Internet-Draft
draft-serra-mcp-discovery-uri-04
Intended status: Informational
Expires: 25 September 2026

M. Serra
Mumble Group

25 March 2026

The "mcp" URI Scheme and MCP Server Discovery Mechanism
draft-serra-mcp-discovery-uri-04

Abstract

The Model Context Protocol (MCP) defines a standard interface for AI agents to connect to external tools and services. However, no standard mechanism exists for an AI agent to autonomously discover which web domains expose an MCP server.

This document defines:

(1) the "mcp" URI scheme, a machine-to-machine identifier for publicly reachable MCP servers;

(2) a two-mode discovery mechanism: a well-known URI (/.well-known/mcp-server) for universal compatibility, and a DNS TXT record format for DNS-native fast discovery;

(3) an optional manifest integrity mechanism using JSON Web Signatures (JWS, [RFC7515]) with keys published via DNS;

(4) a security capability negotiation mechanism through which the manifest declares trust class, authentication requirements, compliance frameworks, and logging policy before connection.

This specification does not define the MCP protocol itself, nor authentication between agent and server. Those are covered by the MCP specification and OAuth 2.1 respectively.

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 25 September 2026.

Copyright Notice

Table of Contents

1. Introduction

1.1. Problem Statement

The Model Context Protocol [MCP-SPEC] defines how AI agents communicate with servers that expose tools, resources, and prompts. However, MCP does not define how an agent discovers that a server exists in the first place. Given a web domain such as "example.com", there is no standard mechanism by which a conforming MCP client can autonomously determine whether an MCP server is available, where it is located, or what security posture it requires before connecting.

This gap forces every MCP deployment to rely on out-of-band configuration: hardcoded endpoints, manually maintained registries, or proprietary discovery mechanisms. At scale, this prevents autonomous agent operation and fragments interoperability across the ecosystem.

1.2. This Document

This document defines:

- The "mcp" URI scheme, providing a standard machine-to-machine identifier for MCP servers (mcp://example.com).
- A JSON manifest published at a well-known HTTPS location (/.well-known/mcp-server), through which a server declares its endpoint, capabilities, and security requirements.
- A DNS TXT record format (_mcp.{host}) enabling DNS-native discovery for operators who control their DNS infrastructure.
- A resolution sequence clients MUST follow, with two explicit operating modes (Section 4):

  base mode: .well-known only. Works on any web server with no DNS configuration required.

  fast mode: DNS TXT first, .well-known for metadata enrichment. For operators who control DNS and require faster, more reliable indexing.

- A security capability negotiation mechanism (Section 6.10), through which the manifest declares the server’s trust class, authentication requirements, compliance frameworks, and logging policy before any connection is established.

1.3. Design Principles

Three principles have guided the design of this specification:

Secure by default. A manifest with no explicit security declaration is treated as "trust_class: public" -- the most permissive safe posture. More restrictive postures MUST be declared explicitly by the server operator. Absence of a declaration is never interpreted as elevated privilege.

Independently deployable layers. This document defines the discovery and security negotiation layer only. It does not define the MCP protocol itself, authentication flows beyond their advertisement in the manifest, or accountability mechanisms for tool call recording. Each layer is deployable without the others.

Accessible to non-technical operators. The discovery mechanism is designed to be implementable by CMS plugins, shared hosting environments, and operators without DNS control. The base mode requires installing a single JSON file. The security capability

negotiation is designed to be driven by user-facing configuration interfaces without requiring operators to author JSON directly (Section 6.10.7).

1.4. Relationship to Other Work

SEP-1649 and SEP-2127 (MCP Server Cards), authored within the Model Context Protocol project, propose a /.well-known discovery endpoint and server metadata format for HTTP-based MCP servers. This document pursues the same goal through the IETF standards process, which provides a vendor-neutral venue for IANA registration of both the .well-known suffix and the "mcp" URI scheme -- consistent with MCP's status as a Linux Foundation project with broad industry participation. The authors welcome coordination with the SEP process to converge on a unified discovery mechanism.

Community feedback on SEP-2127 has identified that operators on fully managed hosting platforms (such as Wix, Squarespace, and similar services) cannot serve arbitrary files at /.well-known paths, as this capability is controlled by the hosting provider rather than the domain owner. This constraint affects a significant portion of the web, including an estimated 43% of sites running on WordPress and similar CMS platforms in shared hosting environments. The base mode defined in Section 4.1 of this document addresses this gap by requiring only a single JSON file installable via a CMS plugin, with no DNS configuration required.

