is the primary orchestration layer that controls a manuscript's entire lifecycle and acts as the control plane for all subsequent XML-centric actions. It ensures that workflow state changes, metadata normalization, and integration triggers are executed between external services.[4]

[4] Open Journal Systems (OJS) Workflow & Architecture - https://en.wikipedia.org/wiki/Open_Journal_Systems - Accessed: 19 March 2026

System Architecture

Client Layer

↓

OJS Application Layer

↓

Plugin Layer

↓

External Services

Grobid

Texture

Validation Services

Key Responsibilities

Submission Management

- Accepting files in several forms (DOCX, PDF, and LaTeX).
- Structured metadata capture (ORCID, affiliations, and funding data)
- Validation and Standardization of Author Roles

Editorial Workflow

- Asynchronous coordination of peer review
- Revision versioning with audit trail
- Managing decision states using workflow states.

Production Integration

- Enforcement of XML-first ingestion pipelines
- Synchronization of JATS nodes and database metadata

- XML artifacts are stored under version control.

Plugin-Based Extensibility

OJS allows you to add functionality using modular plugins.

- XML import and export plugin (JATS ingestion/export).
- REST API Plugin (for programmatic interaction)
- OAI-PMH Interface (obtaining metadata)

Custom plugins can add validation hooks or initiate external pipelines when a submission or publication event occurs.

Grobid Integration

GROBID is a stateless ingestion microservice that converts unstructured PDFs to TEI XML by segmenting documents and processing citations using machine learning models.

Pipeline Integration

```
PDF Input → Grobid → TEI XML → XSLT → JATS XML
```

Scaling Grobid

- Containerized deployment allows for easy horizontal scaling.
- Balancing the load over numerous instances for batch ingestion.

```
docker run -p 8070:8070 lfoppiano/grobid:latest
```

Batch Processing

```
for file in *.pdf; do
```

```
  curl -F "input=@$file"
http://localhost:8070/api/processFulltextDocument
done
```

Texture XML Editor

Texture is a browser-based editing tool that allows individuals to inspect and fix XML generated by computers.

Capabilities

- Visual structural editing with links to XML nodes
- In-line schema validation (JATS conformance).
- Provide immediate feedback on structural concerns.

Workflow Integration

- Load the JATS XML generated by the transformation layer.
- Make semantic changes to the references, sections, and metadata.
- Check for schema constraints.
- Insert normalized XML back into the OJS production line.

Transformation Layer (TEI → JATS)

The transformation layer uses XSLT-based pipelines to convert various XML formats into JATS-compliant structures.

Example Mapping

```
<xsl:template match="tei:p">
  <p><xsl:apply-templates/></p>
</xsl:template>
```

Best Practices

- A modular XSLT design that separates metadata, body, and references.
- Namespace normalization to avoid schema conflicts
- Strong error handling for missing or incorrectly formatted nodes.
- Using precompiled XSLT processors (such as Saxon) to speed up

This layer ensures that the conversion is deterministic and reproducible, which forms the foundation of the JATS-first publication workflow.

The Production Pipeline

Pre-Print Intake

The intake layer processes various sorts of author submissions and converts them into a structured, machine-readable intermediate format. This ensures that everything is identical before conversion to JATS XML. This stage ensures that the format is the same, the encoding (UTF-8), and the metadata are removed.[5]

Format	Tool
DOCX	Pandoc
LaTeX	LaTeXML
PDF	Grobid

Pandoc Conversion

```
pandoc input.docx -t jats -o output.xml
```

[5] Pre-Print Intake, JATS Conversion & Validation Pipeline - https://en.wikipedia.org/wiki/Pandoc - Accessed: 19 March 2026

Technical Considerations

- Before converting, ensure that the styles are consistent (Heading 1 →).
- Remove inline formatting artifacts, such as colors and fonts.
- Retrieve metadata that is already in the file (author, keywords).
- Make sure that the encoding and character sets are right.

Structural Tagging

After conversion, documents must be semantically structured, which requires strict adherence to the JATS hierarchy and element standards.

Core Structural Enforcement

- Section hierarchy → <sec> with nested <title>
- Figures → <fig> with <caption> and <graphic>
- Tables → <table-wrap> with structured <tbody>

Automated Tagging Rules

- Heading detection → <title> within <sec>
- Paragraph grouping → <p> with logical segmentation
- Citations → <xref ref-type="bibr"> linked to <ref-list>

Advanced Tagging Enhancements

- ORCID normalization in <contrib-id>
- Funding metadata → <funding-group>
- Inline math → MathML embedding

Validator Loops

Validation ensures that the JATS standard (NISO Z39.96) is followed exactly and that schema drift does not occur when data is modified or edited.

Validation Stack

- xmllint → syntax validation
- RELAX NG schemas → structural validation
- JATS4R recommendations → best-practice enforcement

Continuous Validation Model

```
Input → Validate → Error → Fix → Revalidate → Approve
```

CI/CD Integration

Pipelines should automatically validate:

```
xmllint --noout article.xml || exit 1
```

Technical Enhancements

- Pre-commit hooks to check XML
- Automated error reporting and diagnostics down to the line level.
- Integration with Texture to fix workflows

Multi-Format Rendering

Rendering converts certified JATS XML into representations that may be transmitted via predictable, stateless transformations.

