Independent Submission
Internet-Draft
Intended status: Experimental
Expires: 5 November 2026

I. Abbott
SoftOboros

4 May 2026

MCP Aggregation Protocol (MCP-AX): Hierarchical Tool Namespace Delegation for Model Context Protocol Servers
draft-abbott-mcp-ax-00

Abstract

This document specifies MCP-AX, an aggregation protocol for Model Context Protocol (MCP) servers. MCP-AX enables hierarchical composition of tool namespaces across heterogeneous networks of MCP servers, from cloud services to resource-constrained embedded devices. The protocol defines a master-subagent architecture directly inspired by the AgentX protocol (RFC 2741), adapted for the tool/resource model of MCP rather than the OID/MIB model of SNMP.

MCP-AX introduces recursive namespace delegation, capability-aware routing, transport bridging for constrained nodes, and irreversibility-gated tool dispatch suitable for autonomous and semi-autonomous AI agent systems.

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 5 November 2026.

Copyright Notice

Table of Contents

1. Introduction

The Model Context Protocol (MCP) [MCP] defines a client-server architecture in which an AI model (the client) discovers and invokes tools, reads resources, and receives notifications from MCP servers. As deployment scales from single-server configurations to enterprise, IoT, and hybrid cloud/edge topologies, the need arises for hierarchical aggregation of MCP servers behind a unified namespace.

This problem is not new. SNMP encountered identical scaling constraints in the 1990s and produced AgentX [RFC2741], which defined a master agent / subagent architecture for delegating OID subtrees. The structural correspondence is direct:

AgentX Concept	MCP-AX Concept
Master Agent	Root Aggregator
Subagent	Subserver
OID Subtree	Tool Namespace Prefix
MIB Registration	Tool Registration
ax.Register PDU	mcpax/register
ax.Get/GetNext	tools/call (routed)
MIB Walk	tools/list (recursive)
ax.Notify PDU	MCP notification (prefixed)
ax.Ping PDU	mcpax/heartbeat
Session ID	session_id

Table 1

This document specifies the MCP-AX protocol, adapting the AgentX model for MCP's tool/resource/notification primitives.

1.1. Design Goals

(a) A model client MUST be able to interact with an MCP-AX aggregator using unmodified MCP protocol. Aggregation is transparent to the client.

(b) Namespace delegation MUST be recursive: an aggregator MAY register as a subserver of a parent aggregator, forming an arbitrarily deep hierarchy.

(c) Resource-constrained devices, including those with RAM budgets below 4 KiB, MUST be supportable through a gateway aggregator by means of a bounded stub profile. The protocol MUST NOT assume that participating nodes can parse JSON, manage sessions, perform authentication, negotiate schemas, or store fully qualified tool names. These are gateway responsibilities, not leaf requirements. The stub profile defines the minimum contract between a constrained device and its gateway; the gateway maps that contract into MCP-AX on behalf of the device.

(d) Tool dispatch MUST carry capability metadata sufficient for the client to reason about latency, mutability, and reversibility without knowledge of the routing topology.

(e) Security context MUST NOT bleed across aggregation boundaries. Each hop authenticates independently.

1.2. Distinction from API Gateways

HTTP reverse proxies and API gateways (nginx, Envoy, Kong) route requests by URI path or header. MCP-AX differs in four respects that are not achievable through gateway configuration alone:

(a) Recursive namespace delegation: aggregation depth is unbounded and each hop is itself an MCP-AX participant, not a transparent proxy.

(b) Capability propagation: tool metadata (latency class, mutability, reversibility, consistency) is a first-class protocol element, not an out-of-band annotation.

(c) Safety semantics: irreversibility gates and agent budget enforcement operate on tool-call semantics, not HTTP verbs.

(d) Embedded transport bridging: sub-TCP transports (UART, BLE, CAN, SPI) with CBOR encoding are protocol-native, not bolted on via sidecar.

1.3. Relationship to AgentX

MCP-AX is a spiritual successor to AgentX [RFC2741], not a profile or extension of it. The wire protocol is MCP JSON-RPC, not AgentX PDUs. However, the registration semantics, namespace delegation model, and master/subagent lifecycle are directly derived from AgentX, which is cited as a normative architectural reference.