[ACTA] defines a signed receipt mechanism for MCP tool call decisions. That document addresses the accountability layer -- what is recorded after an agent uses a server. This document addresses the discovery layer -- how an agent finds a server and what the server declares about itself before the first tool call. The two documents are complementary and independently deployable. The "trust_class" and "compliance" fields defined in Section 6.10 of this document provide the jurisdictional context that accountability layers MAY use to apply appropriate verification policies.

The broader layering of the MCP ecosystem as understood by this document is as follows:

```
Layer 0  DNS bootstrap (_mcp TXT record, fast mode only)
Layer 1  Discovery and security negotiation (this document)
Layer 2  Connection metadata (SEP-1649, SEP-2127)
Layer 3  MCP protocol (transport, tool invocation)
Layer 4  Accountability ([ACTA])
```

In fast mode, Layers 0-1 are traversed once at connection setup and the result cached. In base mode, Layer 0 is skipped and discovery begins at Layer 1. Layer 3 is the ongoing protocol session. Layer 4 is activated per tool call only when required by the server's declared security posture.

1.5. Requirements Language

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.

1.6. Out of Scope

This document does NOT define:

- The MCP protocol itself (defined in [MCP-SPEC])

- Authentication flows beyond their advertisement in the manifest
- The format of MCP tools, resources, or prompts (defined in [MCP-SPEC])
- Discovery of MCP servers on private networks or intranets
- Accountability or audit mechanisms for tool call recording
- A crawling or indexing infrastructure (implementation concern)

2. Terminology

MCP Server: A service exposing tools, resources, and/or prompts via the Model Context Protocol [MCP-SPEC].

MCP Client: An AI agent or application that connects to MCP Servers to invoke tools and retrieve resources.

MCP URI: A URI using the "mcp" scheme as defined in Section 3 of this document, identifying a publicly reachable MCP Server by domain.

Manifest: The JSON document returned at /.well-known/mcp-server, as defined in Section 6 of this document.

Discovery: The process by which an MCP Client determines the endpoint URL of an MCP Server starting from a domain name alone.

3. The "mcp" URI Scheme

3.1. Purpose

The "mcp" URI scheme identifies a publicly reachable MCP Server associated with a web domain. It is intended for use by AI agents and automated crawlers, not by human users or web browsers.

3.2. Syntax

The "mcp" URI scheme is registered following the procedures defined in [RFC7595]. The syntax is defined using ABNF [RFC5234]:

```
mcp-URI       = "mcp://" authority path-abempty [ "?" query ]
authority     = [ userinfo "@" ] host [ ":" port ]
host          = IP-literal / IPv4address / reg-name
reg-name      = *( unreserved / pct-encoded / sub-delims )
path-abempty  = *( "/" segment )
segment       = *pchar
query         = *( pchar / "/" / "?" )
```

Where "userinfo", "IP-literal", "IPv4address", "unreserved", "pct-encoded", "sub-delims", and "pchar" are as defined in [RFC3986].

3.3. Examples

Valid MCP URIs:

```
mcp://example.com
mcp://api.example.com
mcp://example.com/shop
mcp://example.com:8080
```

Invalid MCP URIs:

```
mcp://              (missing host)
mcp:example.com     (missing //)
```

4. MCP Server Discovery

4.1. Operating Modes

This document defines two operating modes for MCP discovery. Clients and server operators SHOULD declare which mode they support in their implementation documentation.

base mode: .well-known only. The client fetches /.well-known/mcp-server directly. This mode works on any web server and requires no DNS configuration. It is the RECOMMENDED default for operators on shared hosting, WordPress, or any environment where DNS control is unavailable.

fast mode: DNS TXT first, .well-known for metadata enrichment. The client queries _mcp.{host} first to confirm MCP presence before incurring TLS overhead. This mode is RECOMMENDED for operators who control DNS and for crawlers operating at scale, where a single UDP packet in <10ms per domain provides significant latency advantages over a full HTTPS GET per domain.

4.2. Discovery Sequence

Upon resolving an MCP URI, clients operating in fast mode MUST attempt discovery in the following order. Clients operating in base mode MUST begin at Step 2.

Step 1 -- DNS TXT Record (fast mode only)

The client MUST query:

```
_mcp.{host}  IN  TXT
```

If a TXT record is present containing "v=mcp1", the client has confirmed MCP presence and MUST proceed to Step 2 to retrieve the full manifest. If the record contains "src=", the DNS TXT record also provides the server endpoint address; in all cases .well-known provides the authoritative manifest.

