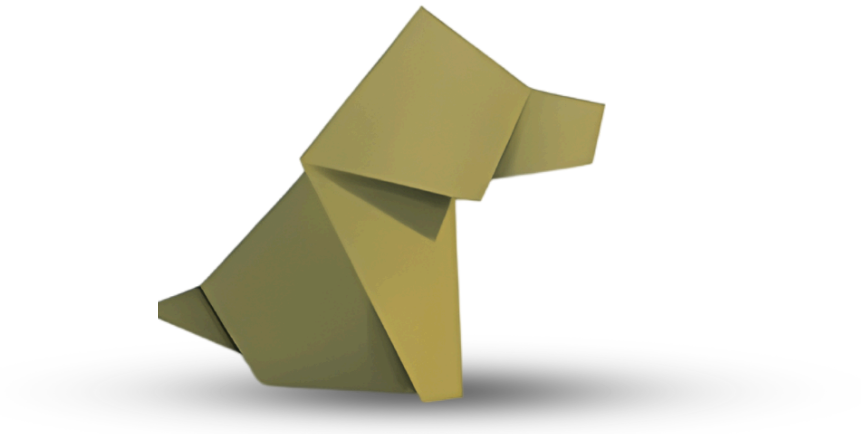
The OAPEN Schema: Standards for Monograph Interoperability

The Universal Monograph Schema: Technical Protocols for Interoperability, Metadata Integrity, and Global Discovery within the OAPEN and DOAB Networks



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

The OAPEN Schema: Standards for Monograph Interoperability

The Universal Monograph Schema: Technical Protocols for Interoperability, Metadata Integrity, and Global Discovery within the OAPEN and DOAB Networks

The Technical Challenges of the Open Access Book

Setting up reliable links between different systems that use different metadata formats is the hardest part of making an open-access monograph process. For this task, you need to be able to work with four output formats: ONIX for routes of distribution, MARC21 for library systems, XML and JSON for repositories, and REST APIs for platform ingestion. The destination systems require specialized schemas that include validation rules, controlled vocabularies, and identification requirements, making the transformation process more complex.[1]

To begin your task, establish a standard truth source. Most systems in the field rely on an internal content management system or an editorial database as their primary data source. Content management systems' data models typically focus on providing user-friendly interfaces rather than designing systems that can interact with other devices. You should start the normalization procedure for this layer as soon as possible. The system should turn contributor information into structured entities that use role codes,

[1] Overview - https://www.editeur.org/83/Overview/ - Accessed: 19 March 2026

ensure authority is maintained through ORCID ID verification, and convert license terms from plain-text labels to resolvable URIs.

Next, show relationships in a structured way using hierarchies. A book is a piece of writing that depends on other things at different levels:

- Book-level: The book's title and subtitle, as well as its ISBN, distributor, and imprint information, are listed in the bibliographic information.
- Chapter-level: The chapter information includes the title, DOI system link, page numbers, and summary text.
- Contributor-level: The contributor information must include the names of both the author and the editor, their educational backgrounds, and unique identification numbers.

The hierarchy must be constructed using either a relational or a document-based schema that maintains parent-child relationships. This approach ensures constant performance for DOI registration and ONIX composite production.

Validation should always be a part of the process. XSD and JSON Schemas must be used to validate the model, and business rules must be checked at each stage of the transformation, from CMS to intermediate XML and ONIX/MARC. Structured logging will track errors in the system, and severe faults will shut it down immediately to prevent further problems.[2]

[2] XML - https://www.w3.org/XML/ - Accessed: 19 March 2026

Instead of manual export methods, the process should leverage deterministic mapping layers such as XSLT and ETL scripts, as well as microservices. To allow for schema tracking, the mappings must be versioned.

The system must handle asynchronous scaling requirements. To construct independent processes for ingestion, transformation, and dissemination, job queues and schedulers must be used in conjunction. The system requires all jobs to be idempotent and retryable. To enable simultaneous batch processing without losing data or causing dependency blocks, the system must keep track of both audit logs and status.

The Metadata Bridge: Mapping Internal CMS Data to ONIX and MARC21

Setting up a metadata bridge begins by exporting data from your content management system (CMS) into a structured intermediate format. This is the first stage in the process. Additionally, the standard output of this format should be JSON or XML. This layer will serve as your working layer while the transformation is carried out.

The first step is to define a crosswalk document. This document serves as a mapping table connecting every CMS field to its corresponding ONIX and MARC21 equivalents. For example:[3]

- `cms.title` → ONIX `<TitleText>` → MARC21 `245$a`
- `cms.author` → ONIX `<Contributor>` → MARC21 `100`

[3] MARC21 - https://www.loc.gov/marc/bibliographic/ - Accessed: 19 March 2026

Utilizing this mapping concept to its full potential, you should put it into practice rather than merely documenting it. When working with JSON, you should use XSLT or Python transformation scripts if your pipeline is built on XML.

