

TECHNICAL WHITEPAPER

The ShowPapers Protocol

From Scattered Documents
to Usable, Portable Records

Original documents. Structured context.
Explicit provenance. Reviewable exchange.

Nirbhay Pherwani · Ameya Kulkarni

hello@showpapers.app

<https://protocol.showpapers.app>

Contents

1	Introduction	2
2	Requirements and Design Choices	2
2.1	What ZIP Provides, and What the Protocol Adds	2
2.2	Exchange as a Selected Snapshot	3
3	Collection Representation	3
3.1	The Collection Format	3
3.2	Records and Identity	4
4	Provenance and Review	4
5	Exchange with AI Agents	5
5.1	Collections, Companions and Proposed Changes	5
5.2	Review and Scope	6
6	Security and Trust Boundaries	6
6.1	Strict Reading and Resource Limits	6
6.2	Protected Envelope	7
7	Implementation Scope	8
8	Conformance and Application Support	8
9	Conclusion	8

Executive summary

Personal paperwork is useful when a person can find the original, understand its details, and connect it to what they need to do. Sharing photos and PDFs alone often leaves behind the organization, Notes and review decisions built around them. The ShowPapers Protocol keeps that context alongside original documents in one portable collection. Specification 1.0.0 defines how original documents, structured details, Notes and review decisions travel together—and how people, apps and AI agents exchange that work without losing its context. ZIP carries the files; the protocol defines their records, relationships, provenance and proposed changes. This report explains the collection contract, review and import rules, agent exchange, validation and optional encryption.

1 Introduction

A person preparing for a trip may have an identity document in Photos, a confirmation in email, and a plan in Notes. Finding those files is one task. Knowing which details have been checked, connecting the right papers to the plan, and keeping that work available when the papers are shared is another. A folder of attachments does not define those relationships.

ShowPapers brings scattered paperwork into usable personal records: originals people can find, details they can inspect, and Notes and organization they can build around them. The protocol addresses the portability of that work. It represents a selected collection with the context needed to interpret its records, so an exchange need not start again from unexplained attachments.

AI adds a second reason to preserve context. An inferred date needs its source and evidence to travel with it. An agent's proposed change enters the library only through an authorized import or selected change. The protocol defines records, provenance and a reviewable exchange; the specification and published test files define the rules implementations must follow [1].

2 Requirements and Design Choices

The design has three requirements. Originals must remain available beside derived information. Record identity, relationships and the person's review state must survive a conforming exchange. Receiving software must validate the collection as a whole and keep external-agent proposals distinguishable from the person's existing work.

2.1 What ZIP Provides, and What the Protocol Adds

ZIP supplies a container for named files [2]. It can carry PDFs, JSON or Notes, but it does not define which file represents a document, how a detail refers to its original, or whether a value is an unreviewed suggestion. A custom ZIP can encode this information; independent readers still need an agreed contract to interpret it.

ShowPapers supplies that contract. Stable IDs bind originals to records and readings. Defined schemas describe details, Notes, folders and relationships. Provenance and review state record where values came from and how they have been treated. A changes document describes proposed edits to one exact collection. Validation checks declared payloads, references, digests and resource limits before an import can proceed.

The protocol gives the contents a shared meaning. ZIP and JSON keep the collection inspectable with common tools [3]. A conforming reader checks both the file structures and the relationships among records, so the same collection can be interpreted consistently across implementations.

2.2 Exchange as a Selected Snapshot

A collection is a selected snapshot, not a live shared database or full-vault backup. It can be copied through ordinary file-sharing mechanisms; later edits do not synchronize to recipients. A separate encrypted envelope can protect a selected copy. This lets the person exchange useful context without requiring a common account, server or model.

3 Collection Representation

3.1 The Collection Format

A readable collection contains exactly one root `manifest.json` and the payloads it declares. Explicit directory entries and undeclared files are refused. Payload paths are derived from canonical record IDs. Readers inspect those paths within the archive; they do not extract them to a filesystem.

An original is the PDF or image. A reading stores text recognized from it and where that text appears. A paper record holds details such as dates or document numbers, together with their origin and review status.

The collection format combines original PDFs or images with optional paper records, readings, details and review state, Notes, people, folder rules, reminders, pins, relationships and agent provenance. These sections describe the records and the work built around them; they are not replacements for the originals. The following sections define their roles.