If no such record exists, the client MUST proceed to Step 2.

Step 2 -- Well-Known URI (REQUIRED in both modes)

The client MUST perform an HTTP GET request to:

```
https://{host}/.well-known/mcp-server
```

with the header:

```
Accept: application/json
```

If the server responds with HTTP 200 and a valid JSON Manifest (Section 6), the client MUST use the endpoint specified in the Manifest.

If the server responds with HTTP 301 or 302, the client MUST follow the redirect, up to a maximum of two redirect levels.

If the server responds with HTTP 404, or the request times out (RECOMMENDED timeout: 5 seconds), the client MUST proceed to Step 3.

Step 3 -- Direct Endpoint (LAST RESORT)

The client MAY attempt a direct MCP handshake at:

```
https://{host}/mcp
```

If the handshake succeeds, the client MAY use this endpoint. If it fails, the client MUST report that no MCP server was found for the given domain.

4.3. Conflict Resolution

When DNS TXT and .well-known specify different endpoints, the .well-known manifest endpoint MUST take precedence. HTTPS provides domain authenticity guarantees that unsigned DNS records do not.

Clients SHOULD log a warning when the DNS TXT endpoint and the .well-known manifest endpoint diverge. This divergence MAY indicate misconfiguration or a DNS-level attack.

5. DNS TXT Record Format

This section defines the DNS TXT record format used in Step 1 of the fast mode discovery sequence (Section 4.2). DNS TXT records are defined in [RFC1035]. For operators who control DNS, a TXT record provides a lightweight discovery primitive that confirms MCP presence and advertises the server endpoint without requiring an HTTPS request.

5.1. Syntax

A domain MAY advertise one or both of the following record types:

```
_mcp.{domain}  IN  TXT  "v=mcp1; src={url}[; auth={type}]"
_mcp.{domain}  IN  TXT  "v=mcp1; registry={url}"
```

5.2. Fields

v=mcp1	MUST	Version discriminator.
src=<url>	SHOULD	Direct MCP server endpoint URL. Use when the domain hosts its own MCP server directly.
registry=<url>	MAY	URL of a catalogue or registry that lists MCP servers for this domain. Use when the domain aggregates multiple MCP servers via a registry backend.
auth=<type>	SHOULD	One of: none, apikey, oauth2.

A record containing "src=" advertises a direct server endpoint. A record containing "registry=" advertises a catalogue endpoint. Both MAY appear in the same record or as separate TXT records for the same domain.

The legacy field name "endpoint=" is deprecated and MUST be treated as equivalent to "src=" by conforming clients.

5.3. Limitations

DNS TXT records are limited to 255 characters per string. For complete manifests, operators MUST use the well-known URI mechanism (Section 4.2, Step 2). DNS TXT records are discovery primitives only; they do not replace the manifest.

6. The MCP Server Manifest

6.1. Format

The Manifest is a JSON document [RFC8259] returned at /.well-known/mcp-server.

6.2. Required Fields

mcp_version	string	MCP spec version (e.g. "2025-06-18")
name	string	Human-readable server name
endpoint	string	URL of the MCP endpoint
transport	string	See Section 6.6

6.3. Recommended Fields (SHOULD)

description	string	Natural language description of the server
auth	object	Authentication requirements (see Section 6.5)
capabilities	array	["tools", "resources", "prompts"]
trust_class	string	Security posture declaration. See Section 6.10.2. Values: "public" | "sandbox" | "enterprise" | "regulated". Default if absent: "public".

6.4. Optional Fields (MAY)

categories	array	Semantic categories (see mcpstandard.dev/categories)
languages	array	ISO 639-1 language codes
coverage	string	ISO 3166-1 country code
contact	string	Contact email or URL
docs	string	Documentation URL
last_updated	string	ISO 8601 timestamp
crawl	boolean	false to opt out of indexing
expires	string	ISO 8601 timestamp after which the manifest MUST be considered stale and re-fetched
cache_ttl	integer	Cache duration in seconds for this manifest. Overrides HTTP Cache-Control for the purpose of manifest staleness evaluation. Default: 3600. MAY be omitted for trust_class "public", "sandbox", "enterprise". REQUIRED when trust_class is "regulated" (see Section 6.10.3).
signature	object	Manifest integrity signature (see Section 6.7)
server_card	string	URL of a SEP-2127 compatible Server Card document, providing richer pre-connection metadata including tool descriptions, resources, and prompts. Clients that support SEP-2127 SHOULD fetch this URL after resolving the manifest. See Section 1.4.
payment_required	boolean	Whether tool calls require payment. See Section 6.11.
payment_methods	array	Supported payment method identifiers. See Section 6.11.
tools_preview	array	Preview of available tools. See Section 6.12.1.
resources_preview	array	Preview of available resources. See Section 6.12.2.
prompts_preview	array	Preview of available prompt templates. See Section 6.12.3.