The term "agent" -- used independently in SNMP network management (1988), AgentX subagent delegation (1999), and contemporary AI agent systems (2024) -- names the same abstraction at three different layers of the stack. MCP-AX makes this recursion explicit.

1.4. MCP Governance Note

MCP originated at Anthropic in November 2024. In December 2025, Anthropic, Block, and OpenAI contributed MCP and the Agent-to-Agent (A2A) protocol to the Agentic AI Foundation (AAIF), a Linux Foundation project. MCP-AX treats MCP as a stable external specification and cites it accordingly. Should MCP be submitted as an IETF RFC in future, the normative reference herein should be updated.

Several IETF Internet-Drafts have already referenced MCP as an external specification for network management use cases [I-D.zeng-mcp-network-mgmt] [I-D.zeng-mcp-network-measurement] [I-D.zeng-mcp-troubleshooting]. MCP-AX follows this precedent.

2. Terminology

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.

Aggregator: An MCP server that exposes a unified tool namespace composed from one or more downstream MCP servers (subservers). Analogous to an AgentX master agent.

Subserver: An MCP server that registers its tools with an upstream aggregator. Analogous to an AgentX subagent.

Leaf Server: A subserver that does not itself aggregate other servers. Terminal node in the hierarchy.

Gateway: An aggregator that additionally performs transport bridging (e.g., UART/CBOR to HTTP/JSON) for constrained subservers.

Namespace Prefix: A dot-delimited string prepended to tool names during registration to establish hierarchical ownership. Analogous to an OID subtree in AgentX.

Tool Route: The ordered list of aggregator segments from root to the leaf server that owns a tool.

Capability Annotation: Metadata attached to a tool registration describing latency class, mutability, reversibility, and transport characteristics.

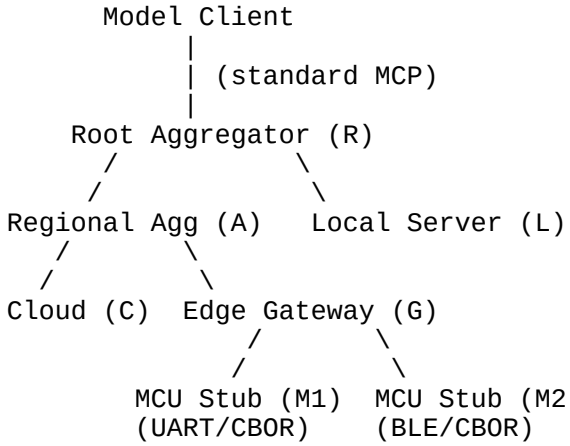
Stub: A minimal MCP implementation on a constrained device,

supporting only tools/list and tools/call over a compact transport encoding.

3. Architecture Overview

3.1. Topology

MCP-AX defines a tree topology rooted at one or more root aggregators. The model client connects to the root aggregator and sees a flat, fully-qualified tool namespace.



Each non-leaf node is an aggregator. Each aggregator accepts registrations from downstream subservers, prepends its namespace prefix to registered tool names, exposes the merged namespace upstream, and routes tools/call requests to the owning subserver.

3.2. Transparency

The model client issues standard MCP requests. It receives tools/list responses containing fully qualified (prefixed) tool names and capability annotations. It issues tools/call with the fully qualified name. The aggregation hierarchy is opaque to the client except as revealed by namespace structure and capability metadata.

4. Namespace Model

4.1. Namespace Syntax

Tool names in MCP-AX use dot-delimited hierarchical segments:

<root-segment>.<aggregator-segment>*.<local-name>

Example: infra.edge.stm32h7.dma2d_status

Segments MUST match the regular expression [a-z0-9_-]{1,63}. The total fully qualified name MUST NOT exceed 255 characters.

4.2. Registration and Ownership

When a subserver registers with an aggregator, it provides its local tool list (via tools/list) and a requested namespace segment. The aggregator validates the segment for uniqueness among current registrations. First-registered-wins semantics apply, consistent with AgentX [RFC2741] Section 7.1.5.1. A conflict MUST result in rejection with error "namespace_conflict".

