Forensic Reconstruction Platform

Technical Assessment: Data Management, Validation, and Backup Editing Subsystems

April 2026

1. Scope

This assessment covers three subsystem files referenced in the FRP Architecture Transition Record as the validation and data management layer. The files assessed are:

- `pw-data-manager-v1_2_9.html` — 2,373 lines. Central data management console with seven tabbed viewports.
- `pw-validator-v1.html` — 909 lines. Standalone/embeddable validator for chronology and heads-of-claim link integrity.
- `backup-editor-v1.html` — 616 lines. Direct editor for chronology-backup.js and appendices-backup.js files.

The assessment covers architecture, data flow, functional capability, integration points, and identified risks. It does not cover the forensic record content or the CHIP/ID grammar.

2. PW Data Manager (v1.2.9)

2.1 Architecture

The data manager is a self-contained single-page application structured as a shell with seven tabbed viewports (vp-0 through vp-6). The shell provides a left navigation panel listing all known localStorage keys, and a viewport area that renders content specific to the selected key or tool. Viewports are created lazily via a `VIEWPORT_REGISTRY` object — each tab index maps to an init function that builds the viewport DOM on first activation.

2.2 Key Registry

The data manager maintains a `PW_KEY_REGISTRY` array (21 entries, next available ID: 22) that declares every localStorage key the platform is expected to use. Each registry entry records:

- **id** — unique integer identifier
- **key** — the literal localStorage key string
- **status** — live, vestigial, or unknown
- **writer** — the script that writes to this key
- **consumers** — array of files that read from this key
- **viewer** — which viewport template renders this key's data
- **notes** — open questions or migration notes

This registry is the platform's single authoritative declaration of its localStorage surface area. The left navigation panel is built from it. Keys with data show a green dot; keys without data show grey. Vestigial and unknown keys are labelled with status badges.

2.3 Tabbed Viewports

Tab	Label	Function
0	Guides	Domain tile grid linking to guide documents. Tile-click loads a guide into the viewport.
1	JSON Tools	Manifest manager: loads a local directory via file picker, builds a directory tree, renders file metadata.
2	Manifests	Manifest generator: produces text or JS manifest from a selected folder. Downloadable output.
3	PW Data	localStorage key inspector. Renders structured view, entry table, or raw JSON for the selected key.
4	Validator	Embeds pw-validator-v1.html in an iframe. Left-panel controls proxy commands to the validator via postMessage.
5	Notes	Notepad with character count and format toggle.
6	NULL 7	Reserved viewport. Empty init function.

2.4 Validator Integration

The validator runs inside an iframe (`validator-frame`). The data manager communicates with it via a `postMessage` protocol. On load, the validator sends a `pw-validator-ready` message to its parent. The data manager responds with `pw-validator-parent-present`, which causes the validator to hide its own dashboard (avoiding duplicate controls). Subsequent commands (load-folder, run-heads, run-chronology, set-archive, view-state, clear-state) are sent as `pw-validator-command` messages. The validator sends back `pw-validator-ui-state` messages containing its current summary HTML and status text, which the data manager renders in its own left panel.

This is a clean parent/child protocol. The validator operates identically in standalone mode (with its own dashboard visible) or embedded mode (dashboard hidden, commands received from parent). The mode is determined at runtime by whether `window.parent !== window`.

3. PW Validator (v1)

3.1 Purpose

The validator checks the integrity of document links and chronology jumps against a known file directory. It answers two questions: (1) does every file referenced in the chronology or heads-of-claim index actually exist in the archive? (2) does every chronology jump target an entry that exists in the chronology index?

3.2 Data Sources

The validator reads from three localStorage keys:

- `chrono-indexing-data` — the entry/document/jump index built by chrono-jump-builder.js on each chronology load
- `systemHeadsIndex` — the heads-of-claim index
- Archive directory indexes: `systemDirectoryIndex_assets`, `systemDirectoryIndex_briefs`, `systemDirectoryIndex_logs`, `systemDirectoryIndex_pdf`

It also supports a legacy fallback key (`systemDirectoryIndex`) for backwards compatibility with the pre-archive-split single index.

3.3 Directory Index Population

The directory index can be populated three ways:

- **Load Folder** — browser file picker with webkitdirectory attribute. The user selects the platform's root folder. The validator partitions files into the four archive buckets (assets, briefs, logs, pdf) and writes each to its own localStorage key.
- **Load Manifest** — reads a JSON or JS manifest file (produced by the data manager's manifest generator). Parses the contents array and partitions into archive buckets.
- **Load Default** — loads a JS manifest from a hardcoded path (`build-logs/manifests/validator-manifest.js`). Silent failure if unavailable.

3.4 Validation Logic

3.4.1 Heads Validation

Iterates the systemHeadsIndex. For each entry's documents array, extracts the file path (stripping any `src=` prefix), normalises it, and checks presence in the directory file set for the selected archive. Produces a report of present/missing document links.

3.4.2 Chronology Validation

Reads chrono-indexing-data. Validates two categories:

- **Documents** — each document's href is normalised, matched against the archive directory file set. Reports present or missing.
- **Jumps** — each jump's targetId is checked against the set of known entry IDs from chrono-indexing-data. Validates that every jump lands on an entry that exists in the chronology. The validator normalises ISO date-format IDs to handle the dateKey format.