6.5. Auth Object

The "auth" object is defined in full in Section 6.10.4, which is the canonical definition. When used as a SHOULD field (Section 6.3), at minimum "required" and "methods" MUST be present.

Example (minimal, for a public server with optional OAuth 2.0):

```
   "auth": {
     "required": false,
     "methods": ["none", "oauth2"],
     "endpoint": "https://example.com/oauth/authorize",
     "metadata_url":
       "https://example.com/.well-known/oauth-authorization-server",
     "scopes": ["mcp:read"]
   }
```

See Section 6.10.4 for the complete field reference.

6.6. Transport Values

The "transport" field MUST contain one of the following values:

"http"	JSON-RPC 2.0 over HTTPS (request/response). This is the RECOMMENDED transport for publicly reachable servers.
"sse"	Server-Sent Events over HTTPS [W3C-SSE]. Suitable for servers that push notifications or stream results.
"stdio"	Standard input/output. This value is reserved for local process-based MCP servers and MUST NOT appear in manifests served via /.well-known/mcp-server, as such servers are by definition not reachable via a network endpoint. Clients SHOULD treat a manifest with transport "stdio" served over HTTPS as malformed.

Servers MAY declare multiple transports using a "transports" array field alongside the primary "transport" field.

6.7. Manifest Signature

Operators who require stronger integrity guarantees for their manifest MAY include a "signature" field containing a detached JWS (JSON Web Signature) as defined in [RFC7515], using a key published via DNS (analogous to DKIM [RFC6376]).

The signature covers the canonical JSON serialization of the manifest object excluding the "signature" field itself.

The "signature" object MUST contain:

alg	string	JWA algorithm identifier (e.g. "RS256", "ES256")
kid	string	Key ID corresponding to a JWK published in DNS
value	string	Base64url-encoded detached JWS signature

Example:

```
   "signature": {
     "alg": "ES256",
     "kid": "mcp-key-1",
     "value": "base64url..."
   }
```

The corresponding public key MUST be published as a DNS TXT record:

```
   _mcp-key.{domain}  IN  TXT  "v=mcp1jwk; kid=mcp-key-1; jwk={...}"
```

Clients that implement signature verification SHOULD reject manifests with invalid signatures. Clients that do not implement signature verification MAY ignore the "signature" field entirely.

NOTE: The signature mechanism defined here is informational and subject to revision in future versions of this document.

6.8. Endpoint Domain Validation

Clients MUST verify that the host component of the "endpoint" field in the manifest matches or is a subdomain of the host from which the manifest was retrieved. Manifests where the endpoint host does not match the retrieval host MUST be rejected.

Examples (manifest retrieved from example.com):

```
"endpoint": "https://example.com/mcp/"          valid
"endpoint": "https://api.example.com/mcp/"      valid
"endpoint": "https://other-domain.com/mcp/"     INVALID -- reject
```

This rule prevents manifest-based endpoint hijacking, where a compromised or fraudulent manifest redirects an MCP client to an unrelated server.

6.9. Expires Field

Servers SHOULD include an "expires" field containing an ISO 8601 timestamp indicating when the manifest becomes stale.

Clients MUST NOT use a cached manifest past its expiry time without re-fetching it. If the "expires" field is absent, clients SHOULD treat the manifest as stale after the period indicated by the HTTP Cache-Control header, or after 24 hours if no Cache-Control header is present.

The recommended validity period for a public manifest is between 1 hour and 90 days.

6.10. Security Capability Negotiation

6.10.1. Overview

MCP servers may operate under very different security regimes: a public sandbox tool, an enterprise API requiring OAuth 2.0, and a regulated healthcare server have fundamentally different client behaviour requirements. Without a pre-connection declaration, a client cannot know which posture applies until after the connection is established -- potentially after sensitive data has already been transmitted.

The manifest SHOULD declare the security posture of the server explicitly via the "trust_class" field (Section 6.10.2). This declaration allows clients to determine required authentication, logging, and compliance behaviour without performing a full protocol handshake. If "trust_class" is absent, clients MUST apply the defaults defined in Section 6.10.7.

The negotiation is unidirectional and declarative: the server publishes its requirements; the client reads and applies them. No additional round-trip is required.

