PROVISIONAL PATENT APPLICATION

Title: Browser-Native Portable Artifact Container and Generation System

Inventor: Nicholas Laudani

Address: 29 Beech Glen St, Roxbury, MA 02119, United States

SPECIFICATION

Browser-Native Portable Artifact Container and Generation System

Provisional Specification

1. Field

The disclosure relates to computer-implemented systems for generating, packaging, publishing, rendering, and verifying digital artifacts. More particularly, the disclosure concerns systems that transition a composite digital work from an editable authoring state into a locked self-rendering artifact state that is browser-renderable as a portable network object. Alternate title =A deterministic portable artifact generator produces a discoverable, platform-agnostic artifact whose identity may be hashed or anchored to blockchain records without requiring the artifact itself to reside entirely on-chain, while remaining uniformly accessible without a specialized reader and preserving the artifact as the primary object of experience, display, and public record

2. Background

Many digital works are distributed through streams, feeds, dashboards, token pointers, cloud records, or downloadable files that depend on a separate application, platform, marketplace, wallet flow, or proprietary viewer. In such arrangements, the public-facing record is often distinct from the artifact itself, leaving creators with a link, token record, or hosted reference rather than a durable browser-renderable object.

Conventional token-centric approaches frequently place the token, marketplace entry, or external record in the primary position while the underlying media and presentation remain referenced elsewhere. Other systems rely on static files that do not preserve integrated media, presentation logic, and linked metadata as a unified browser-facing artifact. There remains a need for a system that directly generates the artifact itself as the primary object, preserves it in a locked post-finalization state, and permits optional verification without making tokenization or ledger participation a prerequisite to artifact existence or display.

In some embodiments, the system may interface with or publish artifact-related data to one or more registry systems, discovery layers, or distributed ledger environments. By way of example and not limitation, current implementations may utilize platforms such as OnlyUno as a registry or discovery layer through which artifact identifiers, metadata, or associated records may be indexed, discovered, or accessed. Such registry systems may further support presentation within galleries, collections, or other discovery interfaces, and may facilitate encoding, referencing, or linking of artifact-related data. However, such use of any specific platform, including OnlyUno, is illustrative of present implementations and is not required. The system is not limited to any particular registry, platform, or discovery mechanism, and equivalent or alternative systems may be used. While such registry or distributed ledger interaction may support discoverability, presentation, or verification within the broader system, such interaction remains optional and non-constitutive of the artifact itself. The artifact container remains directly renderable in a standard browser environment independent of any registry, blockchain interaction, token ownership, or external resolution mechanism.

3. Summary of the Disclosure

The present disclosure provides a computer-implemented system and method for generating a self-rendering digital artifact as a portable browser-native object. In one aspect, a computing system receives defined inputs including media, metadata, and presentation-state information, maintains an editable browser-based composition responsive to those inputs, receives a finalization instruction, captures a finalized state of the composition, and transforms that finalized state into a canonical export package.

The export package can include a canonical HTML payload, structured metadata, and associated assets arranged in a hosted or portable container structure. The same HTML document can serve as both artifact payload and primary viewer. Export-state markers and lock-state logic can distinguish the finalized artifact from the authoring environment while preserving viewer-side functionality for media playback, layout, links, and presentation behavior.

In some embodiments, the artifact is retrievable at a canonical network location such as a permanent URL or finals-path location. In some embodiments, identity attributes are derived from defined inputs, finalized state, canonical export structure, content-derived data, or combinations thereof. In other embodiments, backend addressing identifiers and user-facing artifact identifiers differ while preserving continuity between the finalized artifact and associated verification records.

The artifact may optionally be associated with registry, database, or public-ledger infrastructure for verification, timestamping, or integrity confirmation. However, such verification infrastructure is supportive and separable. The artifact itself is primary and remains renderable without requiring token ownership, a token contract, a separate reader application, or a specialized viewer.

4. Positive Distinction from Token-Centric and Reader-Dependent Systems

Unlike systems in which a tokenized record merely refers to an external digital item, the disclosed system generates and publishes the browser-renderable artifact itself as the primary object. Unlike file types that commonly require a dedicated reader or detached host logic for meaningful use, the

disclosed artifact is renderable in a standard browser as a self-rendering network object that carries its own presentation behavior.

The system therefore reduces intermediary steps by enabling direct creation, finalization, packaging, publication, and browser-native rendering of the artifact before, or independently of, any optional registry or ledger anchoring. Verification may be added, but verification does not constitute the artifact and is not required for ordinary retrieval or rendering of the artifact.

5. Brief Description of Example Figures

Figure 1 illustrates an overall system architecture including an authoring interface, finalization pipeline, artifact package generator, publication path, optional registry services, and optional public-ledger services.

Figure 2 illustrates an example lifecycle from editable authoring state to preview state to locked exported artifact state to browser retrieval at a canonical URL.

Figure 3 illustrates an example artifact container layout including an HTML payload, one or more metadata files, and role-mapped assets stored under a stable path structure.

Figure 4 illustrates an example export method in which a finalized browser composition is transformed into a portable artifact package.

Figure 5 illustrates an example lock mechanism employing exported-state markers and suppression of authoring controls while preserving viewer behavior.

