Configure Workflows as MCP Tools

Last Modified on 09/01/2026 2:57 am EDT

V11.4

1: Workflow as MCP tool overview

Agents in the Agentic Fabric don't call Orchestration AI workflows directly. They call **MCP tools**, and Orchestration AI turns a subset of its existing **HTTP Listeners** into those tools automatically, no new tool-specific code per workflow.

The chain, end to end:

```
Agent (e.g. COE Summarization Agent)
  → MCP call to McpServer (agentic-mcpserver, POST /mcp)
    → HTTPS + OBO to Sequence's mcp/v1/invoke
      → DynamicToolInvoker calls your HTTP Listener endpoint
        → your HTTP Listener triggers the workflow
```

The following two components do the work:

- **`DynamicHttpListenerToolProvider`** : scans every HTTP Listener definition, parses its OpenAPI document (YAML or JSON), and picks out the operations you've opted in.
- **`HttpListenerToolMapper`** : converts each opted-in operation into a tool: a name, a description, and a JSON Schema built from the operation's parameters and request body.

> **NOTE**
> Your job as the workflow developer is entirely in the OpenAPI document attached to your HTTP Listener. You don't write C#, you don't register anything separately, you annotate operations with a handful of vendor extension fields.

2: Prerequisite

The workflow is already behind an HTTP Listener. This mechanism only sees operations defined on an **HTTP Listener** with:

- a non-empty `Slug`
- a non-empty `Version`
- an OpenAPI document (`Document`) in YAML or JSON, with `Operations` NSwag can parse.

If your workflow isn't already reachable through an HTTP Listener plus OpenAPI document, set that up first, that part is unrelated to MCP and uses system's existing HTTP Listener tooling.

3: Fields to turn an operation into a tool

Add the following fields as vendor extensions on the specific OpenAPI **operation**, not the document root, not the path, but the operation object (for example, under `post:` for `/orders/{id}/status`).

Field	Required	Purpose	Fallback, if required
`x-mcp-enabled`	Yes	Opts this operation into MCP discovery. Must be `true` (boolean or the string `"true"`). Anything else, or absence, means the operation is invisible to agents.	Operation is skipped entirely — no tool is created.
`x-operation-handler`	Yes	Object with `name` (the handler that actually runs the workflow) and `workflowSpaceId`. Without a resolvable `name`, the mapper refuses to build the tool.	Operation is skipped entirely.
`x-mcp-name`	No	Overrides the tool name shown to agents.	Falls back to `operationId`, then to `{httpmethod}_{path}`, sanitized.
`x-mcp-description`	No	Overrides the tool description shown to agents.	Falls back to the operation's `description`, then `summary`, then an auto-generated sentence ("Invokes the POST /orders/{id}/status operation of the Orders HTTP Listener.").

`x-mcp-enabled` and `x-operation-handler` are the only two that are load-bearing. The other two are quality-of-life for how the tool presents itself to an LLM — worth setting deliberately, since the description is what the agent reads to decide whether to call your tool at all.

Worked example

A (fictional) *Get Order Status* workflow, exposed on an HTTP Listener with slug `orders` and version `v1`:

```yaml
paths:
  /orders/{orderId}/status:
    get:
      operationId: getOrderStatus
      summary: Get order status
      x-mcp-enabled: true
      x-mcp-name: get_order_status
      x-mcp-description: >-
        Returns the current status, last update timestamp, and carrier
        tracking number for a given order ID.
      x-operation-handler:
        name: GetOrderStatusHandler
        workflowSpaceId: "3a11b0e2-...-workflow-space-guid"
      parameters:
        - name: orderId
          in: path
          required: true
          schema:
            type: string
          description: The Sequence order identifier (GUID).
      responses:
        "200":
          description: OK
```

With `x-mcp-name` set, the agent sees `orders_get_order_status` tool (listener slug + sanitized name, joined with _). Leave `x-mcp-name` off and it falls back to `orders_getorderstatus` (from `operationId`), or `orders_get_orders_orderid_status` if neither is set.

4: Input schema and what makes an operation silently unusable

The mapper derives the tool's JSON schema straight from your OpenAPI operation. You must know these rules, because a violation doesn't error loudly and the operation is just **dropped**, with only a trace log to explain why:

- **Path and query parameters** map to schema properties directly. **All path parameters are treated as required**, regardless of what `required` says on the parameter.
- **Request body:** only `application/json` is read. Any other content type or a body schema that isn't a JSON object with properties, makes the whole operation unusable as a tool.
- **Body parameter style** (`in: body`) is not supported. Use an OpenAPI 3 `requestBody` with a JSON schema instead.
- Supported HTTP methods: `GET` , `POST` , `PUT` , `PATCH` , `DELETE` . Anything else is skipped.
- Nested objects and arrays are supported to a **depth of 8**; deeper than that, the operation is dropped.
- A parameter and a body property **cannot share the same name**, that collision drops the operation.
- Supported primitive types: `string` , `integer` , `number` , `boolean` , `array` , `object` . A schema type outside this set (for example, `null` -only) drops the operation.