6.10.2. The trust_class Field

The "trust_class" field is an OPTIONAL top-level field in the manifest. If absent, clients MUST behave as if "trust_class": "public" were declared. Server operators SHOULD declare "trust_class" explicitly to avoid ambiguity.

The field functions as a preset: each value implies a defined set of mandatory sub-fields and a minimum guaranteed behaviour that any

conforming client MAY assume without inspecting sub-fields individually.

Defined values:

"public" The server is accessible without restrictions. No authentication is required. The manifest MAY be cached freely up to the value of cache_ttl.

"sandbox" The server is experimental or non-production. Clients SHOULD warn users before invoking tools. Clients MUST NOT use sandbox servers in production workflows without explicit user confirmation. The "expires" sub-field is REQUIRED.

"enterprise" The server requires controlled access. Clients MUST NOT attempt connection before resolving the declared authentication method. The "auth" object is REQUIRED and MUST contain at least one entry in "methods".

"regulated" The server operates under a declared regulatory regime. Clients MUST apply the declared authentication method, MUST NOT cache the manifest beyond "cache_ttl", and MUST treat all sessions as subject to logging requirements declared in the "logging" object. The "auth", "compliance", and "logging" objects are all REQUIRED.

If a client encounters an unrecognised "trust_class" value, it MUST treat the manifest as "regulated" for safety purposes and SHOULD log a warning.

6.10.3. Mandatory Sub-Fields by trust_class

The following table defines which sub-fields are REQUIRED (R), OPTIONAL (O), or NOT APPLICABLE (N) for each trust_class value:

Sub-field	public	sandbox	enterprise	regulated
auth	O	O	R	R
expires	N	R	O	O
compliance	N	N	O	R
logging	N	N	O	R
cache_ttl	O	O	O	R

A manifest that declares a trust_class but omits a REQUIRED sub-field for that class is malformed. Clients MUST NOT connect to a server whose manifest is malformed.

6.10.4. The auth Object

When present, the "auth" object MUST contain:

"required" Boolean. If true, the client MUST complete authentication before sending any tool call.

"methods" Array of strings. Defines the authentication mechanisms supported by the server.

Core vocabulary (MUST be supported by conforming implementations):

"none" No authentication. Valid only when required is false.

"bearer" Bearer token. "endpoint" MUST be

present.
"mtls" Mutual TLS. Client certificate required.
"apikey" Static API key. Delivery mechanism MUST be declared in "apikey_header".
"oauth2" OAuth 2.0 flow. "endpoint" and "scopes" MUST be present.

Extension values MUST use the "x-" prefix (e.g., "x-saml", "x-mycompany-sso"). Clients that encounter an unrecognised "x-" value MUST ignore it and attempt connection using remaining known methods. If no known method remains, clients MUST NOT connect and SHOULD surface an error to the operator.

Values without the "x-" prefix that are not in the core vocabulary are invalid and MUST be treated as if absent.

"endpoint" URI. The authentication endpoint. REQUIRED when methods includes "bearer" or "oauth2".

"metadata_url" URI. RECOMMENDED when methods includes "oauth2". OAuth 2.0 Authorization Server Metadata endpoint as defined in [RFC8414]. Allows MCP clients to autoconfigure the full OAuth 2.0 flow without additional out-of-band information. MUST be an HTTPS URL.

"apikey_header" String. REQUIRED when methods includes "apikey". The HTTP header name used to deliver the API key (e.g. "X-Api-Key", "Authorization").

"scopes" Array of strings. REQUIRED when methods includes "oauth2". OAuth 2.0 scopes required by the server (e.g. ["mcp:read", "mcp:write"]).

6.10.5. The compliance Object

Required when trust_class is "regulated". MUST contain:

"jurisdiction" String. ISO 3166-1 alpha-2 country code, or one of the defined regional codes:
"EU" European Union (GDPR applies)
"EEA" European Economic Area
"UK" United Kingdom (UK GDPR applies)
Clients MUST NOT infer compliance requirements beyond what is explicitly declared.

"frameworks" Array of strings. Declared compliance frameworks. This field is informational -- clients MUST NOT refuse connection based on unrecognised framework identifiers. Examples: "GDPR", "HIPAA", "ISO27001", "PCI-DSS", "SOC2".

"certification_url" URI. OPTIONAL. A URL where compliance certification documentation can be retrieved. Clients MAY present this to users or automated auditors.