Figure 6 illustrates an example verification pathway employing registry ingest, redirect handoff, artifact hashing, and optional ledger recording.

Figure 7 illustrates example identity layers including user-facing artifact identity, storage-path identity, and optional verification identity.

6. Definitions

Artifact. A composite digital object comprising media, metadata, and presentation structure intended to be retrieved and rendered as a cohesive browser-facing unit.

Artifact container. A portable browser-renderable package that stores or references an artifact payload, metadata, and associated assets in a stable arrangement.

Browser-native. Renderable in a conventional browser without requiring a specialized native application or proprietary viewer dedicated to the artifact type.

Canonical URL. A stable network location designated for retrieval of the finalized artifact or a primary representation of the finalized artifact.

Finalized artifact state. A non-editable or constrained state in which authoring controls are hidden, disabled, removed, or otherwise suppressed while artifact viewing behavior is preserved.

Optional verification layer. A registry, database, ledger, blockchain, or other verification mechanism that may be associated with the artifact without being necessary for artifact existence, retrieval, or rendering.

7. Detailed Description

In one embodiment, a browser-based authoring interface maintains an editable composition of the artifact. The composition can include text, images, audio, video, links, styling, layout state, helper logic, metadata fields, access-state information, and branding or publication information.

Upon receipt of a finalization command, the system stabilizes the composition, captures a defined final state, updates export metadata, suppresses or disables authoring controls, and assembles a package containing an HTML artifact payload, structured metadata, and associated assets. In some embodiments, the system inserts one or more exported-state indicators into the HTML or associated metadata to distinguish the finalized artifact package from the editable composition.

In one implementation path, a finalization control triggers browser-side logic that freezes an editable composition, builds an exportable HTML package, and transmits a multipart payload containing the artifact package to an upstream publication endpoint. The upstream endpoint can return a final URL, redirect URL, registry receipt, hash value, or related publication data. The returned data can be used for optional gallery handoff, registry ingest, or public-ledger recording.

In some embodiments, the HTML payload and the primary artifact viewer are the same exported HTML document. In some embodiments, the package further includes metadata files such as meta.json or metadata.json. In some embodiments, associated assets are mapped to semantic roles and exported according to role-based or role-mapped asset paths. In some embodiments, the artifact remains renderable after transfer to a different host or storage location, provided the package structure is preserved.

The artifact package may be published under a finals path including a slug, a content-derived identity, a user-facing identifier, or another path component. The user-facing identifier need not match backend addressing identifiers so long as continuity between the artifact, its publication path, and any verification data is preserved.

In some embodiments, verification services receive an artifact hash, a final URL, a registry payload, or combinations thereof. However, the artifact does not depend on those services to exist, render, or function as an artifact. In some embodiments, the system retrieves local or associated metadata after export and modifies viewer presentation based on status, expiration, or access conditions without reintroducing authoring controls.

The disclosed architecture also supports use cases in which the finalized artifact can contain integrated media, metadata, outbound references, and interactive presentation logic. Accordingly, the system supports a browser-viewable object that is not merely a passive file or token reference, but a portable rendered artifact with stable identity and post-finalization integrity.

8. Example Embodiments

Media artifact embodiment. The artifact includes cover imagery, audio, video, creator text, links, and associated metadata arranged into a browser-native promotional, archival, or publication presentation.

Document artifact embodiment. The artifact includes long-form text, exhibits, embedded media, metadata, and outbound references arranged into a browser-native publication or record package.

Commemorative artifact embodiment. The artifact includes images, narrative text, event metadata, and optional multimedia arranged into a persistent browser-viewable life-event artifact.

Verification-light embodiment. The artifact is finalized, exported, hosted, and retrieved without any blockchain interaction.

Verification-enhanced embodiment. The artifact is additionally associated with registry ingest and public-ledger recording for verification.

Portable package embodiment. The artifact package is stored as an HTML payload plus metadata files and assets such that the package can be transferred or mirrored while preserving browser-renderable behavior.

9. Abstract

A computer-implemented system generates a browser-native portable artifact from defined media, metadata, and presentation-state inputs. A finalized state of a browser-based composition is captured and transformed into a self-rendering export package comprising an HTML payload, structured metadata, and associated assets. The exported artifact is placed into a locked non-editable state while preserving viewer-side rendering behavior and can be published at a canonical network location as the primary browser-renderable object. The artifact may include media, metadata, interactive presentation logic, and outbound references, and may optionally be associated with registry or public-ledger verification records without making such verification constitutive of the artifact itself.

CLAIM SET (NON-LIMITING)

1. A computer-implemented method comprising: receiving defined inputs including media, metadata, and presentation-state data; maintaining an editable browser-based composition responsive to the defined inputs; receiving a finalization instruction; capturing a finalized state of the browser-based composition; generating, from the finalized state, a browser-native artifact package including an HTML artifact payload, structured metadata, and one or more associated assets; inserting one or more exported-state indicators; and publishing the artifact such that it is directly renderable by a standard browser as a locked self-rendering artifact.

2. The method of claim 1, wherein the HTML artifact payload and a primary artifact viewer are the same exported HTML document.