PDF Generation

- Pipeline for XSL to FO transition

- Rendering using Apache FOP

HTML Output

```
xsltproc jats-html.xsl article.xml
```

ePub Packaging

- HTML, OPF (metadata), and Navigation (NCX)
- Packed as an EPUB container that supports ZIP

Rendering Optimization

- To speed things up, use precompiled XSLT stylesheets.
- Parallel rendering pipelines for processing batches.
- CDN-based distribution of static files.

Scalability Considerations

- Microservices for stateless rendering
- Job scheduling using queues (like RabbitMQ)
- Caching rendered outputs reduces the requirement to recalculate.

This pipeline ensures that publication outputs are high-throughput, reproducible, and standards-compliant. It serves as the operational backbone for a JATS-first system.

Interoperability Protocols

REST API

Open Journal Systems provides a RESTful API layer that allows applications to access submissions, information, and publication workflows. This API is

the primary method for connecting outside systems to the JATS-first process.[6]

Core Endpoints

```
GET /api/v1/articles
POST /api/v1/submissions
```

Authentication

- JSON Web Token (JWT) for authentication without state
- OAuth2 enables third-party integrations and delegated permission.

Technical Capabilities

- CRUD operations for articles and submissions
- Obtaining metadata in structured formats like JSON and XML.
- API triggers for workflow state changes.

Integration Use Cases

- Publishing without a head (external CMS synchronization
- Analytics pipelines (monitoring usage, citations, etc.)
- AI indexing systems, including semantic parsing and knowledge graphs

Best Practices

- Throttling and rate limiting to ensure API stability
- Versioned endpoints to ensure compatibility with older versions.
- Secure transport over HTTPS with token expiration restrictions.

[6] Interoperability Protocols (OAI-PMH Standard) - https://www.openarchives.org/OAI/openarchivesprotocol.html - Accessed: 19 March 2026

OAI-PMH

OAI-PMH provides a standardized method for gathering and distributing structured metadata to aggregators and indexing services.[7]

Supported Verbs

- Identify → repository metadata
- ListRecords → bulk metadata harvesting
- GetRecord → single record retrieval

Example

```
?verb=ListRecords&metadataPrefix=jats
```

Metadata Formats

- Dublin Core (oai_dc) ensures fundamental compatibility.
- JATS XML (jats) provides entire semantic richness.

Technical Characteristics

- Use datestamps for incremental harvesting.
- Using resumption tokens for pagination.
- Model of request-response without state.

Optimization Strategies

- Pre-cache metadata to get faster answers.
- For large datasets, use gzip to compress the responses.

[7] OAI-PMH (Open Archives Initiative Protocol for Metadata Harvesting) - https://www.openarchives.org/OAI/openarchivesprotocol.html - Accessed: 19 March 2026

- Ensure that the mapping between OAI identifiers and DOIs is always consistent.

Crossref and DOI Systems

The Crossref deposit schema is built from JATS XML, allowing you to register DOIs and make them available to everyone.[8]

Automated DOI Workflow

- Get structured metadata from JATS XML.
- Change to XML that works with Crossref (via XSLT).
- Send a deposit via the Crossref API.
- Get and record the DOI.

Example Deposit Flow

```
JATS XML → XSLT → Crossref XML → API Submission → DOI Assignment
```

Technical Considerations

- Required fields: title, contributors, and date of publication.
- Integrating citation networks through reference linking
- ORCID and adding financial metadata.

Error Handling

- Schema validation errors (XML rejection)
- Finding duplicate DOIs and resolving conflicts.
- Logging metadata inconsistencies between OJS and Crossref

[8] Crossref DOI Registration & Metadata Deposit - https://www.crossref.org/documentation/ - Accessed: 19 March 2026

Reliability Enhancements

- Unsuccessful Deposits or retry options
- Queues where a process is completed in a linear sequence, not concurrently
- Audit trail for tracking DOI lifespan.

An interoperability layer to enable real-time sharing of metadata, automated indexing and persistent identifier access to all the scholarly/broader global infrastructure (i.e., everything required for a machine-readable, scalable publication platform).

Sustainability Metrics

Cost Reduction Model

A JATS-first design saves money by eliminating unnecessary transformations and ensuring employees do not have to do anything extra throughout the manufacturing lifecycle. Automation replaces time-consuming operations such as typesetting, format conversion, and manual data correction.[9]

Automation Reduces

- Creating PDFs manually using XSL-FO and
- XML rendering from a single source allows for frequent format changes.
- Human mistakes are caused by schema-driven validation and automated workflows.