6.10.6. The logging Object

Required when trust_class is "regulated". MUST contain:

"required"	Boolean. If true, the client MUST NOT suppress session logging for this server.
"retention_days"	Integer. OPTIONAL. Minimum retention period for session logs in days. Clients that cannot honour this value SHOULD NOT connect and MUST surface a warning to the operator.

6.10.7. Default Values and Omitted Fields

Implementations MUST apply the following defaults when fields are absent:

trust_class	"public"
cache_ttl	3600 (seconds)
auth.required	false
logging.required	false

The principle is secure-by-default: a missing trust_class is never interpreted as elevated privilege. The most permissive safe default is applied, and the server operator is responsible for declaring any more restrictive posture explicitly.

6.10.8. Implementation Guidance for Non-Technical Operators

Conforming implementations (such as CMS plugins) SHOULD provide a configuration interface that maps user-facing choices to valid manifest fields without requiring the operator to author JSON directly. The following mapping is RECOMMENDED:

```
User declares site type "personal/blog"
  -> trust_class: "public"

User declares site type "business/commercial"
  -> trust_class: "enterprise"
  -> implementation SHOULD auto-detect existing auth systems

User declares site type "sensitive data" (health, finance, legal)
  -> trust_class: "regulated"
  -> implementation MUST prompt for jurisdiction before
     publishing manifest

User declares site type "development/testing"
  -> trust_class: "sandbox"
  -> implementation MUST set expires to no more than 90 days
```

Implementations MUST NOT publish a malformed manifest. If required sub-fields cannot be determined automatically, the implementation MUST block publication and prompt the operator for the missing information.

6.11. Payment Advertisement

Servers that require payment for tool calls MAY declare their payment capabilities in the manifest. This allows clients to determine payment compatibility before connecting.

Fields:

payment_required	boolean	Whether tool calls require payment
payment_methods	array	Supported payment methods

Valid payment method identifiers:

"x402"	HTTP 402 + USDC on-chain (Coinbase x402)
"mpp-tempo"	Machine Payment Protocol / Tempo

```
"stripe"      Stripe Connect
"apikey"      Pre-paid API key balance
```

Example:

```
   "payment_required": true,
   "payment_methods": ["x402", "stripe"]
```

NOTE: On-chain payment methods (e.g., x402) introduce per-transaction latency of ~2 seconds (Base L2) or more. Implementations SHOULD prefer session-level escrow patterns over per-call payments to avoid latency penalties on each tool invocation.

This field does not specify the payment protocol itself. Implementors SHOULD refer to the relevant payment protocol specifications for complete transaction semantics.

6.12. Primitive Previews

Servers MAY include previews of their tools, resources, and prompts directly in the manifest. This allows MCP clients and crawlers to understand server capabilities without establishing a connection, enabling semantic indexing, security pre-screening, and client-side filtering before the initialization handshake.

Three optional fields are defined:

```
tools_preview      array  Preview of available tools
resources_preview  array  Preview of available resources
prompts_preview    array  Preview of available prompt templates
```

Each field MUST be either:

(a) An array of preview objects as defined below, or
(b) The string "dynamic", indicating that the full list MUST be discovered through the MCP protocol's standard list operations and is subject to change.

6.12.1. Tool Preview Object

Each entry in "tools_preview" SHOULD contain:

```
name         string  REQUIRED. Tool identifier
description  string  SHOULD. Natural language description
inputSchema  object  MAY. JSON Schema for tool input parameters
```

Example:

```
   "tools_preview": [
     {
       "name": "search_products",
       "description": "Search products by type, material and size",
       "inputSchema": {
         "type": "object",
         "properties": {
           "query":    { "type": "string" },
           "category": { "type": "string" },
           "limit":    { "type": "integer" }
         },
         "required": ["query"]
       }
     },
     {
       "name": "check_stock",
       "description": "Check real-time warehouse availability"
```

```
    }
  ]
```

6.12.2. Resource Preview Object

Each entry in "resources_preview" SHOULD contain:

uri	string	REQUIRED. Resource URI or URI template
name	string	SHOULD. Human-readable name
description	string	MAY. Natural language description
mimeType	string	MAY. MIME type of the resource

Example:

```
"resources_preview": [
  {
    "uri": "catalog://products/{id}",
    "name": "Product Catalog",
    "description": "Full product details, specs and pricing",
    "mimeType": "application/json"
  }
]
```

6.12.3. Prompt Preview Object

Each entry in "prompts_preview" SHOULD contain:

name	string	REQUIRED. Prompt identifier
description	string	SHOULD. Natural language description
arguments	array	MAY. List of argument objects with name and description fields

Example:

```
"prompts_preview": [
  {
    "name": "compare_products",
    "description": "Generate a comparison between two products",
    "arguments": [
      { "name": "product_a", "description": "First product ID" },
      { "name": "product_b", "description": "Second product ID" }
    ]
  }
]
```

6.12.4. Consistency Requirement

Clients MUST treat primitive previews as advisory. The authoritative list of tools, resources, and prompts is always the one returned by the MCP protocol's list operations (tools/list, resources/list, prompts/list) after connection establishment.

