Independent Submission
Internet-Draft
Intended status: Informational
Expires: 4 December 2026

W. Han
Individual

2 June 2026

AI Manifest: Site-Published Friction-Recovery Descriptors for AI Agents
draft-han-ai-manifest-02

Abstract

This document specifies the AI Manifest, a JSON-based format with which a website operator publishes an AI Friction-Recovery Manifest (AFRM): a declarative, advisory description of the known user-interface (UI) traps on a site, the framework hints that contextualize them, and the interaction shortcuts that bypass them. An autonomous AI agent that uses browser-automation tools discovers the manifest out-of-band and consults it as reference data when it encounters friction, instead of re-inferring site-specific behavior from the full Document Object Model (DOM) by trial and error. The manifest is advisory data published by the origin; it is not part of, and does not modify, any browser-automation tool schema.

The specification defines five interoperable discovery methods, the AFRM three-part schema (frameworkHints, knownTraps, and shortcuts), and an OPTIONAL SHA-256 canonical-hash verification procedure via a trust registry, together with security mitigations against prompt-injection attacks.

Empirical evaluation with a contemporary LLM browser agent on a proprietary-knowledge enterprise task -- an SAP S/4HANA-style table-maintenance screen whose valid values depend on a company-code registration scope that is not derivable from public or training knowledge -- shows that a site-published manifest, consumed directly by the agent with no external runtime, reduces interaction actions by approximately 54% and input tokens by approximately 45% relative to the no-manifest baseline, with both conditions completing the task. The manifest’s value concentrates where the agent cannot succeed from training knowledge alone: site-specific dependencies, data, and hard mechanical blocks.

This -02 revision supersedes the deterministic-runtime framing of draft-han-ai-manifest-01. The earlier revision reported a single-trial preliminary measurement and concluded that an embedded manifest is of negative value unless paired with a deterministic execution runtime (an "SDK"). Subsequent multi-condition measurement with a real LLM agent shows that a manifest is consumed and is useful on its own; an out-of-band discovery path (the Method A well-known URI, or

the user-curated Method E) avoids the prompt-injection-defense friction that the earlier embedded-only trial encountered. A deterministic runtime remains OPTIONAL -- useful for latency- or service-level-sensitive deployments -- but is no longer presented as REQUIRED.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Large Language Model (LLM)-based AI agents increasingly interact with web services via browser-automation protocols such as the Model Context Protocol (MCP), Playwright, Puppeteer, and Selenium WebDriver. Current agents typically parse entire DOM trees or screenshots on every page to infer UI structure, producing three well-known problems:

1. Substantial token consumption due to repeated analysis of large DOMs on every session.

2. High failure rates on complex multi-step transactional UIs such as enterprise resource planning (ERP) systems, academic manuscript submission portals, and government e-services.

3. Absence of a standardized mechanism for a website operator to declare an AI-agent-friendly ("AI-Ready") operational surface.

Related prior work includes robots.txt, llms.txt, agents.txt, and ai-plugin.json. These address crawling permissions, LLM-friendly documentation, agent capability declarations, and API-level integration respectively. None provides step-by-step UI workflow instructions for multi-page transactional flows.

AI Manifest fills this gap by specifying a JSON format that describes a site's known UI traps, framework hints, and interaction shortcuts. An AI agent discovers and, optionally, verifies the manifest, then

consults it as advisory reference data when it encounters friction, instead of rediscovering the same site-specific behavior by repeated DOM-based inference.

2. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

AI Manifest (AI Friction-Recovery Manifest, AFRM) A structured data object, expressed in JSON, that a website operator serves at a well-known URI or otherwise publishes (see Section 3.1), containing at minimum a version, a publisher, and a knownTraps array; each trap entry carries a category, a CSS selector, and an escapeAction. The two terms are used interchangeably in this document.

AI Agent A software process, typically driven by an LLM, that accesses a web page via a browser-automation tool and executes actions on behalf of a human user.

Central Trust Registry A network service that stores SHA-256 hash values of manifests pre-registered by publishers, and responds to real-time trust lookups from AI agents with a status of white-listed, black-listed, or unknown.

Canonical Form The representation of an AI Manifest obtained by lexicographically sorting all JSON object keys at every nesting level and serializing the result using the JSON encoding defined in [RFC8259] with UTF-8.