3. The method of claim 1, wherein the browser-native artifact package includes at least one metadata file.

4. The method of claim 1, wherein associated assets are mapped to semantic roles.

5. The method of claim 1, wherein exported-state indicators identify the artifact as finalized.

6. The method of claim 1, wherein authoring controls are suppressed.

7. The method of claim 1, further comprising modifying presentation based on metadata.

8. The method of claim 1, further comprising publishing to a canonical path.

9. The method of claim 1, wherein the artifact is accessible via a stable URL.

10. The method of claim 1, further comprising generating a redirect URL.

11. The method of claim 1, further comprising transmitting a registry payload.

12. The method of claim 1, further comprising transmitting a hash to a ledger.

13. The method of claim 1, wherein the artifact remains portable across hosts.

14. The method of claim 1, wherein the artifact includes integrated media and logic.

15. A system comprising one or more processors and memory configured to perform the method of claim 1.

16. The system of claim 15, wherein identity attributes are derived from inputs or finalized state.

17. The system of claim 15, wherein backend and user-facing identifiers differ.

18. The system of claim 15, wherein the HTML payload serves as both viewer and artifact.

19. The system of claim 15, wherein the artifact is retrievable without token access.

20. The system of claim 15, wherein lock-state logic distinguishes finalized state.

21. The system of claim 15, wherein a ledger is optional.

22. The system of claim 15, wherein rendering behavior is preserved.

23. The system of claim 15, wherein the artifact is platform independent.

24. A non-transitory computer-readable medium storing instructions that cause processors to perform the method of claim 1.

25. The medium of claim 24, wherein metadata and role-mapped assets are included.

26. The medium of claim 24, wherein authoring controls are suppressed.

27. The medium of claim 24, wherein the artifact is associated with a registry or ledger.

28. The medium of claim 24, wherein the artifact is published at a stable URL.

FIGURE SHEETS

SpinStream Provisional Patent
Example Figure Sheets

Illustrative black-and-white figures for filing with the provisional specification

These figures are illustrative and are intended to support the written description of the browser-native portable artifact container and generation system.

FIG. 1 — Example overall system architecture including authoring interface, finalization pipeline, artifact package generator, publication path, optional registry service, optional public ledger service, and browser retrieval at a canonical URL.

FIG. 2 — Example lifecycle from editable authoring state to preview state to finalized locked state to published artifact state and browser retrieval at a canonical URL.

FIG. 3 — Example artifact container layout including HTML payload, metadata files, role-mapped assets, export-state markers, and canonical path identity.

FIG. 4 — Example export method in which defined inputs are transformed from an editable browser composition into a portable self-rendering artifact package with optional verification transmission.

SpinStream Artifact Container Spec

Artifact Container Definition

This definition is derived from the implemented export/mint pipeline in `script.js`, the mint proxy route in `stripe-checkout/index.js`, and ledger recording workers.

1) Container structure

The frontend export pipeline builds a multipart payload containing the artifact container components:

- `index.html` (canonical exported HTML snapshot)
- `html` (same HTML, backward-compatibility field)
- `meta.json` (attached under `file`)
- `metadata.json` (same JSON attached under `metadata`)
- `asset:<role>:<fileName>` for each uploaded/exported asset
- optional `ownerKey`

This is assembled in `freezeAndUpload()` and POSTed to `/api/mint`.

On-storage layout at finals
The expected minted layout is:

```text
/finals/{slug}/
  index.html
  meta.json
  assets/{...}
```

Notes from implementation:
- `script.js` treats the returned URL as `finalUrl`/`mintedUrl` and derives cover at `/finals/{slug}/cover.png`.
- The repo's `/api/mint` route is a proxy to an external mint worker (`MINT_WORKER_ORIGIN`), so exact backend write semantics for *all* files are external, but frontend and docs consistently operate on `index.html + meta.json + assets/*`.

2) Artifact identity mechanism

Artifact identity in this codebase has three distinct parts:

1. **Mint ID (frontend identity token)**
   - Primary source: `mintIdInput` (user-provided, normalized/validated).
   - Fallback generation: `generateMintId()` uses timestamp components + `Math.random()` suffix.
   - Persisted in DOM attributes (`body.dataset.mintId`) via `ensureMintMetadata()`.

2. **Final URL / slug (backend identity path)**
   - Frontend receives minted URL from mint response fields (`finalUrl`, `mintedUrl`, etc.).

   - Slug generation itself is **not** implemented in this repository; it occurs in the external mint worker behind `/api/mint`.

3. **Timestamp metadata**
   - `ensureMintMetadata()` sets `mintedAt` from `mintDateInput` if valid, otherwise current time.
   - `mintDate` is also captured in mint metadata fields and meta tags.

3) Deterministic export mechanism

Export is deterministic relative to finalized DOM state + selected files at mint time:

- `buildInlineExportHtml()` explicitly snapshots a cloned sanitized DOM (`document.documentElement.cloneNode(true)` then `outerHTML`).
- It injects explicit exported-state marker(s):
  - `data-exported="true"`
  - `data-mint-state="minted"`
- It applies sanitization guards (`assertNoHelperIdentifiers`, `verifyExportSanitization`) and aborts mint on sanitization failure.
- It hides editor-only controls and enforces exported view behavior via injected export-only CSS and metadata tags.

