Web Agent API

Raffi Krikorian <raffi@mozilla.org>

Changelog

- **February 11, 2026** - Added discussion of Chrome's WebMCP announcement and how Web Agent API complements it. Clarified supply-side vs. demand-side framing.
- **January 2026** - Initial draft published for review.

Abstract

This specification defines the Web Agent API, a browser-level interface for AI capabilities on the web. It enables web applications to access language models and tools through a standardized API surface, with user-controlled permissions and provider choice. The specification builds on the Model Context Protocol (MCP) for tool integration while remaining extensible to future protocols.

Introduction

Problem Statement

User context is scattered across web services. When AI applications access this context, connections happen through third-party integrations on terms users don't control. Users cannot bring their preferred model to arbitrary websites. Developers rebuild AI integration infrastructure independently. No standard exists for browser-mediated AI with user-controlled tool access.

Design Goals

- **User agency -** users choose their AI provider and control which context is shared with which origins;
- **Developer simplicity -** web applications access AI through a standard API without managing model connections or inference costs;
- **Interoperability -** applications work across implementations. Compatibility with Chrome's Prompt API where applicable;
- **Extensibility -** the architecture accommodates protocol evolution beyond MCP; and
- **Security -** defense in depth with origin isolation, capability-based permissions, and explicit user consent.

Relationship to MCP

The Model Context Protocol (MCP) defines how AI systems connect to tools via JSON-RPC. Implementations support MCP for tool communication. Implementations may support additional tool protocols.

Relationship to WebMCP

On February 10, 2026, Chrome announced **WebMCP** - a proposal for how websites expose structured tools to AI agents. WebMCP provides declarative and imperative APIs for websites to define actions agents can take: booking flights, filing support tickets, navigating checkout flows. This is the **supply side**: how websites become agent-ready.

The Web Agent API addresses the **demand side**: who controls the AI, where context lives, and how permissions work. These are complementary. WebMCP formalizes how websites expose capabilities. The Web Agent API formalizes how users control the AI that consumes those capabilities.

API Specification

The window.ai Namespace

The ai namespace provides access to language model capabilities. It is compatible with Chrome's Prompt API where functionality overlaps.

ai.languageModel

```
interface LanguageModelFactory {
  Promise<LanguageModelCapabilities> capabilities();
  Promise<LanguageModelSession> create(optional LanguageModelOptions
options);
};

dictionary LanguageModelOptions {
  DOMString systemPrompt;
  float temperature;          // 0.0 to 1.0
  unsigned long maxTokens;
  DOMString provider;         // Extended: provider selection
  DOMString model;            // Extended: model selection
  boolean allowTools;         // Extended: enable tool calling
};

interface LanguageModelSession {
  Promise<DOMString> prompt(DOMString input);
```

```
  ReadableStream promptStreaming(DOMString input);
  void destroy();
};
```

`capabilities()` returns available model capabilities including supported providers and models. `create()` establishes a session. The provider and model options are extensions to Chrome's API.

ai.providers (Extended)

```
interface AIProviders {
  Promise<sequence<ProviderInfo>> list();
};

dictionary ProviderInfo {
  DOMString id;               // e.g., 'anthropic', 'openai', 'ollama'
  DOMString name;
  sequence<DOMString> models;
  boolean isLocal;
};
```

ai.runtime (Extended)

```
interface AIRuntime {
  Promise<RuntimeSelection> getBest(optional RuntimeCriteria criteria);
};

dictionary RuntimeCriteria {
  boolean preferLocal;        // Prefer on-device execution
  boolean requireTools;       // Must support tool calling
  DOMString taskType;         // Hint: 'chat', 'code', 'analysis'
};
```

The window.agent Namespace

The agent namespace provides tool access and autonomous execution capabilities. This namespace is an extension not present in Chrome's Prompt API.

agent.tools

```
interface AgentTools {
  Promise<sequence<ToolDefinition>> list(optional ToolListOptions options);
  Promise<ToolResult> call(DOMString toolName, object arguments);
};
```

```
dictionary ToolDefinition {
  DOMString name;             // Namespaced: 'server_id/tool_name'
  DOMString description;
  object inputSchema;         // JSON Schema
  DOMString serverId;
};

dictionary ToolResult {
  boolean success;
  any content;
  DOMString? error;
  DOMString? errorCode;
};
```

agent.run()

```
interface AgentRunner {
  AsyncIterable<AgentEvent> run(AgentRunOptions options);
};

dictionary AgentRunOptions {
  DOMString task;             // Natural language task description
  unsigned long maxToolCalls; // Default: 5
  unsigned long timeoutMs;    // Default: 300000 (5 minutes)
  sequence<DOMString> tools;  // Restrict to specific tools
};

dictionary AgentEvent {
  DOMString type;             // See event types below
  any data;
};
```