3. Protocol Overview

3.1. Embedding Methods

A website MAY provide an AI Manifest via one or more of the following methods:

3.1.1. Method A: Well-Known URI

The server SHOULD make the manifest retrievable at the following well-known URI [RFC8615]:

/.well-known/ai-manifest.json

In addition, the HTML document SHOULD declare the entry point via an HTML meta element:

```
<meta name="ai-manifest"
      content="/.well-known/ai-manifest.json">
```

3.1.2. Method B: Hidden DOM Element

The server MAY embed the manifest into the HTML response as a hidden element with the display:none style and aria-hidden="true" attribute:

```
<div id="ai-manifest" style="display:none" aria-hidden="true"
     data-manifest='{"version":"1.0", ... }'></div>
```

3.1.3. Method C: HTTP Response Header

The server MAY declare the manifest location and hash in a response header:

```
X-AI-Manifest: url=/.well-known/ai-manifest.json;
               hash=sha256:<hex>
```

Method C is RECOMMENDED in conjunction with Method A, so that an AI agent can discover the manifest URL and validate the hash in a single request-response round trip before fetching the body.

3.1.4. Method D: HTML Link Element

The server MAY declare the manifest via an HTML link element in the document head:

```
<link rel="ai-manifest" href="/.well-known/ai-manifest.json">
```

Method D allows AI agents that parse the DOM but do not inspect HTTP headers to discover the manifest location. The rel value ai-manifest is registered in Section 5.4.

3.1.5. Method E: AI-Side Curation

A user MAY place a manifest at a vendor-conventional path in their own file system, for example ~/.claude/afrm/{domain}.json or ~/.codex/afrm/{domain}.json. The AI runtime loads the file when it visits a matching domain. Because the manifest originates from the user's authored context, it enters the agent's trust context without the prompt-injection screening applied to externally fetched data.

When both an AI-side curated manifest and a site-side published manifest exist for the same origin, the curated manifest SHOULD take precedence as the more explicit trust signal.

Method E is a transitional mechanism intended for domains that have not yet published a site-side AFRM. Once the site publishes its own manifest, the AI-side copy SHOULD be retired or used only to override individual fields.

3.2. Manifest Schema (AFRM Three-Part Model)

An AI Friction-Recovery Manifest (AFRM) is a JSON object [RFC8259] with the following top-level fields:

version (string, REQUIRED) Specification version. This document defines version "1.0".

publisher (string, REQUIRED) Canonical domain name of the publishing website.

manifestId (string, REQUIRED) Stable identifier scoped to the publisher, used for registry lookup.

registry_url (string, REQUIRED) HTTPS URL of the trust registry that the publisher has pre-registered this manifest with.

frameworkHints (object, OPTIONAL) Metadata about the UI framework in use (e.g., framework name, version, rendering model). Allows the AI agent to apply framework-specific heuristics before encountering friction.

knownTraps (array, REQUIRED) Primary content of the AFRM. An ordered array of trap descriptors, each describing a UI pattern that predictably blocks autonomous AI agents. Each element MUST include trapId (string), category (string from the IANA trap registry, see Section 5), selector (string), and escapeAction (string). Optional fields include description and condition.

shortcuts (array, OPTIONAL) An array of direct navigation or action shortcuts that allow the AI agent to bypass multi-step UI flows when a direct API or deep-link equivalent exists.

3.3. Agent Detection Algorithm

Upon loading a page, an AI agent implementing this specification SHOULD perform the following detection sequence before any full-DOM inference pass:

1. Check the agent's local file system for a user-curated manifest at the vendor-conventional path (e.g., ~/.claude/afrm/{domain}.json) matching the current origin (Method E). If found, load it into the trust context and continue; the site-side steps below may still be consulted for supplementary data.

2. Inspect the HTTP response headers for X-AI-Manifest (Method C).

3. If absent, retrieve /.well-known/ai-manifest.json (Method A) or resolve the URI declared by the meta element.

4. If still absent, check the document head for a <link rel="ai-manifest"> element and fetch the referenced URL (Method D).

5. If still absent, search the DOM for an element with id="ai-manifest" and read its data-manifest attribute (Method B).

6. If none is found, fall back to conventional DOM-based inference.

3.4. Canonical Hash and Trust Verification

A trust registry is an OPTIONAL integrity layer. When a site-published manifest declares a registry_url, the AI agent SHOULD compute a SHA-256 hash over the canonical form (see Section 2) of the manifest and send a trust lookup request to that URI before relying on the manifest. Such a request MUST use HTTPS [RFC2818] and MUST carry the tuple {publisher, manifestId, hash} as a JSON body. A user-curated manifest obtained via Method E is trusted as part of the user's own authored context and does not require a registry lookup. Because the manifest is advisory and the agent retains control of every interaction (see Section 3.5), a deployment MAY operate without a registry, in which case the agent relies on its own validation of each suggested action.