Determinism boundaries:
- Deterministic container assembly is evident in frontend export code.
- Mint ID fallback uses randomness (`Math.random()`), so ID creation is not purely deterministic when auto-generated.
- Backend slug derivation and any server-side transforms are external to this repo.

4) Metadata schema

Metadata exists in three layers:

1. **Runtime metadata object** from `getMintMetadata()`:
   - `mintId`, `title`, `artist`, `description`, `mintDate`, `type`, `tags`, `tagsRaw`, `license`, `links`, `coverImageFile`, `coverImageName`.

2. **Embedded HTML metadata** in exported `index.html` via `addExportMetadataToClone()` meta tags:
   - `spinstream:mint-id`
   - `spinstream:mint-title`
   - `spinstream:mint-artist`
   - `spinstream:mint-date`
   - `spinstream:mint-description`
   - `spinstream:mint-type`
   - `spinstream:mint-tags`
   - `spinstream:mint-license`
   - `spinstream:mint-links`

3. **Mint worker payload metadata JSON** (`meta.json` and `metadata.json` carry same structure):

```json
```

```
{
  "artistName": "...",
  "mintMetadata": { "...": "..." },
  "assetRoles": { "fileName": "role" },
  "exportAssetPaths": { "role": "assets/fileName" }
}
```

Additionally, exported `index.html` includes a lock-gate script that fetches `./meta.json` and checks `status`/`expiresAt` to determine lock-screen rendering.

5) Ledger integration

Ledger anchoring path expects:

```json
{ "artifactHash": "0x...64hex", "mintedUrl": "https://..." }
```

Two worker surfaces implement record calls:

- `workers/spinstream-ledger/src/index.js`
  - accepts `/api/record-ledger`, `/record-ledger`, `/ledger/record-ledger`, `/`
  - validates required fields
  - executes `contract.record(artifactHash, mintedUrl)`
  - returns tx hash, block number, and ledger timestamp from Polygon block data

- `stripe-checkout/index.js` (`/api/record-ledger`)
  - validates hash/url format
  - executes `contract.record(artifactHash, mintedUrl)` with timeout handling
  - returns chain metadata and tx details

Important implementation fact:
- Artifact hash **generation** is not present in `script.js` mint/export path in this repository snapshot. The ledger workers consume `artifactHash`; they do not derive it from artifact bytes locally.

6) Viewer structure (`index.html` as artifact viewer)

The finalized artifact viewer is `index.html` served from the minted finals URL.

Viewer behavior comes from export output + runtime script behaviors:

- Export enforces artifact mode with `data-exported="true"` and hides creation UI.
- Export preserves/rewrites asset references to stable paths (e.g., `assets/...` for uploaded media, absolute paths for system assets as needed).
- Export keeps script wiring so the page remains interactive as a viewer, while export markers and CSS suppress editor controls.
- Lock-gate logic in exported HTML can replace body content with an expiration screen using metadata from `meta.json`.

Net effect:
- `index.html` is both the artifact payload and the primary artifact viewer, with state controlled by explicit export markers and metadata, not editor-only mode assumptions.

Claim Development Set (Reordered)

Method Claims

1. Claim Development Set

2. Browser-Native Portable Artifact Container and Generation System

3. Independent Claims

4. A computer-implemented method comprising: receiving defined inputs including media, metadata, and presentation-state data for a composite digital artifact; maintaining an editable browser-based composition responsive to the defined inputs; receiving a finalization instruction; capturing a finalized state of the browser-based composition; generating, from the finalized state, a browser-native artifact package including an HTML artifact payload, structured metadata, and one or more associated assets; inserting one or more exported-state indicators that distinguish the browser-native artifact package from the editable browser-based composition; and publishing or storing the browser-native artifact package such that the HTML artifact payload is retrievable at a canonical network location and renderable by a standard browser as a locked self-rendering artifact.

5. A system comprising: one or more processors and memory storing instructions that, when executed, cause the system to: receive artifact inputs including media, metadata, and presentation-state information; maintain a browser-based authoring state corresponding to a composite digital artifact; transform the browser-based authoring state into a canonical export state responsive to a finalization instruction; package the canonical export state into a portable browser-renderable artifact container including an HTML payload, metadata, and associated assets; suppress authoring controls in the portable browser-renderable artifact container while preserving viewer functionality; and optionally associate the portable browser-renderable artifact container with a registry record or public-ledger verification record.

6. A non-transitory computer-readable medium storing instructions that, when executed by one or more processors, cause the processors to: capture a finalized state of a browser-based digital composition; generate a self-rendering artifact package from the finalized state; mark the self-rendering artifact package as exported; and render the self-rendering artifact package in a browser without requiring a specialized viewer, token contract, or proprietary reader application.

7. Dependent Claims

8. The method of claim 1, wherein the HTML artifact payload and a primary artifact viewer are the same exported HTML document.

9. The method of claim 1, wherein the browser-native artifact package further includes at least one metadata file selected from meta.json and metadata.json.