3.5 Report Rendering

Validation results render in a four-panel layout: dates panel (left), documents panel (centre-left), jump detail panel (centre-right), and jumps panel (right). Dates with missing links show a red left border; dates with all links present show green. Clicking a date populates the document and jump panels for that entry. A summary row shows totals: dates OK/error, documents OK/error, jumps OK/error. The report is persisted to `systemValidationReport` in localStorage and rehydrated on reload.

3.6 Storage Listener

The validator listens for `storage` events on its source keys. If chrono-indexing-data, systemHeadsIndex, systemValidationReport, systemDirectoryIndex, or any archive index key changes, the validator resets the date panel and prompts for re-validation. This means a chronology reload in another tab (which rewrites chrono-indexing-data) is detected immediately.

4. Backup Editor (v1)

4.1 Purpose

The backup editor provides direct read/write access to the backup .js files (chronology-backup.js and appendices-backup.js). It is the only facility that allows entry-level inspection and modification of the opaque HTML payload outside of the authoring tool (chain-three.js).

4.2 Capabilities

- **Load** — accepts a .js backup file via file picker or drag-and-drop. Auto-detects whether the file declares `chronologyBackup` or `appendicesBackup`. Parses the JSON body.
- **Preview** — renders each entry's HTML payload in a preview pane, applying the platform's evidence-type colour classes (custody, evidence, chrono, oversight, etc.).
- **Source** — provides a textarea showing the raw HTML of each entry. Apply writes the textarea content back into the in-memory data structure and updates the preview. Revert restores from the in-memory copy.
- **dateKey / id editing** — each entry's key field (dateKey for chronology, id for appendices) is editable inline.
- **Dirty tracking** — modified entries show a red left border. The dirty set tracks which entries have been touched.
- **Save** — re-serialises the entire data structure as `var [varName] = [JSON];` and triggers a download. The saved file is a direct replacement for the original backup file.
- **Filter** — text filter searches across dateKey, id, and html fields to narrow the visible entry list.
- **Byte count** — shows character count for each entry's source textarea. This supports the byte-count matching mitigation for localStorage truncation.

4.3 Relationship to Platform Pipeline

The backup editor operates outside the localStorage pipeline. It reads and writes .js files directly, not localStorage. The edited backup file is then deployed by replacing the original backup file in the platform's data directory. On next page load, the viewer (chronology.html or appendices.html) picks up the new backup and writes it to localStorage. This makes the backup editor a cold-path tool — edits do not take effect until the backup file is redeployed and the viewer reloads.

5. Integration Map

The three subsystems form a layered support structure around the main platform pipeline:

Subsystem	Reads From	Writes To	Integration Mode
Data Manager	All localStorage keys (via registry)	None (read-only inspector)	Standalone HTML + iframe for validator
Validator	`chrono-indexing-data`, `systemHeadsIndex`, archive directory	`systemValidationReport`, archive directory indexes	Standalone or embedded via postMessage

	indexes		
Backup Editor	Backup .js files (file system)	Backup .js files (download)	Fully offline, no localStorage interaction

6. Identified Risks and Observations

6.1 Key Registry Gaps

Three keys in PW_KEY_REGISTRY have status "unknown" with empty writer fields: `matrix-state` (id 13), `appendices-state` (id 14), `claim-state` (id 15). The appendices-state key is actively written by appendices.html on every load (force-refresh pattern) and consumed by append-search.html. The registry entry does not reflect this. The registry should be updated to show the correct writer (appendices.html via appendicesBackup) and consumers (appendices.html, append-search.html).

6.2 Validator Date Normalisation

The chronology validation function normalises ISO-format targetIds (`date-YYYY-MM-DD`) to `date-DD-MM-YYYY` before checking against the known ID set. This assumes the chrono-indexing-data entries use DD-MM-YYYY format IDs. If the dateKey or ensureListItemId logic changes upstream, this normalisation would silently pass invalid jumps or fail valid ones. The normalisation direction should be documented and pinned to the ID format contract.

6.3 Backup Editor Has No Validation Gate

The backup editor applies source changes directly to the in-memory data structure with no structural validation. A malformed HTML payload, a mismatched dateKey, or an accidental truncation will be saved without warning. The byte count display mitigates truncation detection but does not prevent it. The editor trusts the operator. This is appropriate for the intended user, but should be noted as a design decision rather than an omission.

6.4 Factory System Index Writers Unconfirmed

Three keys in the registry (`systemChronologyIndex`, `systemMatrixIndex`, `systemHeadsIndex`) have empty or uncertain writer fields with notes reading "Confirm writer." These are consumed by the validator (chain-eight.js). The writer provenance should be confirmed and recorded.

6.5 Legacy Key Retention

The legacy `systemDirectoryIndex` key (id 11, status vestigial) is retained for backwards compatibility. The validator falls back to it when the per-archive keys are absent. This is defensible but creates a latent path where stale legacy data could be consumed if the per-archive keys are cleared without clearing the legacy key. The clear-state function correctly removes both.

7. Scope Boundary

This assessment covers the three subsystem files as delivered. It does not cover the authoring tool (chain-three.js), the temporal graph scripts (chrono-core.js, chrono-jump-builder.js), the shell infrastructure, or the forensic record content. Those are documented separately.

— End of Document —