A subserver MAY include an "authority" field in its registration request, using DNS name syntax (e.g., "dns:edge01.factory.example.com"). When present, the (authority, segment) pair constitutes the ownership claim. An aggregator MAY reject re-registration of a segment by a different authority, even after the original session has expired. This provides stable namespace ownership across session churn, failover, and multi-root federation without requiring a global registry.

An authority claim is an assertion, not proof. Without verification, a rogue subserver connecting after a reboot can claim an authority string it does not own. For namespaces containing mutable or irreversible-mutable tools, aggregators SHOULD require cryptographic proof of authority: a signed JWT bearing the authority DNS name as subject, a client certificate whose SAN matches the claimed authority, or equivalent. Aggregators MAY accept unverified authority claims for read-only or development namespaces where the risk of impersonation is acceptably low.

4.3. Namespace Isolation

A subserver MUST NOT register tools with names containing the dot separator. Hierarchical depth is achieved only through recursive aggregation, not through subserver self-prefixing. This prevents namespace spoofing: the aggregator is the sole authority for prefix assignment.

5. Aggregator Behavior

5.1. tools/list Handling

Upon receiving tools/list from upstream (or from the model client), the aggregator returns its cached merged namespace if valid. Otherwise, it issues tools/list to each registered subserver, prefixes results, merges, caches (RECOMMENDED TTL: 60 seconds), and returns.

5.2. tools/call Routing

Upon receiving tools/call for a fully qualified tool name, the aggregator looks up the tool in its routing table, advances the route cursor (Section 7.2), forwards the request to the downstream session, and returns the response. If the tool name is not found, the aggregator MUST return error code -32601 (method not found).

5.3. Recursive Aggregation

An aggregator MAY itself register as a subserver of a parent aggregator, presenting its merged namespace for further prefixing. There is no protocol-imposed depth limit; implementations SHOULD support at least 8 levels.

Implementations MUST detect and reject registration cycles. When an aggregator registers as a subserver of a parent, it MUST include its own aggregator_id and the aggregator_ids of all its downstream subservers in the registration request's "x-mcpax-subtree-ids" field. The parent MUST reject the registration if its own aggregator_id appears in that set. This provides cycle detection at registration time with no runtime overhead on tool dispatch.

6. Registration Protocol

6.1. Registration Request

```
{
  "jsonrpc": "2.0",
  "method": "mcpax/register",
  "id": 1,
  "params": {
    "subserver_id": "550e8400-e29b-41d4-a716-446655440000",
    "segment": "stm32h7",
    "authority": "dns:edge01.factory.example.com",
    "capabilities": {
      "tools": true,
      "resources": false,
      "notifications": true
    },
    "heartbeat_interval_ms": 5000,
    "transport_class": "uart_cbor",
    "version": "2026-05-01"
  }
}
```

The "subserver_id" field is a stable UUID that persists across sessions and reboots. The "authority" field is OPTIONAL; see Section 4.2. Together with the aggregator's own identity, these fields enable deterministic split-brain detection (Section 13.4).

6.2. Registration Response

```
{
  "jsonrpc": "2.0",
  "id": 1,
  "result": {
    "status": "registered",
    "assigned_segment": "stm32h7",
    "session_id": "a3f7c...",
    "heartbeat_deadline_ms": 15000,
    "budget": {
      "max_calls_per_minute": 60,
      "max_mutable_calls_per_session": 10
    }
  }
}
```

6.3. Heartbeat and Deregistration

If heartbeat_interval_ms is non-zero, the subserver MUST send "mcpax/heartbeat" within each interval. Three consecutive missed heartbeats trigger automatic deregistration. A subserver may also send "mcpax/deregister" explicitly. Upon deregistration, the aggregator MUST remove all tools from that subserver's namespace within one heartbeat interval and re-advertise upstream.

These semantics mirror AgentX session timeout (Section 7.1.1 of [RFC2741]).

7. Tool Dispatch and Routing

7.1. Routing Table

```
{
  "infra.edge.stm32h7.dma2d_status": {
    "downstream_session": "a3f7c...",
    "downstream_name":    "dma2d_status",
    "capability":         { ... },
    "registered_at":      "2026-05-01T12:00:00Z"
  }
}
```

7.2. Request and Response Transformation

Routing uses a cursor model rather than string manipulation. Each tools/call request carries an "x-mcpax-route" array containing the full sequence of namespace segments from root to leaf, and an "x-mcpax-cursor" integer indicating the current position in that array. Each aggregator increments the cursor by one and forwards the request downstream. The downstream server uses the segment at the cursor position to identify itself and dispatches to the tool named by the final segment.