Agent event types:

Event Type	Data	Description
`status`	`{ state: string }`	`Lifecycle: 'started', 'thinking', 'complete'`
`tool_call`	`{ tool: string, args: object }`	Tool invocation beginning
`tool_result`	`{ tool: string, result: ToolResult }`	Tool invocation complete
`token`	`{ text: string }`	Streaming token from model

`final`	`{ response: string }`	Final agent response
`error`	`{ code: string, message: string }`	Error occurred

agent.browser

```
interface AgentBrowser {
  AgentActiveTab activeTab;
};

interface AgentActiveTab {
  Promise<TabContent> readability();
};

dictionary TabContent {
  DOMString url;
  DOMString title;
  DOMString text;              // Readability-extracted content
  DOMString html;              // Sanitized HTML
};
```

The `readability()` method requires the `browser:activeTab.read` permission.

agent.mcp

```
interface AgentMCP {
  Promise<sequence<MCPServerInfo>> discover();
  Promise<void> register(MCPServerConfig config);
};

dictionary MCPServerInfo {
  DOMString id;
  DOMString title;
  DOMString url;
  DOMString origin;            // Declaring page origin
};

dictionary MCPServerConfig {
  DOMString url;
  DOMString title;
  object? headers;             // Auth headers
};
```

MCP Server Discovery

Declarative Discovery

Websites declare MCP servers using a `link` element:

```
<link rel="mcp-server"
      href="https://example.com/mcp"
      title="Example Tools">
```

Programmatic Discovery

`agent.mcp.discover()` returns all MCP servers declared on the current page. Servers from other origins are included but marked with their declaring origin.

Programmatic Registration

`agent.mcp.register()` allows runtime registration of MCP servers. This requires the `mcp:servers.register` permission. Registered servers persist for the session only.

Permission Model

A decade ago, browsers introduced permission prompts for cameras and microphones—sensitive resources that websites could request but not access without explicit user consent. The Web Agent API extends this model to AI and context. Your AI, your preferences, your accumulated context: these are resources the browser manages on your behalf. Websites don't get access by default. They request it, and you grant or deny.

All operations require explicit user consent. Permissions are granted per-origin with complete isolation between origins.

Permission Scopes

All operations require explicit user consent. Permissions are granted per-origin.

Scope	Risk	Description
`model:prompt`	Low	Basic text generation
`model:tools`	Medium	AI with tool calling enabled
`mcp:tools.list`	Low	List available tools
`mcp:tools.call`	High	Execute tools (requires allowlist)

`mcp:servers.register`	Medium	Register MCP servers at runtime
`browser:activeTab.read`	Medium-High	Read current page content
`context:identity`	High	Access user identity
`context:payment`	High	Access payment methods

Grant Types

Grant Type	Duration	Behavior
`ALLOW_ONCE`	10 minutes or tab close	Temporary grant for single interaction
`ALLOW_ALWAYS`	Persistent	Stored grant, no re-prompting
`DENY`	Persistent	Blocks future permission prompts

Tool Allowlisting

The `mcp:tools.call` permission is necessary but not sufficient. Even with this permission granted, an origin has access to *no* tools by default. Users must explicitly allowlist specific tools for each origin. This prevents a blanket "allow tools" grant from exposing all available tools to any website.

Origin Isolation

Each origin operates in complete isolation: separate permission grants, tool allowlists, session state, rate limits, and MCP registrations. Origin is verified by the browser implementation, not by page-provided values.

Implementation Considerations

This specification does not mandate a particular implementation architecture. Conforming implementations may use:

- A browser extension bridging to an external process hosting MCP servers;
- A browser extension with WASM runtime for in-browser MCP hosting;
- Native browser integration with built-in MCP hosting;
- An OS-level service providing the backend; or
- A cloud proxy handling tool execution for lightweight clients.

All implementations expose the same window.ai and window.agent surface to web applications, and all implementations speak MCP to tools. Web applications written against the API work regardless of how a particular browser implements the backend.

Enterprise Considerations

The browser layer enables enterprise policy control. Organizations that need to govern which AI models employees use can enforce this through browser policy—the same mechanism already used to manage extensions, security settings, and site permissions. An IT administrator could restrict AI providers to an approved list, require local-only execution for sensitive workloads, or route all inference through a corporate endpoint. The Web Agent API respects these policies.