10. The method of claim 1, wherein associated assets are mapped to semantic roles and exported according to role-mapped asset paths.

11. The method of claim 1, wherein the exported-state indicators include a body attribute, metadata field, or script-accessible marker identifying the artifact package as exported or minted.

12. The method of claim 1, wherein authoring controls are disabled, hidden, removed, or marked read-only in the browser-native artifact package.

13. The method of claim 1, further comprising retrieving local or associated metadata after export and modifying viewer presentation based on status, expiration, or access-state data.

14. The method of claim 1, further comprising publishing the browser-native artifact package to a finals path including a slug, content-derived identifier, or other path identity component.

System Claims

15. The method of claim 1, wherein the canonical network location is a stable URL designated for retrieval of the finalized artifact.

16. The method of claim 1, further comprising generating a redirect URL containing artifact metadata and a final artifact URL for handoff to a gallery, registry, or publication service.

17. The method of claim 1, further comprising transmitting a registry payload to an external ingest service.

18. The method of claim 1, further comprising transmitting an artifact hash and a final URL to a public-ledger recording service.

19. The method of claim 1, wherein the browser-native artifact package remains renderable after transfer to a different host or storage location while preserving package structure.

20. The method of claim 1, wherein the artifact includes integrated media, metadata, outbound references, and interactive presentation logic.

21. The system of claim 15, wherein identity attributes are derived from defined inputs, finalized state, canonical export structure, content-derived data, or combinations thereof.

22. The system of claim 15, wherein backend addressing identifiers differ from user-facing artifact identifiers while preserving artifact continuity.

23. The system of claim 15, wherein the HTML payload serves as both stored artifact payload and primary browser viewer.

Computer-Readable Medium Claims

24. The system of claim 15, wherein the artifact container is retrievable at a canonical URL without requiring token-based access rights to render the artifact.

25. The system of claim 15, wherein the artifact container includes lock-state logic that distinguishes a finalized artifact state from an editable authoring state.

26. The system of claim 15, wherein the optional public-ledger verification record is supportive of integrity verification and is not constitutive of the artifact's existence or display.

27. The system of claim 15, wherein the artifact container is configured to preserve viewer-side rendering behavior for audio, video, links, and layout after finalization.

28. The system of claim 15, wherein the artifact container is publishable as a portable browser-native object that is platform-independent for ordinary retrieval and rendering.

29. The computer-readable medium of claim 24, wherein the instructions cause the processors to include a metadata file and one or more role-mapped assets in the self-rendering artifact package.

30. The computer-readable medium of claim 24, wherein the instructions cause the processors to suppress authoring controls while preserving viewer-side media playback and link behavior.

31. The computer-readable medium of claim 24, wherein the instructions cause the processors to associate the self-rendering artifact package with a registry record, a database record, or a public-ledger record for verification.

32. The computer-readable medium of claim 24, wherein the instructions cause the processors to publish the self-rendering artifact package at a stable URL corresponding to a finalized artifact identity.

33. Claim Positioning Notes

34.

35.

36.

SpinStream Provisional Patent
Example Figure Sheets

Illustrative black-and-white figures for filing with the provisional specification

These figures are illustrative and are intended to support the written description of the browser-native portable artifact container and generation system.

FIG. 1 — Example overall system architecture including authoring interface, finalization pipeline, artifact package generator, publication path, optional registry service, optional public ledger service, and browser retrieval at a canonical URL.