Example for tool "infra.edge.stm32h7.dma2d_status":

"x-mcpax-route": ["infra", "edge", "stm32h7", "dma2d_status"],
"x-mcpax-cursor": 0

Root aggregator matches "infra" at cursor 0, increments to 1, forwards. Regional aggregator matches "edge" at cursor 1, increments to 2, forwards. Gateway matches "stm32h7" at cursor 2, increments to 3, dispatches tool "dma2d_status".

This avoids string parsing errors, supports non-string namespace encodings in future extensions (integer IDs, hashes), and preserves the full route for diagnostics at every hop.

The response is forwarded upstream without modification to the result field. The aggregator MAY add "x-mcpax-latency-ms" to response metadata.

7.3. Timeout Propagation

The aggregator SHOULD set downstream request timeouts based on the capability annotation's latency_class:

latency_class	Timeout
realtime	500 ms
fast	5 s
standard	30 s
slow	120 s
batch	caller-managed

Table 2

8. Transport Bridging

8.1. Gateway Role

A gateway translates between MCP's native transport (HTTP+SSE or stdio) and constrained transports used by embedded stubs. Defined bridged transport classes:

* uart_cbor: UART with COBS framing, CBOR payload
* ble_cbor: BLE GATT characteristics, CBOR payload
* spi_cbor: SPI with length-prefix framing, CBOR payload
* can_cbor: CAN bus with ISO-TP segmentation, CBOR payload
* mqtt_json: MQTT topics, JSON payload

8.2. CBOR-MCP Mapping

The gateway maintains a bidirectional mapping between MCP JSON-RPC and a compact CBOR [RFC7049] representation. Tool names are replaced with integer IDs assigned at registration time. JSON-RPC envelope fields "jsonrpc", "method", "id" map to CBOR map keys 0, 1, 2 respectively. Schema validation occurs at the gateway, not on the stub.

The gateway MUST strip all "x-mcpax-*" metadata (route array, cursor, hop count, latency annotations) before translating a request to the stub transport. A constrained stub receives only its integer tool ID and the CBOR-encoded arguments. Routing metadata is a gateway concern; it MUST NOT leak onto constrained transports where every byte has a cost.

The combination of integer tool IDs, typed argument schemas, and gateway-side validation constitutes a minimal interface description for constrained environments -- functionally equivalent to the subset of IDL semantics required for RPC dispatch, without requiring a separate schema language or code generator on the stub.

8.3. Stub Requirements

An MCP-AX stub MUST NOT be required to parse JSON, implement HTTP or SSE, manage sessions, perform authentication, or store fully qualified tool names. These responsibilities are fully delegated to the gateway. Two stub profiles are defined:

Profile A -- Table Stub: No self-description capability. The gateway owns all tool schemas and maps integer command IDs to MCP-AX tool names from its own configuration. The device exposes numeric command IDs only. Suitable for devices with no dynamic discovery requirement, including legacy 8-bit microcontrollers.

Profile B -- Self-Describing Stub: Announces tool IDs and compact schemas at boot or on gateway request via CBOR registration frames. Suitable for Cortex-M0+, AVR, MSP430, and equivalent 16/32-bit devices.

Profile A target resource budget: effectively zero dynamic RAM for protocol state; the command table is ROM-resident and the gateway handles everything else. Profile B target: fewer than 2 KB flash code, fewer than 256 bytes RAM for request/response buffers. See Appendix "Appendix B: Embedded Stub Reference Profiles" for reference descriptor structures for both profiles.