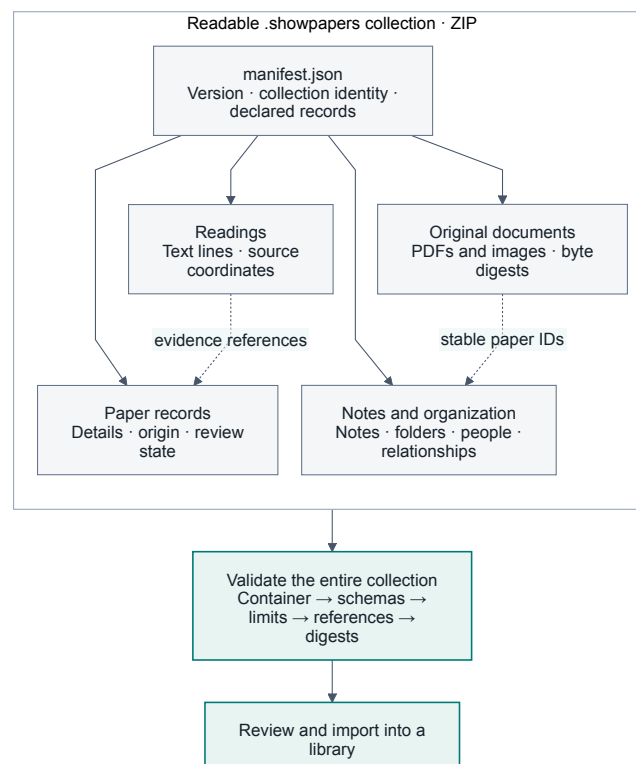


Figure 1: Logical contents of a ShowPapers collection. The manifest indexes declared payloads and also holds organization metadata. Paper records and readings are optional; original bytes are retained.

3.2 Records and Identity

A paper is associated with one original file. Its optional paper record and reading use the original's ID. Titles describe the record to a person; IDs support references among records. A paper's identity on import depends on both its ID and the SHA-256 of its stored original. An incoming paper with the same ID but different bytes is a conflict and can be offered as a separate copy with a new ID.

Not every collection contains every feature. Beyond the required manifest members and originals list, sections can be omitted. Omission of a reading or paper record does not discard the original. This allows a small collection containing just originals while retaining a defined path to richer records. JSON Schema describes the allowed fields and values [4]. Readers also check that records refer to existing items and that the archive follows the collection rules.

4 Provenance and Review

Provenance means the recorded origin of a detail. The format distinguishes three detail origins: `reading`, `person` and `agent`. A reading detail references evidence in recognized lines. A person-origin detail is a custom text value typed by the person, without the `reading/agent` review object. An agent detail identifies an entry in the manifest's agent list, can include optional confidence and evidence, and carries review state. Confidence records the agent's estimate. Acceptance can follow a person's review or a disclosed application policy; neither proves the value correct.

For reading and agent details, review state includes a confirmation flag and an optional corrected value. The effective value is the correction when one exists, otherwise the recorded value. Removed reading suggestions and rejected agent suggestions can be retained so that re-importing the same proposal does not silently restore them. This representation separates origin, recorded value and acceptance state rather than rewriting all three into a single final string. The confirmation flag does not distinguish manual review from automatic acceptance.

The import rules preserve the person's existing work by default. An agent suggestion does not automatically replace a confirmed library value. Pending agent details do not drive reminders, folder rules, end dates or case history until confirmed. The importing person can choose between an existing value and an incoming value when the two conflict.

Preserving incoming review state is an explicit trust decision. The import contract includes a `Keep review marks` option, enabled by default for files whose generator names `ShowPapers` and disabled for other writers. The generator name is a writer-supplied label, so the person needs to trust the file's source when keeping its review marks. With this option off, reading details arrive unconfirmed and relationship decisions arrive as suggestions. `ShowPapers` separately accepts agent details and an unset form type when declared confidence is strictly above 50% and the person adds the collection. Confidence at or below 0.5, or missing confidence, stays pending. Saved corrections and selected types remain protected. Accepted details keep their agent identity and evidence; a person can inspect and correct them later. Other protocol consumers can choose manual review or their own disclosed policy.