FIG. 1

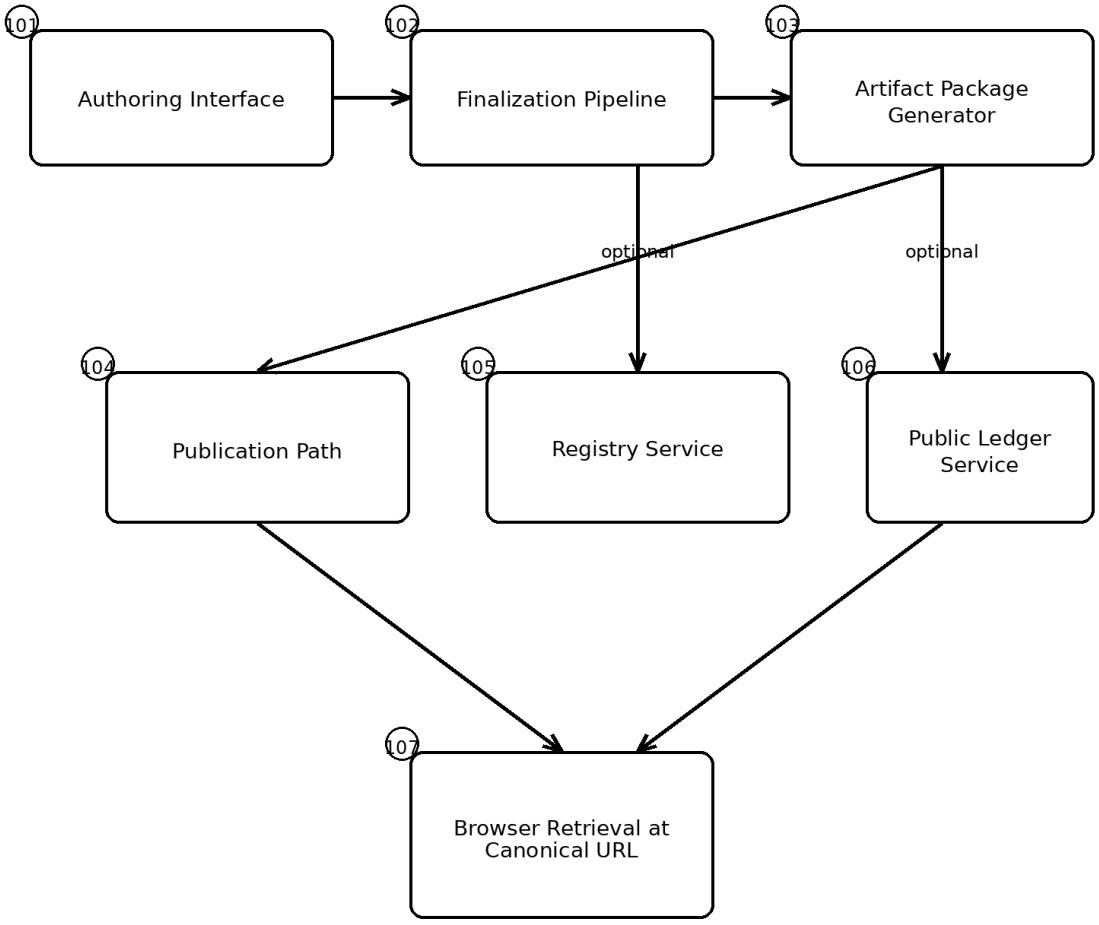

101
Authoring Interface
102
Finalization Pipeline
103
Artifact Package Generator
optional
optional
104
Publication Path
105
Registry Service
106
Public Ledger Service
107
Browser Retrieval at Canonical URL

FIG. 2 — Example lifecycle from editable authoring state to preview state to finalized locked state to published artifact state and browser retrieval at a canonical URL.

FIG. 2

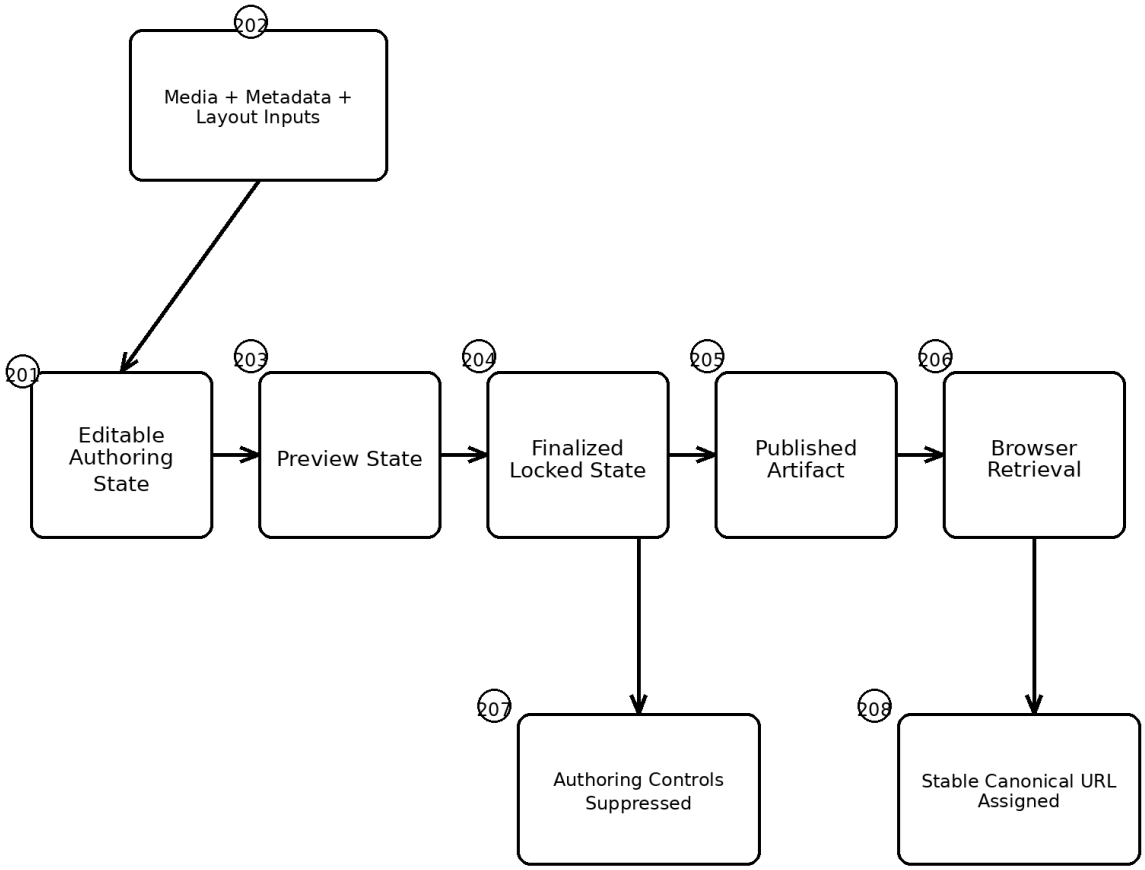

202
Media + Metadata + Layout Inputs
201
Editable Authoring State
203
Preview State
204
Finalized Locked State
205
Published Artifact
206
Browser Retrieval
207
Authoring Controls Suppressed
208
Stable Canonical URL Assigned

FIG. 3 — Example artifact container layout including HTML payload, metadata files, role-mapped assets, export-state markers, and canonical path identity.