9. Capability Metadata

9.1. Capability Annotation Schema

```
{
  "latency_class":  "realtime|fast|standard|slow|batch",
  "consistency":    "strong|eventual|best_effort",
  "mutable":        boolean,
  "reversible":     boolean,
  "idempotent":     boolean,
  "transport":      "native|uart_cbor|ble_cbor|...",
  "auth_scope":     "read|write|admin",
  "cost_class":     "free|metered|expensive",
  "availability":   "always|scheduled|best_effort|degraded",
  "schema_version": "<semver>"
}
```

The "consistency" field indicates the data consistency model of the tool's backing state. Agents SHOULD use this to determine whether parallel invocations across tools in the same subtree may observe stale state, and whether retry after failure requires read-before-write verification.

The "availability" value "degraded" indicates that the tool remains listed but may return structured error responses for some or all invocations. This supports partial-failure semantics required by industrial control and observability pipelines where binary present/absent is insufficient. See Section 13.2.

9.2. Aggregator Annotation Obligations

The aggregator MUST propagate capability annotations upstream without modification. It MUST add "x-mcpax-hops" indicating the aggregation depth. It MUST adjust "latency_class" upward if the aggregation path introduces latency that changes the effective class; it MUST NOT adjust latency_class downward.

10. Security Considerations

10.1. Per-Hop Authentication

Each aggregation boundary is an authentication boundary. Credentials MUST NOT be forwarded across aggregation hops. This is the critical lesson from SNMP's community string model [RFC3414]: transitive credential forwarding violates least privilege. MCP-AX mandates independent authentication at each hop, with tokens scoped to the immediate downstream session per [RFC8707].

10.2. Scope Restriction

An aggregator MUST scope the authorization context per registered subserver. A subserver registered with auth_scope "read" MUST NOT receive requests that imply "write" or "admin" operations.

10.3. Namespace Spoofing Prevention

The prohibition on subserver self-prefixing (Section 4.3) prevents a malicious subserver from registering tools that appear to belong to a different subtree. The aggregator is the sole authority for prefix assignment.

10.4. Transport Security

Aggregator-to-aggregator links MUST use TLS 1.3 or equivalent. Gateway-to-stub links (UART, SPI, BLE) operate in physically constrained environments where TLS may be infeasible. Gateways MUST treat stub-provided data as untrusted and validate all inputs against the registered schema before forwarding upstream.

11. Irreversibility and Safety Gates

11.1. Motivation

When the model client is an autonomous AI agent, tool invocations may have real-world consequences that are difficult or impossible to reverse. The aggregation hierarchy is the natural enforcement point for safety policy: the aggregator has visibility into the blast radius of downstream operations that individual leaf servers lack.

11.2. Irreversibility Classification

Tools with "reversible": false and "mutable": true are classified as irreversible-mutable and MUST be flagged with "x-mcpax-safety": "irreversible_mutable" in the namespace.

11.3. Gate Enforcement

An aggregator operating in "gated" mode MUST intercept tools/call requests targeting irreversible-mutable tools, return a "confirmation_required" response to the client (including tool name, arguments, capability annotation, and route), and await "mcpax/confirm" before dispatching downstream. Unconfirmed requests expire after a configurable timeout (default: 300 seconds).

The "mcpax/confirm" request MUST include a "proof" field containing a cryptographic token obtained from an authority external to the requesting model client. Acceptable proof types include: a signature from a human operator's key pair, a signed nonce from an external policy engine, or an approval token from an out-of-band authorization service. The aggregator MUST validate the proof against a configured trust anchor before dispatching the gated request.

Without this requirement, an autonomous model client can trivially self-confirm by issuing "mcpax/confirm" immediately after receiving "confirmation_required", rendering the gate ineffective. The proof field ensures that confirmation requires a cryptographic assertion that the model client cannot manufacture from its own context.

Safety Gate Monotonicity: once a request is classified as requiring confirmation at any hop in the aggregation hierarchy, all upstream hops MUST preserve that requirement. No aggregator or client MAY remove or downgrade a gate introduced by a downstream aggregator. Any aggregator MAY escalate an ungated request to gated; none MAY de-escalate. This ensures that the most conservative policy in the path governs.

11.4. Agent Budget Propagation

Budget enforcement (max_calls_per_minute, max_mutable_calls_per_session, max_cost_units_per_session) is per-session, declared in the registration response, and non-negotiable. A subserver exceeding its budget MUST receive error -32003 ("budget_exceeded").

12. Notification Aggregation

Subserver notifications are prefixed with the originating namespace segment and forwarded upstream. The aggregator MUST NOT reorder notifications from a single subserver.