Servers SHOULD ensure that the previews are consistent with the actual primitives returned during initialization. Servers MAY omit sensitive tool descriptions from the preview and mark them as "dynamic" if pre-connection disclosure is undesirable.

6.12.5. Relationship to SEP-1649 and SEP-2127

The primitive preview mechanism defined here is intentionally aligned with the MCP Server Cards proposal (SEP-1649) from the MCP maintainers at Anthropic. The key differences are:

- This document uses "tools_preview", "resources_preview", and "prompts_preview" field names to make the advisory nature explicit, while SEP-1649 uses "tools", "resources", "prompts".

- The well-known path in this document is "/.well-known/mcp-server" while SEP-1649 proposes "/.well-known/mcp/server-card.json". The authors welcome coordination with MCP maintainers to align these paths.

- This document additionally defines the "mcp" URI scheme and DNS TXT discovery (base mode and fast mode), which are not covered by SEP-1649.

- This document requests formal IANA registration of both the URI scheme and the well-known URI suffix, as noted as a future requirement in SEP-1649.

The "server_card" field defined in Section 6.4 provides a direct pointer from this document's manifest to a SEP-2127 Server Card, allowing the two specifications to coexist without requiring convergence on a single well-known path.

6.13. Minimal Example

```
{
  "mcp_version": "2025-06-18",
  "name": "Example MCP Server",
  "endpoint": "https://example.com/mcp",
  "transport": "http"
}
```

6.14. Full Example

```
{
  "mcp_version": "2025-06-18",
  "name": "Example Shop MCP Server",
  "description": "Product catalog and order management",
  "endpoint": "https://example.com/mcp",
  "transport": "http",
  "trust_class": "enterprise",
  "auth": {
    "required": true,
    "methods": ["oauth2"],
    "endpoint": "https://example.com/oauth/authorize",
    "metadata_url": "https://example.com/.well-known/as",
    "scopes": ["mcp:read", "mcp:write"]
  },
  "capabilities": ["tools", "resources"],
  "categories": ["e-commerce", "fashion"],
  "languages": ["en", "it"],
  "coverage": "IT",
  "contact": "api@example.com",
  "docs": "https://example.com/mcp/docs",
  "last_updated": "2026-03-25T00:00:00Z",
  "expires": "2026-09-25T00:00:00Z",
  "cache_ttl": 3600,
  "server_card":
    "https://example.com/.well-known/mcp/server-card.json",
  "payment_required": false,
  "crawl": true
}
```

6.15. HTTP Response Requirements

Servers MUST respond with:

```
HTTP/1.1 200 OK
Content-Type: application/json
```

Servers SHOULD include:

```
Cache-Control: max-age=3600
```

7. Security Considerations

7.1. Manifest Spoofing

A malicious actor could expose a fraudulent /.well-known/mcp-server document. Clients MUST verify that the domain in the manifest endpoint matches the domain of the original MCP URI.

When HTTPS is used (as REQUIRED by this specification), TLS certificate validation provides a baseline authenticity guarantee. Operators requiring stronger guarantees SHOULD use the manifest signature mechanism defined in Section 6.7.

7.2. DNS Hijacking

DNS TXT records may be spoofed on networks without DNSSEC. Operators SHOULD enable DNSSEC where available. When DNS TXT and .well-known endpoints conflict, the .well-known manifest MUST take precedence, as HTTPS provides domain authenticity guarantees that unsigned DNS records do not (see Section 4.3).

DNS-based manifest signatures (Section 6.7) inherit the same vulnerability. Operators using the signature mechanism SHOULD also enable DNSSEC for the signing key record.

7.3. Rate Limiting

Crawlers may generate excessive requests to /.well-known/mcp-server. Servers SHOULD implement standard HTTP rate limiting and return HTTP 429 (Too Many Requests) with appropriate Retry-After headers when limits are exceeded.

7.4. Sensitive Data

The Manifest is a public document. It MUST NOT contain credentials, tokens, API keys, or any private operational information.