[9] Event-Driven Architecture & Scalable Systems - https://microservices.io/patterns/data/event-driven-architecture.html - Accessed: 19 March 2026

Quantitative Impact

Metric	Legacy	JATS-first
Processing time	5–10 days	1–2 days
Cost/article	High	Reduced
Error rate	High	Minimal

Technical Insight

Moving from human-dependent workflows to deterministic, machine-driven pipelines that use XML as a reusable asset across all outputs reduces expenses.[10]

Automation Strategy

Automation is enabled via a distributed, event-driven design that ensures system growth and fault tolerance.

Core Components

- Asynchronously processing tasks in a queue (RabbitMQ/Kafka).
- Container orchestration (Docker/Kubernetes) to isolate services and expand them
- Stateless microservices for parsing, testing, and rendering

Pipeline Model

```
Submission → Queue → Worker Nodes → XML Processing → Output Distribution
```

[10] Deterministic Systems & Automation (CI/CD Principles) - https://martinfowler.com/articles/continuousIntegration.html - Accessed: 19 March 2026

Technical Advantages

- Decoupled services reduce the number of bottlenecks inside a system.
- Retry mechanisms for failed jobs
- Horizontal scalability as workloads change.

Monitoring and KPIs

To keep operations running, you must be able to monitor and measure performance across the pipeline.[11]

Key Metrics

- The XML validation success rate is more than 99%.
- Processing time per article (from beginning to end)
- API response time and data capacity
- Error rates in the parsing and transformation layers

Monitoring Stack

- Prometheus collects metrics.
- Grafana (Data Visualization Dashboards)
- Centralized logging (the ELK stack).

Alerting Mechanisms

- Alerts based on thresholds for failed validation
- Monitor the queue backlog.
- Service health assessments using heartbeat endpoints

[11] Monitoring & Observability (Metrics and KPIs) - https://prometheus.io/docs/introduction/overview/ - Accessed: 19 March 2026

Scalability Model

The architecture supports horizontal scaling, ensuring that performance does not decrease as the number of submissions increases.[12]

```
Load Balancer
↓
Multiple Processing Nodes
↓
Shared Storage (XML Repository)
```

Implementation Details

- Stateless services allow you to scale up and down quickly.
- S3/MinIO distributed storage for XML persistence.
- Load balancing makes sure that duties are distributed evenly.

Long-Term Sustainability

Forward-compatible design and modular extensibility ensure that things last.

Key Principles

- Support for schema evolution (JATS updates without interrupting the pipeline).
- Backward compatibility with older versions of XML
- Upgrade to modular pipes without changing the entire system.

[12] Horizontal Scalability & Load-Balanced Architecture - https://en.wikipedia.org/wiki/Scalability - Accessed: 19 March 2026

Strategic Outcome

This strategy ensures that the publishing infrastructure remains strong, adaptable, and prepared for the future.[13] It can accommodate new standards, tools, and interoperability requirements without significantly altering its development.

Conclusion

The JATS-first design represents a significant shift for scholarly publishing platforms. Organizations can achieve new levels of efficiency in creation, distribution, and discovery by transitioning from static texts to structured, machine-readable XML.[14]

This blueprint demonstrates how to put this concept into practice by:

- Pipelines for Structured Ingestion
- Layers of automatic transformation
- Loops for Continuous Validation
- APIs and protocols that operate together

For system professionals, the most important thing to remember is that XML is not a format; rather, it is the infrastructure layer upon which modern scholarly communication must be built.

[13] Future-Proof & Scalable Architecture (Microservices Principles) - https://martinfowler.com/microservices/ - Accessed: 19 March 2026
[14] JATS XML in Scholarly Publishing - https://jats.nlm.nih.gov/ - Accessed: 19 March 2026

Zeba Academy is a specialized technical research and training initiative dedicated to the principles of Sovereign Systems Engineering. Founded by Sufyan bin Uzayr - an author and university instructor as well as Google Cloud-Certified DevOps Engineer - Zeba Academy serves as a bridge between deep academic theory and high-stakes industrial implementation.

We reject the "enshittification" of modern software. Our core mission is the promotion of Anti-Bloat Architecture through the mastery of:

- **Systems Languages:** Using Rust, Zig, and C++ to build high-performance foundations that prioritize memory safety and deterministic execution.
- **SRE & DevOps:** Professional-grade automation via Google Cloud, Terraform, and Immutable Infrastructure to eliminate manual "toil" and operational fragility.
- **High-Performance Interfaces:** Utilizing Flutter for cross-platform development to deliver near-native mobile experiences without the lag of standard web-based wrappers.
- **Lean Web Publishing:** Reclaiming WordPress and PHP by stripping away the "slop", using Redis object caching and Unix sockets to transform standard platforms into high-speed, GEO-stable engines for modern publishing.
- **Legacy Modernization:** Applying memory-safe paradigms and modern build systems to century-old computational problems and aging C codebases.

Zeba Academy doesn't just teach code; we architect reliability. By merging the analytical rigor of Historical Research with the precision of Google-Certified Cloud Engineering, we provide our "Operatives" with the directives necessary to build systems that are safe, fast, and permanent.

Website:- https://zeba.academy

زیبا

Zeba Academy

zeba.academy