The aggregator MUST implement per-subserver rate limiting (default: 100/second). If exceeded, the aggregator buffers up to a configurable depth (default: 1000), then drops oldest notifications, increments a "notifications_dropped" counter (exposed as a tool on the aggregator), and emits a synthetic "notification_overflow" notification upstream. This directly addresses the SNMP trap storm problem [RFC3413].

Notifications MAY include "x-mcpax-event-id" (UUID) and "x-mcpax-causal-parent" (UUID or null) fields. These do not impose cross-subserver ordering but enable downstream consumers to reconstruct causal chains when notifications from multiple subservers relate to

the same triggering event. Aggregators MUST forward these fields without modification if present. Aggregators MUST NOT hold causal parent notifications in state or attempt to correlate them; they are opaque pass-through values. Causal reconstruction is a consumer concern, not an aggregator concern.

13. Failure Modes and Recovery

13.1. Subserver Failure

On heartbeat timeout or connection loss, the aggregator emits "subserver_lost" upstream. On reconnection, tools are re-added. The aggregator MUST NOT cache tool definitions across sessions.

13.2. Degraded Mode

Rather than immediately removing all tools from a failed subserver's namespace, an aggregator MAY transition those tools to "availability": "degraded". In degraded mode, tools remain listed in tools/list responses but tools/call requests return a structured error:

```
{
  "code": -32002,
  "message": "tool_degraded",
  "data": {
    "reason": "subserver_unreachable",
    "since": "2026-05-01T12:05:00Z",
    "retry_after_ms": 5000
  }
}
```

The aggregator MUST remove degraded tools entirely after a configurable grace period (RECOMMENDED: 5 minutes) if the subserver does not recover. Degraded-mode semantics are essential for industrial control and observability pipelines where abrupt namespace changes can trigger cascading failures in upstream consumers.

13.3. Aggregator Failure

The parent detects heartbeat loss and removes the entire subtree. Subservers MAY reconnect to a configured backup aggregator. Convergence time after failure MUST be less than three heartbeat intervals across the affected subtree.

13.4. Split-Brain Prevention

Each subserver carries a stable "subserver_id" (UUID) in its registration request. Each aggregator maintains its own "aggregator_id" (UUID). An aggregator MUST reject a registration from a subserver_id that is already registered with a different aggregator_id in the same tree.

For cloud deployments, detection SHOULD use a shared registration store (e.g., DynamoDB, etcd) keyed by subserver_id. For edge deployments where shared state is unavailable, the aggregator SHOULD query the subserver for its current parent_id and reject if it differs. This provides deterministic behavior without mandating a specific distributed consensus system.

14. IANA Considerations

This document requests the following registrations:

(a) MCP-AX method namespace "mcpax/*" in the MCP method registry (to be established by the Agentic AI Foundation or a future IETF working group).

(b) MCP-AX capability annotation keys "x-mcpax-*" in the MCP metadata registry (to be established).

(c) MCP-AX transport class identifiers in a new "MCP-AX Transport Classes" registry.

15. References

15.1. Normative References

[MCP] Agentic AI Foundation / Linux Foundation, "Model Context Protocol Specification", Originally authored by D. Soria Parra and J. Spahr-Summers, Anthropic. Governance transferred to the Agentic AI Foundation, a Linux Foundation project, December 2025., November 2025, <https://modelcontextprotocol.io/specification/>.

[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/rfc/rfc2119>.

[RFC2741] Daniele, M., Wijnen, B., Ellison, M., and D. Francisco, "Agent Extensibility (AgentX) Protocol Version 1", RFC 2741, DOI 10.17487/RFC2741, January 2000, <https://www.rfc-editor.org/rfc/rfc2741>.

[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/rfc/rfc8174>.

[RFC8707] Campbell, B. and J. Bradley, "Resource Indicators for OAuth 2.0", RFC 8707, DOI 10.17487/RFC8707, February 2020, <https://www.rfc-editor.org/rfc/rfc8707>.

15.2. Informative References

[I-D.zeng-mcp-network-measurement] Zeng, X., "MCP-based Network Measurement Framework", Work in Progress, Internet-Draft, draft-zeng-mcp-network-measurement-00, October 2025, <https://datatracker.ietf.org/doc/html/draft-zeng-mcp-network-measurement-00>.