The second step is the normalization process. It is necessary to wipe up your data in the following steps:

- The dates should be converted into the ISO 8601 format.[4]
- It is necessary to create structured fields in order to separate the various components of the contributor(s)' names (last name, second name(s)).
- Using established terminology, such as ISO 639-2, is also recommended.[5]

The third phase entails managing controlled vocabularies. The ONIX standard requires precise codes for displaying contributor responsibilities, subject matter, and content formats. Make lookup tables that turn your internal numbers into standard codes that have already been set. These maps will be kept in configuration files that can be changed over time, and database tables can be updated without changing the code. Any value differences will be recorded by the system along with their context, while a certain fallback code keeps the process from breaking.[6]

Validation is the fourth step. The team will validate the ONIX XML and MARC21 records after they are created by comparing them to official schema

[4] ISO 8601 - https://www.iso.org/iso-8601-date-and-time-format.html - Accessed: 19 March 2026
[5] ISO 639-2 - https://www.loc.gov/standards/iso639-2/ - Accessed: 19 March 2026
[6] ONIX codelists - https://ns.editeur.org/onix/en - Accessed: 19 March 2026

definitions, including the XSDs for the ONIX and MARCXML schemas. The transformation pipeline should include automatic validators that test the system throughout the build and export process. The system will generate structured logs for all validation errors and warnings, while major problems will cause the job to fail, allowing non-essential warnings to be handled with flags.

The CMS should not save any transformed outputs. An object store or dedicated metadata repository should collaborate with versioned artifacts to provide an environment in which every export is reproducible. The system requires this separation because it provides secure reprocessing capabilities whenever mapping or standard changes occur. Instead of allowing users to change editorial records directly, the system should start regeneration jobs using stored source identifiers. The system will ensure data integrity during pipeline operations and subsequent data deliveries by employing checksums and timestamps to track all changes without sacrificing metadata correctness.

In order to make getting and sending objects faster, directory or bucket systems need to follow a pattern that includes both identifier types and format types. Output from the system can be accessed by other systems through APIs and file locations. System delivery is automated by scheduled jobs and event triggers, ensuring that all channels work together and use the same processing methods across different settings.

The OAPEN/DOAB Integration Layer

After metadata standardization is completed, the process of depositing content into repositories begins. You must construct an automatic submission layer that authenticates requests via repository APIs using REST or SWORD methods. The method requires you to bundle metadata and files into compliant payloads that include all required fields and identifiers. When the system goes down for a short time, it needs retry mechanisms that retain all response data for future use. Along with getting repository identification numbers, the system must check its replies to determine whether the data processing went well. The system must send its documents via asynchronous queues, allowing deposits to proceed without disrupting either upstream or downstream distribution activities.

Automating Deposit via SWORD or SFTP Protocols:

- Use SWORD if the repository accepts API-based submissions. The client should issue POST requests with ZIP packages including XML and other files.[7]
- SFTP should be used when several files need to be uploaded simultaneously. The system should be configured to move files to predefined repository directories at regular intervals.[8]
- Structured Archives should include not only monographs but also a variety of digitally valuable objects:

[7] SWORD - https://swordapp.github.io/SWORDv2-Profile/SWORDProfile.html - Accessed: 19 March 2026
[8] OpenSSH Manual Pages - https://www.openssh.org/manual.html - Accessed: 19 March 2026

    - Metadata XML
    - PDF or EPUB files.
    - Supporting assets (cover photos and extra materials)

- One solution is to use a queue to process deposits asynchronously, freeing up your processing line.
- Create failover conditions to handle any type of failure, including faulty HTTP responses.

Compliance with the "OAPEN Deposit Service" Standards:

- Before submitting your XML, ensure that it meets the OAPEN schema criteria.
- The system requires all necessary information to be entered, including the title, contributors, ISBN, abstract, license, and publisher sections.[9]
- The approach demands that license information be encoded using Creative Commons URLs rather than plain text.
- The system must generate checksums using SHA-256 and other methods for each submitted file and verify them after they are uploaded.
- The system should assess OAPEN validation reports and record any mistakes that need to be addressed.

The system must track all repository deposits using comprehensive status monitoring. The system must define various deposit states, such as pending,

[9] Publishers - https://www.doabooks.org/en/publishers - Accessed: 19 March 2026

submitted, failed, and accepted, and all status changes require the system to save timestamped records, as well as response codes and payload information, into your database. The system supports real-time monitoring, auditing, and troubleshooting.

Version control and change detection must be built into the system using hash and timestamp comparisons. This way, only information and files that have been changed can be resubmitted. The system has to keep operational idempotence, which means it can't do the same thing twice. It also has to keep historical data safe and make sure that repository records are always the same across multiple submission periods and automated processes.

Persistent Identification (PID) Strategy

Before deploying a PID system, organizations must determine their identification-use policies. Rather than allowing manual assignments, your pipeline should automate identifier assignment with event-driven triggers and validation checkpoints.

The process of giving an ISBN to a book begins when the title is ready for publishing. This identifier must be propagated throughout all system outputs; your CMS should store it. The same ISBN must be used for ONIX, MARC21, and repository metadata. To provide traceability and eliminate identifier conflicts across different systems and interconnections, the system requires uniqueness restrictions to be enforced, as well as checksum format validation and event recording for assignment activities.[10]

[10] ISBN - https://www.isbn-international.org/ - Accessed: 19 March 2026

Each chapter of the document requires a separate DOI assignment. The automated registration system will handle the generation of DOIs:

1. The system generates DOI suffixes using a predetermined design process.
2. The system must submit metadata to Crossref and DataCite via their application programming interfaces.[11]
3. The system must store the obtained DOI information in your content management system.

