

Pre-Note Delta: evidence exhibits

Pre-Note Delta answered a specific preparation question for clinicians: what changed with this patient since I last saw them? The work combined an evidence-linked clinical application, tools that made authenticated vendor information available to coding agents, AI-operated testing, evaluation of generated prose, and a demonstrated cloud environment.

Justin Sublette owned the product direction, architecture, integration and evaluation decisions, working extensively with AI coding agents on implementation, investigation and tests. The exhibits make those decisions inspectable; they do not attribute every source line to unaided human authorship.

This is a historical source selection prepared September 16, 2026. Implementation and retained receipts are frozen at repository revision ef8546aaa7a79c006f75a68da027389912b1f2d1, except the explicitly identified pre-repair comparison. The system was retired on July 3, 2026. No application, model or cloud run was performed for this curation. Source excerpts are verbatim selected ranges; result tables are labeled editorial transcriptions. Patient records, vendor documentation contents and environment identifiers are excluded.

1. A precise product question expressed in code

The March 5 design decision separated two scopes: clinician identity selects the prior encounter, while changes after that time are collected across the patient's record. Restricting those changes to the same clinician would miss the very events the clinician needed to discover. The application separated a React web interface, an API service and a shared TypeScript core for identity, baseline calculation, summaries and evidence normalization.

In `computeBaseline.ts`, candidates are sorted newest first. The clinician-selection branch skips the current encounter and matches encounter participants against resolved clinician identities:

```
// Clinician-last selection.
if (hasClinicianContextUsable) {
  for (const c of candidates) {
    if (skipRef && c.encounterRef === skipRef) continue
    scannedForClinician += 1

    // Intersection between encounter participants and clinicianRefs.
    for (const pref of c.participantRefs) {
      if (!clinicianRefs.has(pref)) continue
    }
  }
}
```

The surrounding implementation returns the matching encounter and records why fallback was needed when none qualifies. The fixed twenty-section summary does not flatten uncertainty into prose: items carry evidence references, and each section can carry its own errors. The relevant declarations in `schema.ts` are:

```
export interface SummaryItem {
  text: string
  severity: SummarySeverity
  evidence: EvidenceRef[]
}
```

```
export interface SummarySection {
  key: SummarySectionKey
  title: string
  items: SummaryItem[]
  errors: SummaryError[]
}
```

An April 21 retained sandbox receipt connects this design to observed evidence navigation. **Editorial transcription of its aggregates:**

Observed boundary	Recorded result
Selected sandbox patient runs	16/16 passed
Expected evidence references loaded	3,599/3,599
Section instances containing evidence	203
Section instances with no matching rows/evidence	117
Failed section instances in the final run ledger	0

The completed ledger included operator recovery after intermediate desktop-state failures. Eighteen of twenty sections met a five-patient positive-example target; care-team and consent/code-status evidence remained thinner. This establishes a substantial working sandbox application and evidence interface, not clinical benefit or universal data coverage.

Sources: March 5 baseline-scope ADR; computeBaseline.ts; schema.ts; *Epic Golden Summary Section Evidence Proof Receipt*, April 21.

2. Giving coding agents access to authenticated specifications

The integration depended on material available inside an authenticated Epic developer session. Generic FHIR knowledge did not replace that vendor-specific information. Justin's development workflow therefore acquired browser capture and local documentation tooling, progressing from DOM extraction to authenticated capture and a structured reference library.

The retained exporter executes inside the signed-in browser. Its bounded worker pool retrieves selected API payloads and records individual failures. The request helper in export_epic_official_docs_from_browser.js shows the authentication mechanism:

```
async function fetchJson(relativePath) {
  const response = await fetch(`${basePath}${relativePath}`, {
    credentials: 'include',
    headers: {
      Accept: 'application/json, text/plain, */*',
    },
  });
  if (!response.ok) {
    throw new Error(`HTTP ${response.status} for ${relativePath}`);
  }
  return await response.json();
}
```

One important correction appears in the April 14 capture receipt: the live catalog's app-selected flags were unreliable. The collector used a frozen selected-name list to resolve live API identities instead. The recorded collection obtained **191 of 192 selected API documents, six rendered guidance pages, and 59 of 60 broader reference targets**. Failures remained visible rather than becoming an apparent complete corpus.

This is evidence of engineering the information-access environment around a coding agent. The exhibit supplies the capture mechanism and aggregate outcome; it does not redistribute the authenticated documentation or claim access to every Epic document. Justin's account establishes that suitable native browser access was unavailable in his workflow at the time, not an industry-wide first.