The registry response is a JSON object containing a status field with one of the following values:

* "white" — the manifest is trusted; the agent MAY proceed to execution.

* "black" — the manifest is explicitly distrusted; the agent MUST abort and SHOULD alert the human user.

* "unknown" — the manifest is not registered; the agent SHOULD warn the user and MAY fall back to DOM-based inference.

Implementations MAY cache a non-expired registry response keyed by the manifest hash, to avoid repeated network round trips for an identical manifest.

3.5. Consulting the Manifest

The AFRM is advisory reference data, not a mandatory execution script. An AI agent operates with its own capabilities and consults the manifest when it would otherwise expend effort rediscovering site-specific behavior. Two consultation patterns are typical:

* Reactive (friction recovery): when the agent encounters a UI condition matching a knownTraps entry's category and selector, it applies the associated escapeAction instead of recovering by trial and error. This is the primary use.

* Proactive (shortcut): when a shortcuts entry offers a direct navigation or action equivalent to a multi-step UI flow, the agent MAY take it directly.

The agent retains full control of the interaction and MAY continue to use its own DOM- or vision-based inference at any time; the manifest supplements that inference and does not replace it.

4. Central Trust Registry

A Central Trust Registry accepts manifest registrations from publishers and answers real-time hash lookups from AI agents. A conforming registry SHOULD:

* Store the SHA-256 hash of the canonical form of each registered manifest, together with the publisher and manifestId fields.

* Perform static analysis on the submitted manifest (its knownTraps and shortcuts entries) and reject or black-list manifests whose selectors or actions match a published pattern of prompt-injection risk (for example, selectors targeting iframe elements for cross-origin form submission, or actions outside the registered action set).

* Expose a community-driven mechanism for reporting and black-listing malicious manifests.

This document does not mandate a specific registry operator. Multiple interoperable registries MAY exist, and each manifest declares which registry is authoritative for it via registry_url.

5. IANA Considerations

5.1. Well-Known URI Registration

This document requests IANA to register the following entry in the "Well-Known URIs" registry established by [RFC8615]:

URI Suffix: ai-manifest.json

Change Controller: Independent Submission Stream editor

Reference: This document

Status: provisional

Related Information: None

5.2. AI Manifest Actions Registry (initial)

This document requests IANA to create a new registry named "AI Manifest Actions", with the following initial registrations. Registration policy: Specification Required [RFC8126].

click Invoke a click event on the selected element.

fill Type a value into a text input element.

select Choose an option from a drop-down list element.

upload Attach a file to a file input element.

wait Pause for a condition or duration.

navigate Change the current URL.

assert Verify that a condition holds before proceeding.

5.3. AI Manifest Trap Categories Registry (initial)

This document requests IANA to create a new registry named "AI Manifest Trap Categories", with the following initial registrations. Registration policy: Specification Required [RFC8126].

shadow-dom-trap A UI component rendered inside a Shadow DOM tree whose selectors are not accessible via standard document.querySelector calls.

virtual-scroll-trap A list or grid that renders only a visible

viewport window, requiring programmatic scrolling before off-screen items can be targeted.

iframe-context-trap An action target located inside a cross-origin or same-origin <iframe> that requires a context-switch before interaction.

native-dialog-trap A browser-native modal dialog (invoked via window.alert(), window.confirm(), or window.prompt()) that freezes the renderer event loop until dismissed. An AI agent must call the appropriate WebDriver dismiss or accept command rather than attempting a DOM interaction, which will block indefinitely.

delayed-render-trap A UI element that does not appear in the DOM until after an asynchronous operation completes; the agent must wait for the element’s visibility predicate before acting.

5.4. Link Relation Type Registration

This document requests IANA to register the following entry in the "Link Relations" registry established by [RFC8288]:

Relation Name: ai-manifest

Description: Refers to an AI Friction-Recovery Manifest (AFRM) that describes known UI traps, framework hints, and interaction shortcuts for the current origin, to assist autonomous AI agents.

Reference: This document

6. Security Considerations

6.1. Prompt Injection Risk

A malicious website could publish an AI Manifest whose entries lead an AI agent to perform actions harmful to the user (for example, submitting a form to a third party with user-supplied credentials). Two complementary mitigations apply. First, the manifest is advisory: the agent consults it on friction and retains control of every interaction (see Section 3.5), so a suggested action is subject to the agent’s own safety judgment rather than executed blindly. Second, when a manifest declares a registry, the trust registry mechanism (Section 4) provides an integrity and reputation check. Agents MUST NOT act on a manifest whose registry lookup returns "black" and SHOULD warn the user before relying on an "unknown" manifest.