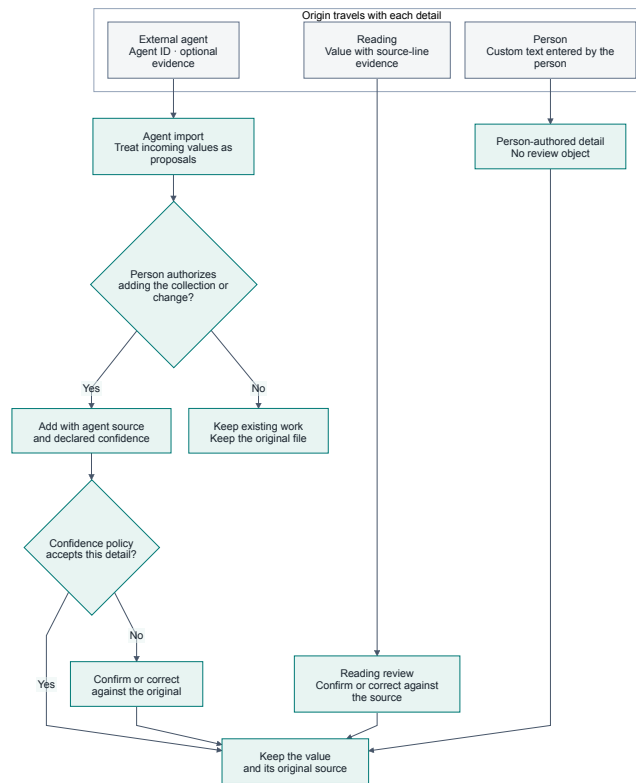


Figure 2: Details record whether they came from document reading, a person or an external agent. Reading and agent details also carry review state; corrections preserve the recorded source value.

5 Exchange with AI Agents

5.1 Collections, Companions and Proposed Changes

An agent capable of reading files and running code can inspect a collection and return a complete new archive. External-agent details must record their origin and identify the agent. A returned collection is validated before an application offers its contents for import or review. The receiving agent needs archive-reading support to inspect the file and code execution to build a new collection. The agent guide provides the required schemas, commands and validation steps.

For agents that can produce text but cannot build an archive, a changes document describes 1–500 proposed operations against one exact base. It binds to the base collection ID, revision and archive SHA-256. That binding detects a mismatched base before changes are applied. The person authorizes the proposed edits separately during review. Supported operations include proposals affecting details, notes, folders, people and relationships, and removals for review. A changes document cannot add original bytes: new originals require a complete collection.

A readable JSON companion describes a collection for a system unable to inspect ZIP files. It carries structured details and reading text; original document bytes remain in the collection. Visual evidence is checked against those originals. Because a companion contains personal document context, it needs the same care when sharing. Its schema and the changes schema define the two JSON exchange contracts.

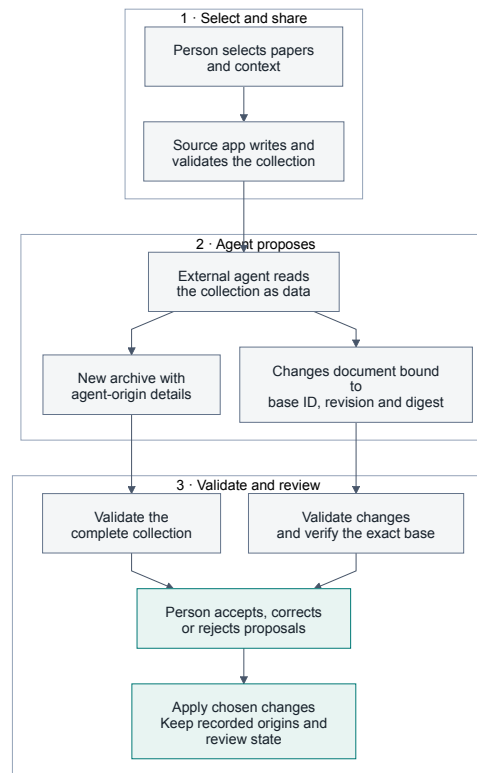


Figure 3: The agent round trip. A validated collection or changes document carries proposed work back to the person, who reviews it before accepted changes enter the library.

5.2 Review and Scope

The reference tool can preview proposed changes and construct a validated next-revision archive. The app-side contract separately requires a review screen, selection of accepted changes and applying the selection as one complete operation: an interrupted apply leaves the library unchanged. After the person applies selected operations, agent details follow the receiving app's acceptance policy; confidence at or below 0.5, or missing confidence remains pending for review. Constructing a valid output file and accepting its changes into a library are separate steps.