7.5. Manifest Integrity via JWK Signature

The optional signature mechanism defined in Section 6.7 provides integrity verification for the manifest content. Implementers should be aware that:

- The signature covers content integrity but not freshness; a valid signature does not guarantee the manifest is current.
- Key rotation should be handled by publishing new key IDs in DNS and updating the manifest accordingly.
- Clients that cannot verify signatures MUST NOT treat unsigned manifests as less trustworthy than signed ones in the absence of a policy requiring signatures.

8. IANA Considerations

8.1. URI Scheme Registration

This document requests registration of the "mcp" URI scheme in the IANA Uniform Resource Identifier (URI) Schemes registry:

Scheme name:	mcp
Status:	Provisional
Applications:	AI agents, MCP clients
Contact:	marco.serra@mumble.group
Change controller:	IETF
Reference:	This document

8.2. Well-Known URI Registration

This document requests registration of the "mcp-server" well-known URI suffix in the IANA Well-Known URIs registry per [RFC8615]:

URI suffix:	mcp-server
Change controller:	IETF
Specification document:	This document
Related information:	None

9. Reference Implementation

A reference implementation of this specification is available at https://mcpstandard.dev, including a live /.well-known/mcp-server endpoint, a Python client library for resolving mcp:// URIs, CMS plugins for WordPress and Django, and an optional voluntary crawler (MCP Search Console).

Source code and community discussion are available at:

https://github.com/99rig/mcp-discovery

Registration with the Search Console is entirely optional. Conforming clients can discover any MCP server by following the resolution sequence defined in Section 4 without contacting any central registry or index.

10. References

10.1. Normative References

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

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005. <https://www.rfc-editor.org/rfc/rfc3986>

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008. <https://www.rfc-editor.org/rfc/rfc5234>

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, May 2015. <https://www.rfc-editor.org/rfc/rfc7515>

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, May 2015. <https://www.rfc-editor.org/rfc/rfc7517>

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

[RFC8259] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, December 2017.

<https://www.rfc-editor.org/rfc/rfc8259>

[RFC8414] Jones, M., Sakimura, N., and J. Bradley, "OAuth 2.0 Authorization Server Metadata", RFC 8414, June 2018. <https://www.rfc-editor.org/rfc/rfc8414>

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

[MCP-SPEC] Anthropic, "Model Context Protocol Specification 2025-06-18", <https://modelcontextprotocol.io/specification/2025-06-18>

10.2. Informative References

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987. <https://www.rfc-editor.org/rfc/rfc1035>

[RFC6376] Crocker, D., Hansen, T., and M. Kucherawy, "DomainKeys Identified Mail (DKIM) Signatures", STD 76, RFC 6376, September 2011. <https://www.rfc-editor.org/rfc/rfc6376>

[RFC7595] Thaler, D., "Guidelines and Registration Procedures for URI Schemes", RFC 7595, June 2015. <https://www.rfc-editor.org/rfc/rfc7595>

[W3C-SSE] Hickson, I., "Server-Sent Events", W3C Recommendation, February 2015. <https://www.w3.org/TR/eventsource/>

[MCPSTANDARD] Serra, M., "mcpstandard.dev -- MCP Discovery Reference Implementation", 2026. <https://mcpstandard.dev>

[GH-DISCUSSION] Serra, M., "MCP Server Discovery -- mcp:// URI scheme and /.well-known/mcp-server", GitHub Discussion #2462, 2026. <https://github.com/modelcontextprotocol/modelcontextprotocol/discussions/2462>

[ACTA] Farley, T., "Signed Decision Receipts for MCP Transport Layer", Internet-Draft draft-farley-acta-signed-receipts-00, 2026. <https://github.com/tomjwxf/ScopeBlindD2/blob/main/specs/draft-farley-acta-signed-receipts-00.md>

[SEP-1649] Anthropic, "MCP Server Cards: HTTP Server Discovery via .well-known", MCP Spec Enhancement Proposal 1649, 2025. <https://github.com/modelcontextprotocol/modelcontextprotocol/issues/1649>

[SEP-2127] Anthropic, "MCP Server Cards - HTTP Server Discovery via .well-known", MCP Spec Enhancement Proposal 2127, 2026. <https://github.com/modelcontextprotocol/modelcontextprotocol/pull/2127>

[MCP-WWW] kormco, "mcp-www: DNS-native MCP discovery client and benchmark", 2026.

<https://github.com/kormco/mcp-browse>

Author's Address

Marco Serra
Mumble Group
Milan, Italy
Email: marco.serra@mumble.group
URI: https://mcpstandard.dev