[I-D.zeng-mcp-network-mgmt] Zeng, X., "Model Context Protocol (MCP) Extensions for Network Equipment Management", Work in Progress, Internet-Draft, draft-zeng-mcp-network-mgmt-01, October 2025, <https://www.ietf.org/archive/id/draft-zeng-mcp-network-mgmt-01.html>.

[I-D.zeng-mcp-troubleshooting] Zeng, X., "Using MCP for Intent-Based Network Troubleshooting Automation", Work in Progress, Internet-Draft, draft-zeng-mcp-troubleshooting-00, October 2025, <https://www.ietf.org/archive/id/draft-zeng-mcp-troubleshooting-00.html>.

[RFC3413] Levi, D., Meyer, P., and B. Stewart, "Simple Network Management Protocol (SNMP) Applications", STD 62, RFC 3413, DOI 10.17487/RFC3413, December 2002, <https://www.rfc-editor.org/rfc/rfc3413>.

[RFC3414] Blumenthal, U. and B. Wijnen, "User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)", STD 62, RFC 3414, DOI 10.17487/RFC3414, December 2002, <https://www.rfc-editor.org/rfc/rfc3414>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/rfc/rfc7049>.

Appendix A: AgentX Full Correspondence Table

AgentX Concept	MCP-AX Concept
Master Agent	Root Aggregator
Subagent	Subserver
OID Subtree	Namespace Prefix
MIB Registration	Tool Registration
ax.Register PDU	mcpax/register
ax.Unregister PDU	mcpax/deregister
ax.Get/GetNext/GetBulk	tools/call (routed)
ax.Notify PDU	MCP notification (prefixed)
ax.Ping PDU	mcpax/heartbeat
ax.Response PDU	JSON-RPC response
Session ID	session_id
Context	Namespace segment
sysUpTime	registered_at timestamp
AgentX socket (unix/tcp)	HTTP+SSE / stdio / bridged

Table 3

Key divergences from AgentX:

(a) Wire format is JSON-RPC (or CBOR for bridged transports), not binary PDUs.

(b) Namespaces are locally unique by first-registration, not globally by IANA OID assignment.

(c) Capability annotations (latency, mutability, reversibility) have no AgentX equivalent.

(d) Irreversibility gates and agent budget enforcement are MCP-AX additions with no SNMP analog.

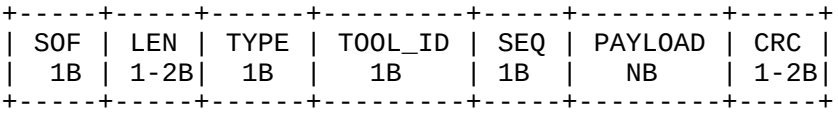
Appendix B: Embedded Stub Reference Profiles

B.1 Profile A -- Table Stub

Profile A targets devices with no self-description capability, including legacy 8-bit microcontrollers (8051-class, 6800-class). The gateway owns all tool schemas and name mappings. The device exposes only numeric command IDs over a minimal frame protocol.

Resource budget: effectively zero dynamic RAM for protocol state. The command dispatch table is ROM-resident. The gateway configuration file defines the mapping from command IDs to MCP-AX tool names, argument schemas, and capability annotations.

Reference wire frame (gateway-to-stub and stub-to-gateway):




```
TYPE:  0x01 = call request
       0x02 = call response
       0x03 = registration (Profile B only)
       0x04 = heartbeat

PAYLOAD: fixed structs, TLV, or raw bytes.
         Interpretation is defined in gateway config.
```

Profile A devices need not use CBOR. The gateway translates between the device's native byte layout and MCP-AX JSON-RPC. The stub's only contract is: receive (TOOL_ID, PAYLOAD), execute, return (SEQ, PAYLOAD, status byte).

```
<CODE BEGINS>
/* Profile A: ROM command table -- no CBOR, no schemas on device */
typedef struct {
    uint8_t   cmd_id;
    uint8_t   req_len;       /* fixed request payload length  */
    uint8_t   resp_len;      /* fixed response payload length */
    int (*handler)(const uint8_t *req, uint8_t *resp);
} mcpax_cmd_t;

static const mcpax_cmd_t commands[] = {
    { 0x01, 0,  4, read_sensor_handler  },
    { 0x02, 1,  0, set_led_handler      },
};
<CODE ENDS>
```

B.2 Profile B -- Self-Describing Stub

Profile B targets 16/32-bit microcontrollers (Cortex-M0+, AVR, MSP430) that can announce tool IDs and compact argument schemas at boot via CBOR registration frames.

