Independent Submission
Internet-Draft
Intended status: Informational
Expires: 16 November 2026

W. Han
Individual

15 May 2026

AI Manifest: Embedded Workflow Instructions for AI Agents
draft-han-ai-manifest-01

Abstract

This document specifies the AI Manifest protocol, a JSON-based format for websites to declare step-by-step user interface (UI) workflow instructions readable by autonomous AI agents. By embedding the manifest, website operators allow AI agents using browser-automation tools to execute multi-step transactions directly via Cascading Style Sheets (CSS) selectors, without repeated analysis of the full Document Object Model (DOM). The specification defines three interoperable embedding methods, a SHA-256 canonical hash verification procedure via a central trust registry, and security mitigations against prompt injection attacks.

Empirical results from a reference implementation demonstrate an 81.9% reduction in input tokens consumed by the AI agent and an increase in task success rate from 20% to 100% on a representative multi-step transaction, compared with conventional DOM-analysis approaches.

This document also documents an empirical finding (-01 update, 2026-04-28) that manifest embedding alone is insufficient for autonomous execution by modern LLM agents due to prompt-injection defenses in their system policies. A deterministic execution runtime (referred to herein as the "SDK") is required to consume the verified manifest outside the LLM reasoning loop and return only the structured outcome to the agent. Without such a runtime, an embedded manifest may produce negative value (slower than baseline) due to user-approval-gate latency.

A -01 update (2026-05-15) introduces two additions. First, the manifest schema is extended to the AI Friction-Recovery Manifest (AFRM) three-part model: frameworkHints, knownTraps (primary content), and shortcuts. This aligns with the independently observed structure of real-world practitioner documents and with the companion position paper (IUI 2027). Second, a fifth discovery mechanism is specified -- Method E (AI-side curation) -- in which the user places a manifest at a vendor-conventional file-system path (e.g., ~/.claude/afrm/{domain}.json); the AI runtime loads it on domain match without prompt-injection screening because the file originates from the user's authored context. A new trap category, native-dialog-trap, is also registered.

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 enumerates ordered UI operations keyed to CSS selectors. An AI agent detects, parses, and verifies the manifest before executing the listed steps, and avoids further DOM-based inference for those steps.

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 A structured data object, expressed in JSON, that a website operator embeds into a web page or serves at a well-known URI, containing at minimum a task identifier, an ordered steps array, and for each step an action field and a CSS selector.

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

Prior to execution, the AI agent MUST compute a SHA-256 hash over the canonical form (see Section 2) of the manifest and send a trust lookup request to the URI in the registry_url field. The request MUST use HTTPS [RFC2818] and MUST carry the tuple {publisher, manifestId, hash} as a JSON body.

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. Execution

When trust is confirmed, the agent executes the steps array in declared order, mapping each step's action and selector to a browser-automation primitive (e.g. "click", "fill", "select", "upload"). For the duration of a manifest-driven execution the agent SHOULD NOT perform additional LLM-based inference over the page DOM.

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 steps array 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 embed an AI Manifest whose steps array leads an AI agent to perform actions harmful to the user (for example, submitting a form to a third party with user-supplied credentials). The Central Trust Registry mechanism (Section 4) is the primary mitigation. Agents MUST NOT execute a manifest whose registry lookup returns "black" and SHOULD warn the user before executing 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.

Empirical study by the author (2026-04-28) on a representative enterprise multi-step UI (one trial per condition) compared three configurations:

* Tier A — Native page without manifest: 97.7 seconds total wall-clock to task completion;

* Tier B — Embedded manifest with the LLM agent processing the manifest directly: 154.7 seconds, due to user-approval gate latency. The agent in this trial classified the manifest content as a prompt-injection pattern and reverted to manual page navigation, defeating the manifest's intent;

* Tier C — Embedded manifest with a deterministic runtime (SDK) consuming the manifest outside the LLM reasoning loop: 5.0 seconds, with no user-approval interruption.

Implications:

* Manifest embedding alone (Method B without a runtime) SHOULD NOT be relied upon as a complete automation surface; in practice it may produce negative value due to repeated user-approval prompts.

* Implementations SHOULD pair the manifest with a deterministic runtime that performs discovery, registry verification, static analysis, and step execution outside the LLM context, returning only a structured success/failure summary to the LLM.

* Externally-referenced manifests (Method A, /.well-known/) avoid prompt-injection-defense triggering when the runtime fetches the manifest out-of-band, although they require additional HTTP round-trips.

This finding does not alter the protocol on the wire but informs deployment patterns and the role of the 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 three-tier comparison study (2026-04-28, one trial per condition) used a representative enterprise approval UI ("IRIS-style vacation request") and the same LLM (Claude) across the three configurations described in Section 6.2. Wall-clock results: Tier A (native, no manifest) 97.7 s; Tier B (manifest embedded, processed by the LLM directly) 154.7 s; Tier C (manifest embedded plus deterministic SDK runtime) 5.0 s. Tier C achieved approximately a 19-fold speedup over Tier A and a 31-fold speedup over Tier B. The surprising Tier-B-slower-than-Tier-A result is attributed to the LLM agent rejecting the embedded manifest as a possible prompt-injection pattern and reverting to manual navigation, consistent with LLM safety policy.

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: the event was never observed. This result confirms that a deterministic SDK runtime, not unaided LLM inference, is required to achieve reliable manifest consumption. All eight pre-registered secondary hypotheses (H2-H8, covering token reduction,

task-completion rate, and step-accuracy metrics conditioned on successful detection) were accepted with large effect sizes (Cohen's d = 5.00 for token reduction in the SDK condition). 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 AFRM consumption at the level of individual knownTraps entries is not yet uniform across LLM agents, further motivating the deterministic SDK runtime architecture specified in this document.

9. Intellectual Property Rights Disclosure

The technology described in this document is the subject of Korean Patent Application No. 10-2026-0071716, filed on 2026-04-21 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>.

Author’s Address

Won-pyo Han
Individual
Korea, Republic of
Email: pk102h@naver.com