Text inside originals, notes or companions is untrusted data. It does not grant permission, widen file access or change an agent's task. A malicious document can contain instructions; provenance fields and archive validation cannot prevent an AI model from following them. Implementations must enforce scope in their host and treat document text as evidence rather than authority. Sending readable papers to a cloud AI service gives that service access to their contents under its own privacy and retention practices. For sensitive papers, a person can instead use a trusted local AI setup configured to keep document content on the device, including any connected tools.

6 Security and Trust Boundaries

6.1 Strict Reading and Resource Limits

The container admits stored or deflated ZIP entries and refuses ZIP64, ZIP-layer encryption, extra fields, symlinks and unsafe paths. JSON must be valid UTF-8 without a byte-order mark or duplicate keys, and objects reject fields the format does not define. Collection JSON has a maximum nesting depth of 32. Readers enforce declared sizes, CRCs, SHA-256 digests, media signatures, expansion bounds and

cross-record references before returning an inventory.

A readable archive is bounded at 103 MiB. It is refused as a whole if truncated, corrupt, oversized or partly unsupported. Rejecting partial input avoids silently importing a plausible subset while omitting a record that changes its interpretation. These checks bound parsing and resource use. Applications also need maintained document parsers and appropriate isolation when rendering the admitted PDFs and images.

SHA-256 checks establish agreement between payload bytes and the manifest. They detect corruption relative to those declarations, but do not authenticate the writer: an attacker who can rewrite the archive can also recompute its digests. A file can pass every format check while containing false information or a forged document. Agent names and review marks likewise provide labels, not signatures.

6.2 Protected Envelope

The protected envelope wraps exact readable archive bytes with Tink Streaming AEAD [5]. The contract selects AES-128-GCM-HKDF with 1 MiB ciphertext segments, a 16-byte key, HKDF-SHA-256 and a 16-byte derived key. The envelope uses Tink's streaming construction with GCM and HKDF [6, 7], an independently generated key, and authenticated segments.

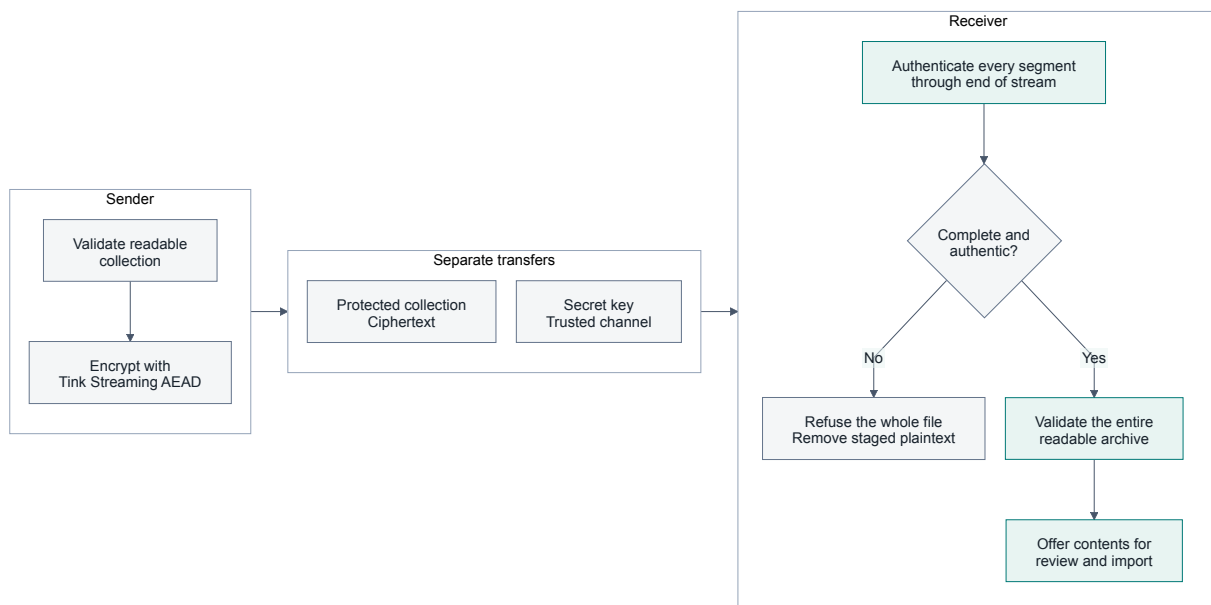


Figure 4: The protected envelope. The key is kept apart from the ciphertext. Readers authenticate the complete stream and validate the inner archive before exposing an inventory.