TIP
Keep your request shape flat and simple, path params for identifiers, a single JSON body object for the rest, primitive or lightly-nested properties. If your tool doesn't show up, this section is where to look first, followed by the trace log entry:

```
"Dynamic MCP tool discovery skipped operation {method} {path} of HTTP Listener '{slug}'..."
```

5: Naming collisions

Tool names must be unique. There are two sources of collision:

- **Built-in tools**: `echo`, `calculator`, and `get_time` are reserved names baked into `ToolExecutorService`. Don't shadow them.
- **Other dynamic tools**: if two operations (in the same or different listeners) resolve to the same sanitized name, whichever is processed second is skipped, with a trace log naming the listener and operation that lost.

Tool names are lowercased and sanitized (non-alphanumeric → `_`, collapsed, then truncated to 128 characters), so `Get Order Status!!` and `get-order-status` collide. Setting an explicit, distinct `x-mcp-name` per operation is the safest way to avoid this once you have more than a couple of MCP-enabled listeners.

6: Publish latency

Tool definitions aren't rebuilt on every discovery call. `DynamicHttpListenerToolProvider` caches per-listener results and only reloads HTTP Listener definitions on an interval, controlled by:

```
<sequence.engine/agenticFabric DynamicToolsReloadIntervalSeconds="300" />
```

The default is 300 seconds (5 minutes) if unset. After you edit an operation's `x-mcp-*` extensions or add a new one, expect up to that interval before it shows up via `GET mcp/v1/tools`. There is no manual “refresh now” surfaced to workflow developers, so plan your testing loop accordingly and don't assume a change is broken just because it hasn't appeared yet.

The cache key is the listener's slug + version + document type + raw document content, so any edit to the document (not just the MCP-related fields) forces a rebuild on the next scheduled reload.

7: Verify your tool is live

1. Wait for the reload interval (see #6), or restart the engine service if you need it immediately.
2. Call `GET mcp/v1/tools` on the Sequence Integration API. Your tool should appear in the `DiscoverToolsResponse` list, with the name and description you expect and the input schema you designed.
3. Call `POST mcp/v1/invoke` with `toolName` and matching `arguments` to confirm the workflow actually runs and returns a `ToolExecutionResult` with `success: true`.
4. Only after that, you can expect it to be callable end-to-end through an agent via the MCP server (`agentic-mcpserver`, `POST /mcp`). That hop adds On-Behalf-Of (OBO-token) plumbing that's outside your control as a workflow developer, see #8.

> **NOTE**
> If #2 doesn't show your tool, then re-check #3, is `x-mcp-enabled: true`, and `x-operation-handler.name` resolvable? and #4, does your schema violate any of the drop rules?, before assuming a caching issue.

8. What happens downstream

This section explains the rest of the path so you understand what's happening after your tool is

discoverable. None of it requires action from you as a workflow developer, but it explains a class of failures you'd otherwise misattribute to your workflow.

- An agent calls the MCP server (`agentic-mcpserver`) over `POST /mcp` , using its own OBO-derived bearer token.
- The MCP server forwards the call to Sequence's `mcp/v1/invoke` endpoint using OBO delegation, meaning the workflow runs as the end user whose thread triggered the agent, not as a service account.
- If OBO consent or scopes are misconfigured at the platform level, your workflow will fail with an auth error that has nothing to do with your OpenAPI annotations. That configuration lives in the Fabric's app-registration setup, not in your HTTP Listener.
- The MCP server is stateless and honors `X-Forwarded-Proto` , since Sequence typically sits behind a reverse proxy /TLS terminator, irrelevant to you unless you're debugging a redirect or URL-mismatch issue.

Open item · pending confirmation

Confirmation is needed from the Agentic Fabric team on the current OBO scope names, and on which Entra app registration workflow developers should request access under. This was not covered in the source documentation reviewed for this article.

9. Checklist

	Item
✓	Workflow is reachable via an HTTP Listener with a Slug, Version, and OpenAPI document.
✓	Target operation has `x-mcp-enabled: true` .
✓	Target operation has `x-operation-handler.name` (and workflowSpaceId if applicable) pointing at a resolvable handler.
✓	`x-mcp-name` and `x-mcp-description` are set explicitly and describe the tool the way you'd want an LLM to read it, don't rely on the auto-generated fallback for anything user-facing.
✓	Request shape is flat: path/query params for identifiers, one JSON body object for the rest, nesting depth ≤ 8, no name collisions between params and body properties.
✓	Tool name doesn't collide with `echo` , `calculator` , `get_time` , or another dynamic tool.
✓	Verified via `GET mcp/v1/tools` and a direct `POST mcp/v1/invoke` call before expecting agent-level success.