Forensic Reconstruction Platform

Subsystem Architecture Transition Record

From Three-File Static Architecture to Modular localStorage Pipeline

April 2026

1. Purpose

This document records the structural transition of the Forensic Reconstruction Platform from its original three-file static architecture to the current modular system. It is written as an engineering record, not a design proposal. The transition has already occurred. This paper documents what changed, why it changed, and how the current architecture addresses the limitations of the original.

2. Original Architecture

2.1 File Inventory

The original platform consisted of three HTML files and two shared utility scripts:

File	Function
`chronology.html`	Self-contained chronology viewer. All CSS, all JavaScript, all drawer panel markup, the jump menu builder, hash navigation, and the full entry DOM embedded inline.
`appendices.html`	Self-contained appendices viewer. All CSS inline, jump menu populated from a hardcoded JavaScript array of appendix IDs and titles. All appendix content baked directly into the HTML body.
`search.html`	Self-contained search interface. Carried a complete copy of the chronology DOM inside a hidden element. Search operated by querying this baked-in copy. All styles, all script, all panel markup inline.
`print.js`	Shared print utility. Referenced by chronology and appendices.
`printtab.js`	Shared print-tab utility. Referenced by appendices.

2.2 Data Flow

There was no data flow. Each file was a self-contained document. The chronology existed as literal HTML list items inside `chronology.html`. The same list items existed as a second copy inside `search.html` within a hidden `<ul id="chronology-shadow">` element. The appendices existed as literal HTML inside `appendices.html`. There was no shared data source, no localStorage pipeline, and no backup file.

2.3 Search Mechanism

Search operated directly against the baked-in DOM copy. The `chronology-shadow` element in `search.html` contained every chronology entry as inline markup. The search function iterated over `<li>` elements within this hidden list, testing text content against the user's query. Results linked to `chronology.html` with a fragment identifier derived from the matched entry's date.

There was no appendices search facility.

2.4 Jump Menu

In `chronology.html`, the jump menu was built at runtime by `buildInterleavedMenu()`, which walked the inline DOM, generated stable IDs via `ensureListItemId()`, and populated a `<select>` element.

In `appendices.html`, the jump menu was populated from a hardcoded JavaScript array. Each appendix ID and title was a literal string in the script block. Adding or removing an appendix required editing both the HTML body and the jump menu array.

2.5 Structural Limitations

1. **Dual maintenance of chronology content.** Every edit to the chronology required the same edit in two places: the entry in `chronology.html` and its copy in `search.html`. Drift between the two copies was inevitable and undetectable without manual comparison.
2. **No separation of data from presentation.** The chronology entries were the DOM. There was no distinction between the forensic record and the HTML that rendered it. An AI assistant editing the file could alter the record's structure without understanding that the HTML was evidence geometry, not presentation markup.
3. **No chain or chip infrastructure.** The original architecture predated the CHIP/ID grammar entirely. There were no `data-chains` attributes, no domain colour system, no chip rendering, no seed letter assignments, and no marker taxonomy. Cross-references existed as plain hyperlinks with no structural metadata.
4. **Hardcoded appendices index.** The appendices jump menu was a literal array. Every addition, removal, or title change required a code edit. The menu could not respond to the actual content of the page.
5. **Monolithic files.** All CSS, all JavaScript, and all panel markup lived inside each HTML file. Shared behaviour (drawer panel logic, colour variables, chip rendering) did not exist as shared code. Each file carried its own version of whatever it needed.
6. **No appendices search.** The original architecture provided no facility to search appendix content.

3. Current Architecture

3.1 Structural Principle

The current architecture separates data from presentation through a localStorage pipeline. The forensic record is stored as a JavaScript backup file (`chronology-backup.js` / `appendices-backup.js`) which writes a JSON structure into localStorage on page load. Viewer files read from localStorage and inject the entry HTML into the DOM at runtime. Search files read from the same localStorage keys. The backup file is the single source of truth. No file carries a baked-in copy of the record.

3.2 File Inventory

3.2.1 Viewer Files

File	Function
`chronology.html`	Chronology viewer shell. Loads backup, writes to localStorage, injects entries into the DOM. Hosts three drawer panels (marker taxonomy, CHIP/ID guide, target panel), tab bar, ribbon, and jump menu. Lives outside the shell iframe.
`appendices.html`	Appendices viewer. Force-refreshes localStorage from backup on every load (removeItem then setItem). Injects entries from storage. Dynamic jump menu built from the actual DOM after injection. Lives inside the shell iframe.

3.2.2 Search Files

File	Function
`search.html`	Chronology search. Reads chronology-state from localStorage, injects into a hidden shadow list, searches against that DOM. Hosts its own marker panel and CHIP/ID guide panel. Domain tile filter for chain-aware search. Chip rendering via search-legend.js. Lives inside the shell.
`append-search.html`	Appendices search. New facility with no predecessor in the original architecture. Reads appendices-state from localStorage, injects into a hidden shadow container, searches against card elements. Lives inside the shell.

3.2.3 Data Files

File	Function
`chronology-backup.js`	Single source of truth for all chronology entries. Declares a global variable containing a JSON structure with an entries array. Each entry has a dateKey and an html field. The html field is the opaque payload — complete markup transported without interpretation.

appendices-backup.js	Single source of truth for all appendix entries. Same structure: entries array, each with an html field.

