Independent Submission
Internet-Draft
Intended status: Informational
Expires: 23 October 2026

W. Han
Individual

21 April 2026

AI Manifest: Embedded Workflow Instructions for AI Agents
draft-han-ai-manifest-00

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.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 23 October 2026.

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.

3.2. Manifest Schema

An AI Manifest 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.

task (object, REQUIRED) Contains a task-level id and an ordered steps array. Each element of steps is an object with at least step (integer), action (string), and selector (string). The action value MUST be one of the registered actions (see Section 5).

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

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

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

4. 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.

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. 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.

6.3. 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.

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>.

Author’s Address

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