FIG. 3

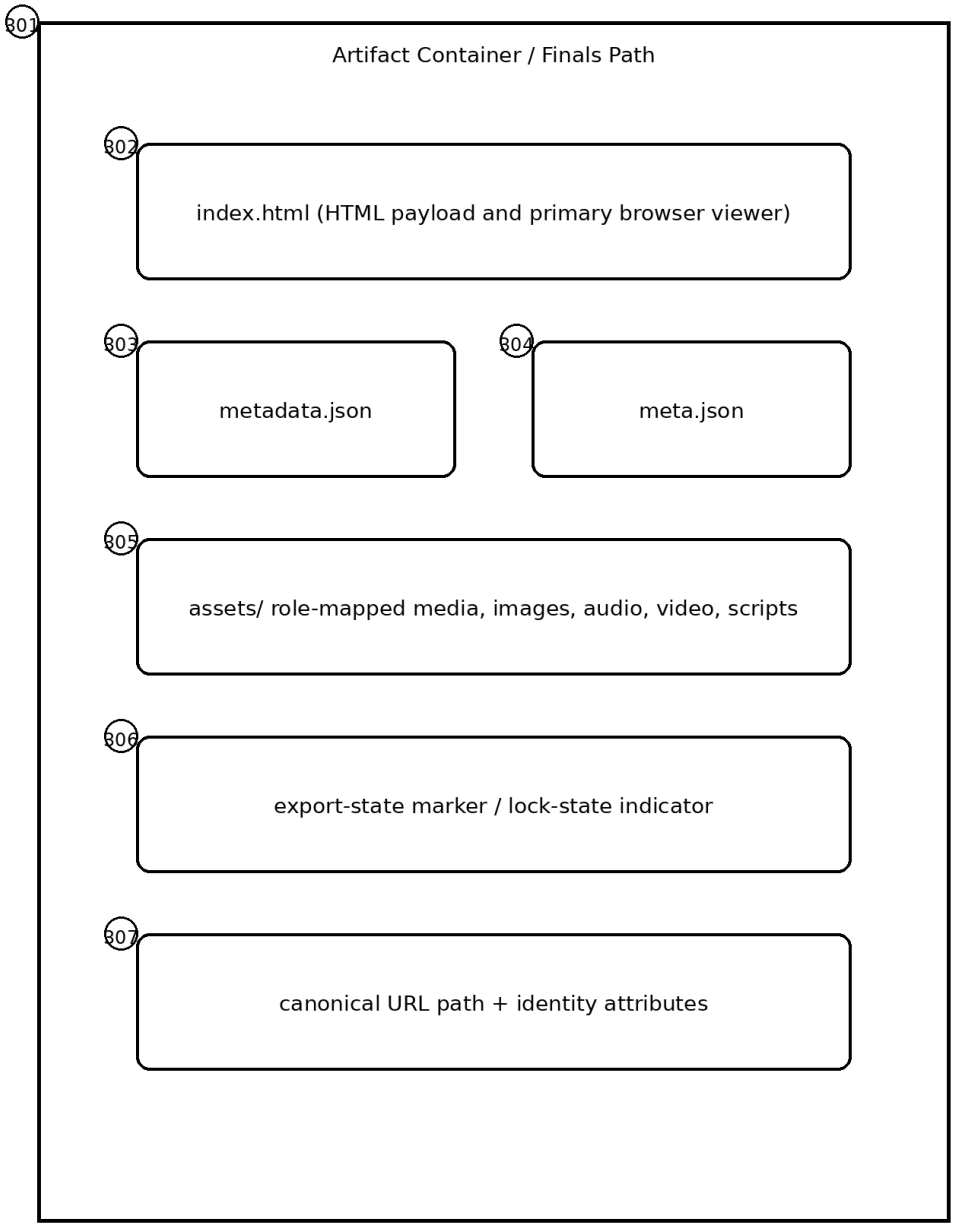
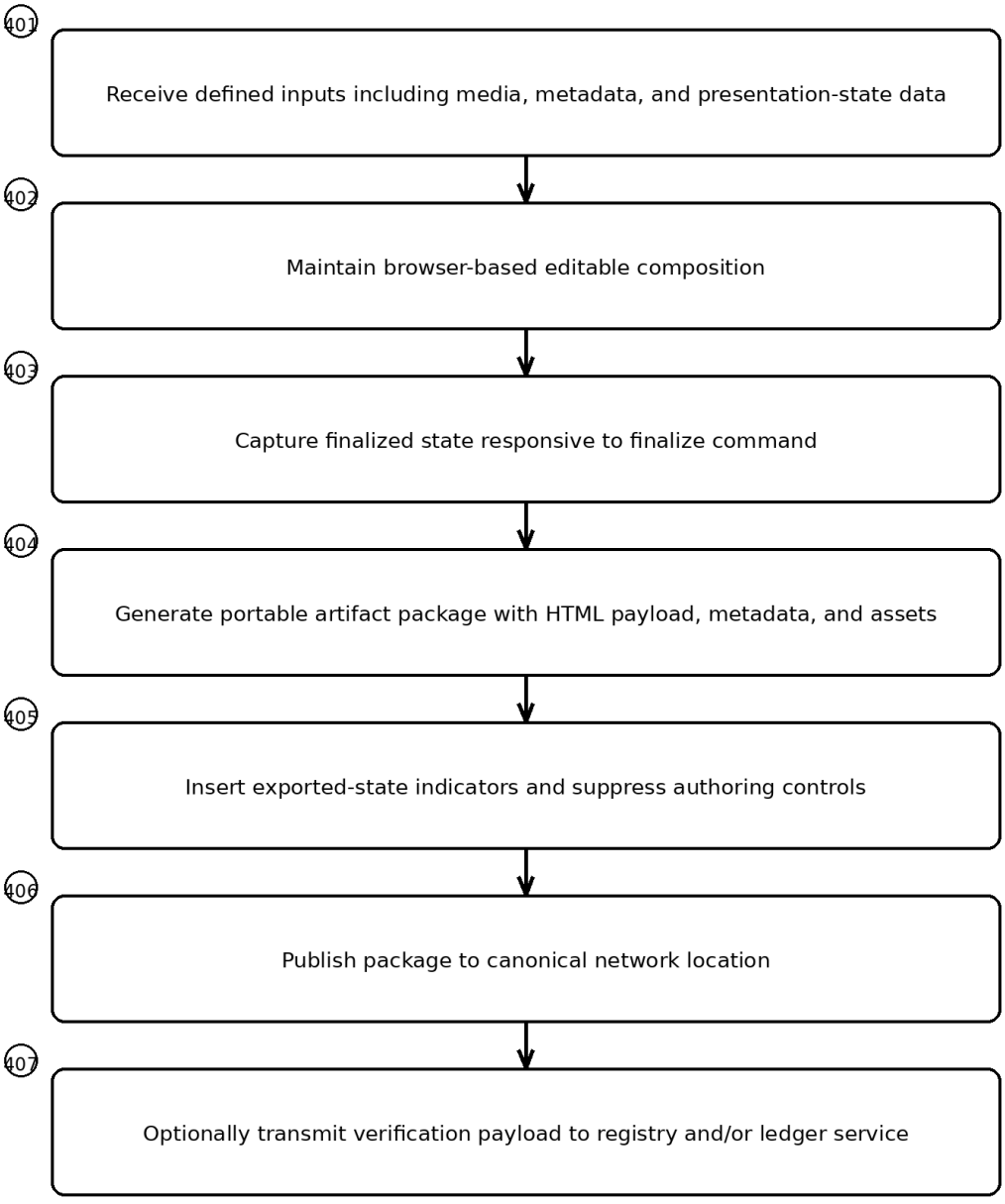