6.2. Interaction with LLM Agent Prompt Injection Defenses

Modern Large Language Model (LLM) agents (including but not limited to Claude, ChatGPT, and Gemini) implement prompt-injection defense as an immutable system-level policy. Such policy treats web-page-embedded content that contains executable instructions, including AI Manifests, as untrusted external data and requires explicit user approval prior to execution.

A preliminary single-trial measurement by the author (2026-04-28, one trial per condition) on an inline-embedded manifest observed that an LLM agent processing a page-embedded manifest directly was slower than the no-manifest baseline, because the agent classified the embedded content as a possible prompt-injection pattern and reverted to manual navigation. That trial led the -01 revision to recommend a deterministic runtime. Two factors limit that conclusion: the manifest was embedded inline (Method B) rather than fetched out-of-band, and the comparison was a single trial whose fastest figure excluded the agent’s own reasoning time.

Subsequent measurement with a contemporary LLM agent and an out-of-band manifest does not reproduce the negative result. When the manifest is discovered out-of-band -- at the well-known URI (Method A) or from the user-curated path (Method E) -- it is not treated as injected page content, and the agent consults it without reverting to manual navigation. On a proprietary-knowledge enterprise task the agent completed the task both with and without the manifest, and the manifest reduced interaction actions and input tokens (see Section 8).

Implications for deployment:

* A manifest discovered out-of-band (Method A or Method E) SHOULD be preferred over an inline-embedded manifest (Method B), because out-of-band data is not subject to the agent’s page-content prompt-injection screening.

* A site-published manifest, consumed directly by the LLM agent with no external runtime, is sufficient to provide value. Implementations MAY additionally pair the manifest with a deterministic runtime that performs discovery, registry verification, and execution outside the LLM context for latency- or service-level-sensitive deployments, but such a runtime is OPTIONAL.

* Because the manifest is advisory data consulted only on friction, an agent that ignores it falls back to its baseline behavior; the manifest does not reduce the agent’s capability below baseline.

This finding does not alter the protocol on the wire but informs deployment patterns and the OPTIONAL role of a runtime component vis-a-vis the LLM agent.

6.3. Integrity of the Manifest

The SHA-256 hash is computed over the canonical form of the manifest so that semantically equivalent encodings produce identical digests. Implementations MUST NOT rely on a hash computed over non-canonical bytes.

6.4. Transport Security

All communication with the registry MUST use HTTPS with server authentication per [RFC2818]. Registry operators SHOULD sign their responses with a public key published out of band so that an AI agent can verify the integrity of a cached response.

7. Privacy Considerations

Registry lookups necessarily expose to the registry operator the fact that a particular AI agent has visited a particular publisher’s manifest. Registry operators SHOULD minimize the retention of client identifiers associated with lookup requests. Agents MAY employ private, time-limited caching of registry responses to reduce the frequency of such lookups.

8. Implementation Status

Note to RFC Editor: This section is intended to be removed prior to publication as an RFC.

A reference implementation, including an example publisher server, a reference registry, two AI agent variants (DOM-analysis baseline and manifest-aware), and an automated benchmark harness, is available at https://github.com/11pyo/AINavManifest under the MIT License.

In the reference benchmark — a two-step ERP order-entry transaction repeated 30 times with input tokens counted via the tiktoken cl100k_base encoding — the manifest-aware agent consumed an average of 341 input tokens per task with a 100% task success rate (30 of 30 runs), while the DOM-analysis baseline consumed an average of 1887.6 input tokens with a 20% success rate (6 of 30 runs). Raw results accompany the reference implementation.

A subsequent measurement (2026-05, Claude Sonnet 4.6 browser agent) used a proprietary-knowledge enterprise task: an SAP S/4HANA-style table-maintenance screen on which the valid account-category

references depend on a company-code registration scope (the standard codes exist globally but are not registered for this company code, so they are rejected), and on which a cell edit must be confirmed with a commit gesture before the row unlocks. The correct values are not derivable from public or training knowledge. Two conditions were compared with a contemporary LLM agent and no external runtime: a no-manifest baseline and a site-published manifest. Both conditions completed the task and saved successfully. The manifest condition used approximately 54% fewer interaction actions (6 versus 13) and approximately 45% fewer input tokens (about 47.8K versus 87.3K), with fewer errors and fewer reasoning pauses. Wall-clock time was closer (about -22%), because at this difficulty the agent's own reasoning time dominates in both conditions; interaction actions and input tokens reflect the efficiency difference more directly than wall-clock time does.