The encrypted file's exact format prefix is associated data. The key file carries canonical Tink key material and is capped at 4096 bytes; the encrypted collection is capped at 104 MiB. Decryption must reach authenticated end-of-stream and validate the complete inner archive before exposing any inventory or allowing import. Wrong keys, altered segments, truncation and trailing bytes fail closed.

Protection is only useful while the key remains secret. Anyone given both the file and its key can decrypt it, and encryption cannot control a recipient after decryption. The format provides no server recovery mechanism for a lost key. This envelope is distinct from each app's native encrypted storage and full-vault recovery design. Application support for protected collections must be checked separately from file conformance.

7 Implementation Scope

The protocol standardizes the records produced by document reading and the rules for exchanging them. Applications choose their OCR engines, on-device models, extraction pipelines and native storage. Reading-origin records can carry output from these different implementations through the same collection contract.

A collection is a portable copy of selected records. Physical originals remain necessary where required, and later edits stay with the copy in which they were made. Account services, live synchronization and access to a recipient's copy belong to the applications and services used for sharing.

Interoperability depends on the capabilities needed for each exchange: collection reading and writing, protected-envelope support, or JSON companions and proposed changes. The conformance tools let implementers check these capabilities against the same defined records and refusal rules.

8 Conformance and Application Support

Specification 1.0.0 defines the collection contract through schemas and normative rules. Readers validate the complete collection, including archive entries, payloads and cross-record references. The supported exchanges depend on each implementation's capabilities.

The standard-library Python tool creates, validates, inspects and edits readable collections, previews and applies changes, and writes companions. Separate tooling implements protected envelopes. The browser validator checks files locally without uploading them; it recognizes protected envelopes, which require a supporting application for decryption.

Implementations verify conformance with the JSON Schemas, semantic checks, valid and invalid corpus, and differential comparisons against the reference implementation. These checks cover record shapes, ZIP constraints, ordering, cross-record references and refusal behavior. Reproducible fixtures make disagreements visible and give implementers a practical basis for verifying interoperability.

9 Conclusion

The ShowPapers Protocol makes the work around personal documents portable. Originals travel with their structured details, linked Notes, organization and review decisions, so sharing a collection preserves more than a folder of attachments. People, apps and AI agents can interpret that context through one defined contract. Agent-proposed details retain their origin and review state, and the import rules protect the person's existing work.

Specification 1.0.0 brings these pieces together: stable record identity, explicit provenance, reviewable changes, complete validation and an optional protected envelope. The schemas, examples and reference tools give implementers a concrete path to creating and exchanging collections. A person can carry their papers and the context built around them into the next app, conversation or task.

References

- [1] N. Pherwani and A. Kulkarni. *ShowPapers Protocol Specification*, version 1.0.0, 2026. <https://protocol.showpapers.app/spec>.
- [2] PKWARE. *APPNOTE: .ZIP File Format Specification*. <https://pkware.cachefly.net/webdocs/casestudies/APPNOTE.TXT>.
- [3] T. Bray, ed. *The JavaScript Object Notation (JSON) Data Interchange Format*. RFC 8259, December 2017. <https://www.rfc-editor.org/rfc/rfc8259>.
- [4] A. Wright, H. Andrews, B. Hutton and G. Dennis. *JSON Schema: A Media Type for Describing JSON Documents*. Draft 2020-12, June 2022. <https://json-schema.org/draft/2020-12/json-schema-core>.
- [5] Google. *Streaming Authenticated Encryption with Associated Data (Streaming AEAD)*. Tink documentation. <https://developers.google.com/tink/streaming-aead>.
- [6] M. Dworkin. *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*. NIST SP 800-38D, November 2007. <https://doi.org/10.6028/NIST.SP.800-38D>.
- [7] H. Krawczyk and P. Eronen. *HMAC-based Extract-and-Expand Key Derivation Function (HKDF)*. RFC 5869, May 2010. <https://www.rfc-editor.org/rfc/rfc5869>.