Resource budget: fewer than 2 KB flash code, fewer than 512 bytes ROM for schema tables, fewer than 256 bytes RAM for request/response buffers. Transport: UART at 9600 baud or higher, COBS framing. Encoding: CBOR [RFC7049], map-only payloads.

```
<CODE BEGINS>
/* Profile B: self-describing tool table with CBOR type hints */
typedef struct {
    uint8_t   tool_id;
    uint8_t   arg_count;
    uint8_t   arg_types[4];   /* CBOR major types */
    uint8_t   ret_type;
    int (*handler)(const uint8_t *args, uint8_t *resp,
                   uint16_t *resp_len);
} mcpax_tool_t;

static const mcpax_tool_t tools[] = {
    { 0x01, 0, {}, CBOR_MAP, dma2d_status_handler },
    { 0x02, 1, {CBOR_UINT}, CBOR_MAP, set_led_handler },
};
<CODE ENDS>
```

B.3 Boot Sequence

Profile A:

1. Stub powers on; gateway detects presence (UART RTS/CTS, GPIO, or poll).
2. Gateway loads tool mappings and schemas from its own configuration.
3. Gateway sends mcpax/register upstream on behalf of stub.
4. Gateway begins heartbeat on behalf of stub.
5. Stub enters idle state, awaiting framed command bytes.

Profile B:

1. Stub powers on; sends CBOR registration frame listing tool_ids and schemas.
2. Gateway maps tool_ids to fully qualified names from its configuration.
3. Gateway sends mcpax/register upstream.
4. Gateway begins heartbeat on behalf of stub.
5. Stub enters idle state, awaiting CBOR tool call frames.

B.4 Gateway Responsibilities

For both profiles, the gateway handles all protocol complexity:

* CBOR/byte-frame to JSON-RPC bidirectional translation
* Tool ID to fully qualified name mapping
* Schema validation of arguments before forwarding to stub
* Stripping of all x-mcpax-* routing metadata at the stub boundary
* Authentication context for upstream aggregator
* Heartbeat generation on behalf of stub
* Buffering and retry for unreliable transports (BLE, CAN)

Appendix C: Example Topologies

C.1 Enterprise SaaS Aggregation

```
              Model Client
                   |
              Root Aggregator
               /       |        \
              Google  Jira    Salesforce
              Drive  Server    Server

              Namespace: google_drive.*, jira.*, salesforce.*
```

C.2 Industrial IoT with Edge Gateway

```
                  Model Client
                       |
                  Cloud Aggregator (AWS)
                       |
                  Edge Aggregator (Jetson)
                   /         |          \
                  PLC     Sensor    Actuator
                  (Modbus  Array    Controller
                  gateway) (BLE)    (UART/CBOR)

                  Irreversible-mutable actuator tools:
                  confirmation gate at cloud aggregator.
```

C.3 Recursive Regional Hierarchy

```
             Model Client
                  |
             Global Aggregator
              /              \
             US Regional    EU Regional
              /      \        /      \
             US-E   US-W   EU-W    EU-E

             Namespace: global.us.east.*, global.us.west.*,
                        global.eu.west.*, global.eu.east.*

             Model infers geographic locality from namespace.
```

Acknowledgments

The authors gratefully acknowledge the designers of AgentX [RFC2741], whose architectural insights proved remarkably durable across a quarter-century gap and an entirely different application domain. The term "agent" has, it turns out, aged exceptionally well.

The irreversibility gate mechanism draws on safety-critical HMI design principles and the concept of useful friction at irreversibility boundaries in autonomous agent planning systems.

Author’s Address

Ira Abbott
SoftOboros
Ottawa Ontario
Canada
Email: ira@softoboros.com