3.2.4 JavaScript Modules

File	Function
chrono-core.js	Tab management, entry cloning, pane activation. The runtime engine for the chronology viewer's multi-pane display.
chrono-jump-builder.js	buildInterleavedMenu(), processListItem(), ensureListItemId(), runtime ID generation, chronoIndex/chronoDocs/chronoJumps collection, chrono-indexing-data localStorage write. The jump menu and indexing engine.
chrono-ancillary.js	exportDom(). Reads chronology-state from localStorage and produces a downloadable chronology-backup.js file. The round-trip export facility.
legend.js	Drawer panel open/close/toggle logic for all three panels (marker, guide, target). Loaded by chronology.html. Mutual exclusion: opening one panel closes the others.
search-legend.js	Marker taxonomy data (24 markers, 6 categories), chain domain data (6 domains, 14 numbered states), chip rendering (renderChips(), buildChip()), marker panel filters, and CHIP/ID guide panel content. Loaded by search.html. Includes its own panel open/close logic — deliberately separate from legend.js because search.html has no target panel and the two files never co-load.

3.2.5 CSS Files

File	Function
chrono-common.css	Shared CSS variables (:root), linked-evidence link styles, and evidence-type colour classes (custody, cad, report, append, office, judged, open, chrono, claim, evidence, doc_note, rui, mg4, oversight, mod, paper, route).
chrono-shell.css	Chronology viewer shell layout: tab bar, pane area, ribbon, flash-highlight animation, year-break styling, chronology-root counter, jump menu, scroll progress bar.
chrono-search.css	Domain tile filter bar, marker panel layout, guide panel layout, marker card expand/collapse, tag colour classes, guide domain palette.
pw-drawers.css	Consolidated drawer panel styles for all three panels (marker, guide, target). Sticky headers, tab positioning, card styles, filter

	bar. Used by chronology.html where all three drawers are present.

3.3 Data Flow

3.3.1 Chronology Pipeline

`chronology-backup.js` → `localStorage['chronology-state']` → DOM injection in `chronology.html`

`localStorage['chronology-state']` → DOM injection in `search.html` (hidden shadow list)

`chrono-jump-builder.js` → `localStorage['chrono-indexing-data']` → consumed by validator subsystem (pw-data-systems.html, pw-validator.html)

3.3.2 Appendices Pipeline

`appendices-backup.js` → `localStorage['appendices-state']` → DOM injection in `appendices.html`

`localStorage['appendices-state']` → DOM injection in `append-search.html` (hidden shadow container)

3.3.3 Key Principle

The HTML stored in each backup entry's `html` field is an opaque payload. It is transported from the backup file through localStorage to the DOM via `insertAdjacentHTML` without interpretation. The JavaScript acts as a Repeater, not a Translator. The payload arrives in the DOM exactly as it was authored. This is the core architectural invariant.

4. Transition Summary

Aspect	Original	Current
Data source	Inline HTML in each file	Backup .js files as single source of truth
Search data	Baked copy of chronology DOM in search.html	Hydrated from localStorage at runtime
Appendices search	Did not exist	append-search.html, reading from appendices-state
Appendices jump menu	Hardcoded JS array	Dynamic, built from DOM after injection
Chain/chip grammar	Did not exist	6 domains, 14 classes, 12 seeds, 24 markers, renderChips() in search-legend.js
CSS	Inline in each file, no sharing	Factored into 4 shared CSS files
JavaScript	Inline in each file	5 dedicated JS modules plus backup data files
Drawer panels	Did not exist	3 panels: marker taxonomy, CHIP/ID guide, target jump list
Domain tile filter	Did not exist	Chain-aware domain filtering in search
Shell architecture	All files independent	Chronology outside shell; search, appendices, append-search inside shell iframe
Export facility	None	chrono-ancillary.js exportDom() round-trips localStorage to downloadable .js
Validation subsystem	None	pw-data-systems.html / pw-validator.html consuming chrono-indexing-data

5. Known Architectural Risks and Mitigations

5.1 localStorage String Truncation

localStorage silently truncates payloads that exceed the browser's storage quota. This is a silent data loss path. The mitigation is byte-count matching: payload length measured at the source (backup file) must equal payload length measured after retrieval from localStorage. Any discrepancy indicates truncation.

5.2 dateKey Generation Drift

If the dateKey used by the backup file does not match the dateKey expected by the jump builder or search, entries become unreachable. The dateKey is a derived value. Its derivation logic must be identical everywhere it is computed.

5.3 Grammar Reversal by AI Assistants

The opaque payload principle requires that AI assistants treating the HTML as a transport format do not decompose, reinterpret, or restructure the markup. A documented instance occurred during the `buildInterleavedMenu` rebuild, where an AI assistant reversed the grammar from Transmission (opaque payload preserved as outerHTML) to Translation (decomposed into interpreted fields). The mitigation is the Repeater principle: JavaScript must act as a Repeater, not a Translator. The `insertAdjacentHTML` call is the final step. No interpretation precedes it.

5.4 Spurious ID Injection

Runtime IDs generated by `ensureListItemId()` during `buildInterleavedMenu()` execution must not be serialised back into the backup file. If a DOM round-trip captures these runtime IDs, the backup becomes contaminated with IDs that are implementation residue, not part of the authored record. The authoring tool (`chain-three.js`) emits zero IDs into the entry HTML payload. IDs are generated at runtime only and discarded when the page unloads.

6. Scope Boundary

This document covers the structural transition only. It does not cover the CHIP/ID revision schedule, the seed letter assignments, the marker taxonomy content, the forensic record itself, or the evidential chain analysis. Those are separate records maintained independently.

— End of Document —