Sources: browser exporter, lines 15-26 and 31-69; *Epic Official Docs Library Receipt*, April 14, particularly the selected-name correction and result counts.

3. Testing an AI operator separately from the product

The desktop harness used a model to inspect screenshots and structured state, then choose among allowed UI actions. The controller bounded steps, validated actions, retained decisions and fell back when confidence was insufficient. This model operated the test environment; a separate model path generated optional product prose.

The low-confidence branch in `desktopAi.ts` records the disposition rather than treating any model response as progress:

```
if (decision.confidence < options.ai.minConfidence) {
  const failure = {
    status: 'fallback' as const,
    reason: `Desktop AI confidence ${decision.confidence.toFixed(2)} below minimum
${options.ai.minConfidence.toFixed(2)}`,
    usedEscalation,
    fallbackTarget: resolveFallbackTarget(options.ai, options.approvalEnabled, 'automatic'),
    stepCount: step,
  }
  options.bundle.writeJsonArtifact('ai/pilot-summary.json', failure)
  options.bundle.addCheck('desktop_ai_pilot_completed', false, failure.reason)
  return failure
}
```

Its retained unit test injects a mock decision with confidence 0.3 against a 0.8 threshold. The exact assertions are:

```
assert.equal(result.status, 'fallback')
assert.equal(result.fallbackTarget, 'deterministic')
```

Those assertions test controller behavior under a supplied model response. They do not measure a live model's desktop competence. The pilot also allowed a model-issued success action, so its own completion flag was insufficient evidence of application correctness. A surrounding browser harness independently observed real page requests, summary streams and AI requests. Its observer window included background generation that could begin before the AI tab opened, preventing a late observer from misclassifying successful preload as a missing request.

Sources: `desktopAi.ts`; `hyperspace_desktop_ai.test.ts`; browser observers and selection-window handling in `launchpad/runner.ts`. These are retained implementation and test definitions, not newly executed results.

4. A successful request was not enough: finding and repairing source-use failure

The March 9 quality audit covered 96 captured cases across scenario profiles, six baseline choices and two drafting modes. It classified **53 as passing, 42 as input-limited, and one as a major rewrite defect**. The defective stream completed but contained corrupted output. Sparse input and bad generation were deliberately kept separate.

A twelve-case investigation then asked whether richer source notes actually influenced the final prose. The initial digest-and-merge arrangement could receive source text yet underuse its digest or fail on large inputs. Commit 12fdfa78403df0b3a5fbd97960c76825ec3e848d, March 9, replaced that path with bounded direct source packets. The following table transcribes the retained before/after aggregate fields; the March 10 record reconfirmed the repaired shape using the browser-faithful harness.

Recorded measure	Before repair	Repaired path
Complete captures	10	12
Partial captures	2	0
Confirmed source-enriched results	0	4
Effectively summary-only results	8	6
Insufficient-source results	4	2

Before-repair source: Epic_LaunchPad_AI_Source_Proof_Matrix.md at 581f223f42ce1217f0c35bb49e2ddc11a204528a, the correction's parent revision. After-repair source: the retained March 10 version at the frozen snapshot.

This evidence connects a diagnosis to an architectural change and an observed result. It also preserves the remaining limits: direct access to sources did not make every case richer. These were internal assessments; the surviving records do not identify who or what assigned each quality verdict. They do not establish clinician adjudication. Later behavior checks did not rescore the full 96-case quality matrix.

5. Commercial deployment engineering reached an operating demonstration

The implementation included dedicated customer-environment infrastructure, private application/data services, controlled public ingress, migrations and audit retention. A retained April 25 receipt records an actual AWS demonstration deployment owned by Vector Mosaic, rather than only a topology proposal.

Editorial transcription, with deployment identities removed:

Receipt observation	Recorded outcome
Web service desired / running / pending tasks	2 / 2 / 0
Separate API service desired / running / pending tasks	2 / 2 / 0
Public health response	HTTP 200
Public load balancer	WAF associated; deletion protection enabled
Known synthetic audit event	Delivered into encrypted S3 archive
Archive retention for that event	Six-year Governance Object Lock

The synthetic event was read back with encryption and retention metadata. This supplies a concrete operational result for the deployment work without exposing cloud account, network or object identifiers. It proves the recorded demonstration boundary; it does not establish a hospital rollout, institutional approval, commercial adoption or present-day service health.

Source: *Pre-Note Delta Demo Prod WAF and API Boundary Receipt*, April 25, service table, live audit-delivery section and result summary. Together with the preceding exhibits, it shows work extending from a precise product idea through agent tooling, application integration, test-quality improvement and operating infrastructure.