Tab 1

MCP Metadata RFC

Authors: Victor Morales (@electrocucaracha)
Date: June 19, 2025

As adoption of the Model Context Protocol (MCP) continues to grow, the need for a standardized mechanism to expose server metadata becomes increasingly important. This RFC proposes a convention for serving MCP server metadata through a well-known URI: */.well-known/mcp.json*.

The primary goal of this proposal is to improve the discoverability of MCP servers. By exposing a machine-readable metadata document, MCP servers can enable automated tools—such as registries, dashboards, and developer tooling—to efficiently discover, index, and interact with MCP instances consistently.

Providing a standard, well-known endpoint for this metadata also facilitates enterprise adoption by enabling clients to programmatically determine in advance the tools, resources, and prompt capabilities offered by each server, reducing the risk of prompt poisoning attacks. This predictable discovery process simplifies the external integration, supports compliance and governance workflows, and enhances interoperability across diverse MCP implementations.

This enhancement not only aligns with established web conventions (e.g., RFC 8615: Well-Known URIs) and reduces the operational burden on consumers by providing a standardized source of truth for server characteristics, including capabilities, supported features, and ownership information.

Metadata Scope

The metadata document is intended to expose the following key elements:

- Overview of the MCP server
- Features offered by the server
- Supported authentication mechanisms
- Supported transport protocols
- Tool identity and description
- Full JSON Schema definitions for accepted parameters and return values

Benefits

Introducing a well-known metadata file provides multiple benefits to the MCP ecosystem:

- Automated Discovery – Registries can automatically detect and catalog MCP servers.

- Simplified Integration – Consumers no longer need to rely on out-of-band documentation or manual configuration.
- Ecosystem Growth – Encourages interoperability.

Samples

Implementation Example

The following Python code demonstrates how to expose the metadata using the MCP Python SDK:

```
        @self._mcp.custom_route("/.well-known/mcp.json", methods=["GET"])
        async def well_known_mcp(_: Request) -> Response:
            if self._metadata is None:
                features = []
                for items, item_type in [
                    (await self._mcp.list_tools(), "tool"),
                    (await self._mcp.list_prompts(), "prompt"),
                    (await self._mcp.list_resources(), "resource"),
                ]:
                    for item in items:
                        t = vars(item)
                        t["type"] = item_type
                        features.append(t)

                repo = git.Repo(search_parent_directories=True)

                self._metadata = JSONResponse(
                    {
                        "name": "GitHub MCP Server",
                        "description": "MCP server that provides seamless
integration with GitHub APIs, enabling advanced automation and interaction
capabilities for developers and tools.",
                        "schemaVersion": "2025-06-18",
                        "transport": ["streamable-http", "stdio"],
                        "language": "python",
                        "authentication": [],
                        "git": {
                            "repository": repo.remote().url,
```

```
                    "commitSHA": repo.head.object.hexsha,
                },
                "features": features,
            }
        )

    return self._metadata
```

Example Output

An example `/.well-known/mcp.json` response for a GitHub MCP server:

```
{
  "name": "GitHub MCP Server",
  "description": " MCP server that provides seamless integration with
GitHub APIs, enabling advanced automation and interaction capabilities
for developers and tools.",
  "schemaVersion": "2025-06-18",
  "transport": ["streamable-http", "stdio"],
  "language": "go",
  "authentication": ["oauth2"],
  "git":
    {
      "repository": "https://github.com/github/github-mcp-server",
      "commitSHA": "87a477037fa1b48016635d4447523c65918ef5fe"
    },
  "features": [
    {
      "name": "get_issue",
      "title": null,
      "description": "Gets the contents of an issue within a
repository.",

      "inputSchema": {
        "properties": {
          "issue_number": {
            "title": "The number of the issue",
            "type": "number"
          },
          "owner": {
            "title": "The owner of the repository",
            "type": "string"
          },
          "repo": {
            "description": "The name of the repository",
            "type": "string"
          }
        },
        "required": [
          "owner",
          "repo",
          "issue_number"
        ]
      },
      "outputSchema": {
```

```
        "properties": {
          "result": {
            "title": "Result",
            "type": "string"
          }
        },
        "required": [
          "result"
        ],
      },
      "annotations": {
        "title": "Get issue details",
        "readOnlyHint": true
      },
      "meta": null,
      "type": "tool"
    },
    {
      "name": "get_me",
      "description": "Get details of the authenticated GitHub user. Use
this when a request includes ‘me’, ‘my’. The output will not change
unless the user changes their profile, so only call this once.",
      "type": "tool",
      "inputSchema": {
        "properties": {
          "reason": {
            "description": "The reason for requesting the user
information",
            "type": "string"
          }
        }
      },
      "outputType": "object",
      "annotations": {
        "title": "Get my user profile",
        "readOnlyHint": true
      }
    }
  ]
}
```


Schema Definition

Each metadata follows a structured schema that includes:

- `name:` Name of the MCP server
- `description:` Natural language summary of server's purpose.
- `schemaVersion:` The version of Schema used
- `transport:` List of supported transport protocols (e.g., stdio, sse, streamable-http)
- `language:` Programming language used in the implementation
- `authentication (optional):` Supported authentication mechanisms (e.g., oauth2)
- `git:` Source code information
    - `repository:` URL of the source repository
    - `commitSHA:` Specific commit SHA of the running instance
- `features:` A list of supported features
    - `name:` Canonical feature name
    - `description:` Natural language explanation.
    - `type:` Feature type (e. g., tool, prompt, resource)
    - `inputSchema (optional):` JSON Schema describing accepted inputs
    - `outputSchema (optional):` JSON Schema describing accepted outputs
    - `annotations (optional):` Extra metadata or hints.

Next steps

Once finalized, this metadata schema should be registered with IANA to reserve the */.well-known/mcp.json* path. This formal registration will standardize usage across implementations and promote consistent adoption.

Conclusion

This RFC proposes a standardized method for exposing MCP server metadata via a well-known URI. The metadata provides a structured, machine-readable description of server capabilities, authentication methods, transport protocols, and supported features.

By adopting this convention, the MCP ecosystem will benefit from:

- Easier tool integration
- Reduced implementation friction
- Stronger interoperability
- Increased visibility for MCP servers

We invite feedback and collaboration from the community to refine and adopt this proposal.

References

- GitHub discussions: https://github.com/orgs/modelcontextprotocol/discussions/84
- Well-Know URI Registration: https://github.com/protocol-registries/well-known-uris