The DOI metadata must match your internal records, which you keep as a necessity. When the document's title or contributors change, utilize the API to update the DOI metadata.[12]

You must use ORCID authentication for all contributor identification operations. Authors should be able to link their ORCID IDs when submitting their work. The system will save these identifiers for use in future metadata exports.[13]

You must devise a technique for resolving disagreements that occur in your job. Every DOI must point to a reliable landing page. You must build redirects that will keep DOIs operational when your URLs change.

To manage relationships, link identifiers internally:

- ISBN → book

[11] DataCite - https://support.datacite.org/docs/api - Accessed: 19 March 2026
[12] API - https://api.crossref.org/swagger-ui/index.html - Accessed: 19 March 2026
[13] Integration Guide - https://info.orcid.org/documentation/integration-guide/ - Accessed: 19 March 2026

- DOI → chapter
- ORCID → contributor

Arrange the above relationships so they can be exported to ONIX and repository XML.

Lastly, make checks for truth. Before you publish, make sure that:

- All of the DOIs work properly.
- Please use valid ORCID IDs.
- ISBNs work the same way on all computers.

This makes sure that the identifying information stays the same all the way through the delivery network.

Distribution Dynamics: Managing High-Standard Digital Pipelines

It is necessary for you to distribute the content to external platforms once the repository has been fully deposited. The pipeline must be able to process a variety of formats and deliver content to multiple distinct endpoints for the system to function properly.

Key Components of High-standard Digital Pipelines:

- Alternatively, a workflow engine (such as Airflow) could be used as an intermediary for job management between the conversion and delivery phases.[14]

[14] Apache Airflow - https://airflow.apache.org/docs/ - Accessed: 19 March 2026

- Convert source files to the needed formats:
    - PDF documents generate unchanging page designs.
    - EPUB documents allow users to display content in a flexible manner.[15]
    - The XML format allows automated system processing of ordered data elements.
- Create ONIX streams for platforms requiring commercial metadata supply.
- Use platform APIs to send metadata and content information via direct integration.
- The system requires a uniform logging architecture that tracks all delivery attempts and results.
- The system uses parallel processing to manage multiple titles simultaneously, thereby increasing operational efficiency.

Each platform has its own set of requirements. Some platforms require ONIX feeds, while others support direct API consumption. Create platform-specific adapters rather than a general solution that only supports one format.

Delivery status must be tracked across all platforms. A title can be accepted by one platform but denied by another. The system must detect all variances and offer reprocessing alternatives.

[15] EPUB - https://www.w3.org/publishing/epub3/ - Accessed: 19 March 2026

The system requires updates by detecting metadata and file alterations, which trigger redistribution. Match current records to existing records using identifiers to execute updates rather than new entries.

The system requires performance monitoring. The system must track three performance indicators: processing time, success rate, and error frequency. The data will help you enhance your pipeline by identifying operational issues.

Quality Assurance: Peer Review Transparency and Licensing Validation

You ought not to use quality assurance as a final review step; rather, you should integrate it throughout your entire pipeline. The first step is to establish validation criteria applicable to all content and metadata.

Your metadata schema should contain structured fields that require the following information:

- The first component involves documentation of the review type, which might be single-blind or double-blind.
- The second need is documentation of the review status, which must be indicated as reviewed or not reviewed.
- The next step is to identify reviewer positions, which should be provided as needed.

The fields must be exported into ONIX and repository XML so that indexing systems can access them.

The license regulations require the use of machine-readable forms. The license must be saved as a URI link that includes the Creative Commons link, and it must appear in all output documents. Text descriptions should not serve as the sole source of information.[16]

All information output from the system must be schema-validated. The system should use automated tools to check ONIX, MARC21, and repository XML documents against their respective formats. That's when the pipeline will stop, and the system will save the mistake.

The following elements of the document need to be verified for accuracy:

- The first check makes sure that the metadata information fits the content files that go with it.
- The second check needs to make sure that all of the identifiers work correctly.[17]
- For the third check, it's necessary to make sure that all of the needed fields have valid data entered.

The entire testing process must be automated via script development, with the scripts linked to your continuous integration system. Before any updates can be published to users, the system must first run validation tests.

The system must generate audit logs that track all system actions. The system must record all events that occur during metadata creation, as well as subsequent modifications, validation processes, and information delivery. This technique facilitates both debugging and compliance requirements.

[16] Licenses List - https://creativecommons.org/licenses/ - Accessed: 19 March 2026
[17] DOI Foundation - https://www.doi.org/ - Accessed: 19 March 2026

The sophisticated configuration will utilize anomaly-detecting technology. The system must identify duplicate titles, missing identification numbers, and instances of mismatched subject classification.

Directly integrating QA into your workflow process will ensure that your published monograph meets compliance criteria while remaining accessible across all devices.

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