The earlier single-trial three-tier figures reported in draft-han-ai-manifest-01 (a "Tier C" embedded-manifest-plus-SDK time of 5.0 s, with 19- to 31-fold speedup claims) are superseded by this measurement and SHOULD NOT be cited: that "Tier C" figure measured a deterministic runtime executing cached selectors and excluded the agent's reasoning time, so it is not comparable to unaided-agent wall-clock numbers.

A controlled experiment (Phase 1, n=30 trials, 2026-04-28) examined whether an AI agent spontaneously detects and loads a site-side AFRM during routine task execution without explicit user instruction. The primary hypothesis (H1: manifest_detected fires at least once during 30 trials) was rejected: spontaneous detection was not observed. This indicates that reliable consumption should not depend on the agent volunteering to fetch an unannounced manifest; it motivates explicit out-of-band discovery (Method A with a visible link or footer hint, or the Method C/D headers) and the user-curated Method E, rather than a requirement for a deterministic runtime. The token-reduction figures from this simulation were large but were computed against an idealized fixed-length manifest baseline whose variance approached zero, which inflates standardized effect-size estimates; the previously cited Cohen's d = 5.00 is therefore an artifact of that idealization and is withdrawn. A calibrated re-analysis that injects realistic run-to-run variability onto the measured means keeps the action- and token-reduction effects large and stable without relying on the idealized figure. Raw data and analysis scripts accompany the reference implementation.

A cross-AI interoperability study (Phase 2, 2026-05-11) tested whether three major LLM agents (Claude, ChatGPT, Gemini) can consume an AFRM presented directly in a conversation context. Results: Claude correctly extracted and applied the AFRM content (trap

descriptors, selectors, escape actions) when the manifest was provided in-context; ChatGPT demonstrated label-level recognition (identified the document as an AFRM) but did not apply individual trap entries procedurally; Gemini returned generic speculation without manifest-specific extraction. These results suggest that procedural AFRM consumption at the level of individual knownTraps entries is not yet uniform across LLM agents. For agents that do not yet consume per-trap entries reliably, an OPTIONAL deterministic runtime can bridge the gap; for agents that do (as demonstrated above), a site-published manifest is consumed without one.

9. Intellectual Property Rights Disclosure

The technology described in this document is the subject of Korean Patent Applications No. 10-2026-0071716 (filed 2026-04-21) and No. 10-2026-0097280 (filed 2026-05-29), both by the author. The applicant commits to offer any essential claims under Fair, Reasonable, and Non-Discriminatory (FRAND) terms to implementers of this specification, as declared in the project repository.

10. Acknowledgments

The author thanks the Anthropic Claude Code, Model Context Protocol, and OpenAI function-calling communities for the empirical observations that motivated this work.

11. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <https://www.rfc-editor.org/info/rfc2818>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <https://www.rfc-editor.org/info/rfc8615>.

[RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <https://www.rfc-editor.org/info/rfc8288>.

Appendix A. Document History

Note to RFC Editor: This section is intended to be removed prior to publication as an RFC.

draft-han-ai-manifest-02 (2026-06) Realigns the document with the friction-recovery model. The manifest is described as advisory, site-published reference data discovered out-of-band, consulted on friction, and orthogonal to (not a modification of) any browser-automation tool schema. The deterministic-runtime ("SDK") recommendation of -01 is relaxed from REQUIRED/SHOULD to OPTIONAL/MAY. The single-trial "Tier C" 5.0 s figure and the 19- to 31-fold speedup claims are superseded and withdrawn, as is the Cohen’s d = 5.00 effect size (an artifact of a near-zero-variance idealized baseline). A multi-condition measurement with a contemporary LLM agent on a proprietary-knowledge enterprise task is added (approximately -54% interaction actions and -45% input tokens, both conditions completing). The Execution section is reframed as advisory consultation rather than deterministic step execution. The title is updated accordingly.

draft-han-ai-manifest-01 (2026-05-15) Extended the schema to the AFRM three-part model (frameworkHints, knownTraps, shortcuts); added Method E (AI-side curation) and the native-dialog-trap category.

draft-han-ai-manifest-00 (2026-04-21) Initial version: embedding methods A through C, the central trust registry, and canonical hash verification.

Author’s Address

Won-pyo Han
Individual
Korea, Republic of

Email: pk102h@naver.com