301
Artifact Container / Finals Path
302
index.html (HTML payload and primary browser viewer)
303
metadata.json
304
meta.json
305
assets/ role-mapped media, images, audio, video, scripts
306
export-state marker / lock-state indicator
307
canonical URL path + identity attributes

FIG. 4 — Example export method in which defined inputs are transformed from an editable browser composition into a portable self-rendering artifact package with optional verification transmission.

FIG. 4


401
Receive defined inputs including media, metadata, and presentation-state data
402
Maintain browser-based editable composition
403
Capture finalized state responsive to finalize command
404
Generate portable artifact package with HTML payload, metadata, and assets
405
Insert exported-state indicators and suppress authoring controls
406
Publish package to canonical network location
407
Optionally transmit verification payload to registry and/or ledger service

SpinStream ↔ OnlyUNO Mint Trace (Code-Level)

Summary
This note consolidates the inventor-supplied code trace for the mint pipeline. It supports the patent disclosure by showing a concrete freeze-at-mint pathway in which a browser-authored artifact is stabilized, exported, uploaded, and optionally associated with registry and ledger services.

Key supported points
- Finalization is triggered from the browser editor through a mint/finalize sequence.
- The finalized DOM state is exported into canonical HTML plus metadata and assets.
- The payload includes index.html, meta.json, metadata.json, and role-mapped assets.
- The artifact is marked exported/minted and authoring controls are disabled or rendered read-only.
- A final URL is returned and used for optional gallery redirect, optional registry ingest, and optional ledger recording.
- The visible code strongly supports freeze-at-mint packaging but does not fully prove end-to-end deterministic identity generation for every implementation path.

Patent significance
This trace strongly supports claims directed to browser-native artifact generation, finalization, packaging, published retrieval, exported-state markers, and optional registry/ledger association. It also supports careful claim drafting that treats deterministic identity as an embodiment rather than the only available implementation.