Runtime MCP Reference

Live-runtime MCP surface for FrameworX. Read tags, alarms, historian, and UNS state from a running TServer.

New in 10.1.5. RuntimeMCP gains an HTTP transport on port `10120`, a Windows Service wrapper, target rebinding via `runtime_connect (solution, profile)` for automated testing workflows that switch between Execution Profiles without restarting the MCP process, and a pre-auth `runtime_describe_binding` tool that returns the bound TServer's `/health` payload without requiring authentication.

Tool catalog refresh (10.1.5). Several earlier standalone tools were merged into broader tools: `runtime_set_target` is now the with-args branch of `runtime_connect`; `runtime_find_by_iri` is now `runtime_describe_object(iri=…)`; `runtime_get_info` is the no-args branch of `runtime_connect`; `runtime_get_object_context` folded into `runtime_describe_object`; `runtime_browse_uns` became `runtime_browse_object_model`; `runtime_list_by_type` became `runtime_list_instances`; `runtime_search_tags` became `runtime_search_uns`; `runtime_list_derived_types` was removed (folded into `runtime_describe_object.derivedTypes`). See the full deprecation table on AI Runtime Connector.

RuntimeMCP is the MCP server that speaks to a **running** FrameworX runtime. Unlike DesignerMCP (design-time, live `.dbsln` in an IDE) and ConsoleMCP (design-time, file-based JSON workspace), RuntimeMCP targets **runtime state** — live tag values, active alarms, historical queries, and the UNS namespace as the runtime sees it. Use RuntimeMCP when AI-driven workflows need to observe or test a running solution, not author one.

Setup steps are covered in Claude Code MCP Setup and MCP and Claude Setup. Server-surface overview lives in MCP SDK Reference. Execution Profile details are in Runtime Execution Profiles Reference.

Vocabulary vs. DesignerMCP and ConsoleMCP
Server Variants
Connection Identity — (solution, profile)
Startup Modes (HTTP only)
Tool Catalog
runtime_set_test_value (Development only)
Diagnostic Tools
runtime_describe_binding
runtime_connect
runtime_describe_object
MCP Client Registration
Ad-hoc HTTP Client Handshake
Error Codes
Version History

Vocabulary vs. DesignerMCP and ConsoleMCP

RuntimeMCP tools carry a `runtime_` prefix so the mode is always obvious in a transcript. The design-time tools in DesignerMCP and ConsoleMCP work on configuration (the solution); the runtime tools work on state (the running process).

Design-time (Designer / Console)	Runtime (this server)	Object of action
`search_uns`	`runtime_search_uns`	Configured UserTypes / AssetTree vs. live UNS instances.
`get_objects`	`runtime_describe_object`	Configuration row vs. live object with inheritance and cross-references.
`browse_object_model`	`runtime_browse_object_model`	Namespace schema vs. live namespace walk.
(no equivalent — values are runtime-only)	`runtime_get_value`, `runtime_get_tag_history`	Live and historical tag values.
(no equivalent — alarms are runtime-only)	`runtime_get_active_alarms`, `runtime_query_alarm_history`	Live alarm state and history.

The `runtime_` prefix is load-bearing — do not unify names with DesignerMCP. A tool name is the agent's strongest cue about which world it is talking to.

Server Variants

RuntimeMCP ships as three executables, each tailored to a deployment shape.

Server	Transport	Target framework	Use case

RuntimeMCP	stdio	.NET 10	Per-session subprocess, spawned by Claude Desktop or Claude Code. One runtime per client session.
RuntimeMCPHttp	HTTP (SSE) on port `10120`	.NET 10	Long-running daemon. Supports target rebinding and multi-runtime setups. Required for any workflow that switches targets without restarting the MCP process.
RuntimeMCPHttpService	N/A (launches subprocess)	.NET Framework 4.8	Windows Service Control Manager wrapper. Installs, starts, and stops `RuntimeMCPHttp` as a system service. The service itself hosts no tools — it spawns `dotnet RuntimeMCPHttp.dll /service` as a child process and forwards lifecycle events.

All three share the same general tool surface. From FrameworX 10.1.6, the Development-only test tool `runtime_set_test_value` is available in the stdio `RuntimeMCP` only. The HTTP variants add bearer-token authentication through the configured API key in `RuntimeMCPHttp.json`. From FrameworX 10.1.6, `RuntimeMCPHttp` listens on the local machine only until you configure an API key (`X_API_KEY`). With a key, it listens on all network interfaces and requires the key on every request. Use a reverse proxy if you expose the HTTP surface beyond a trusted subnet.

From FrameworX 10.1.6, a remote client that connected to `RuntimeMCPHttp` without a key no longer connects until both sides use a key, sent in the `X-API-KEY` header.

Connection Identity — `(solution, profile)`

A FrameworX runtime is identified by **which solution** it is running and **which Execution Profile** it is running under. The same `.dbsln` can run concurrently under multiple profiles (Production, Development, Validation, Custom), each bound to its own TCP port via the solution's `RuntimeExecutionProfiles` configuration. Port is a consequence of the active profile, not an identity; editing profile port offsets in the solution changes the port without changing what solution or profile is running.

RuntimeMCP therefore accepts `solution` and `profile` as the connection coordinates. Port is looked up at bind time from InfoServer's registry of running TServers and returned to the client in the bind response for observability.

Profile values:

Integer	Name	Aliases accepted
`0`	Production	`prod`, `production`
`1`	Development	`dev`, `development`
`2`	Validation	`val`, `validation`
`3`	Custom	`custom`

Aliases are case-insensitive. Responses always echo both the canonical integer and the full profile name.

Startup Modes (HTTP only)

`RuntimeMCPHttp` supports two startup modes. The stdio `RuntimeMCP` binary is always process-per-session and retargeting it means restarting the subprocess — modes do not apply there.

Retargetable (default)

Accepts `runtime_connect(solution, profile)` at runtime to switch the bound TServer without restarting the process. Designed for automated tests that drive many targets from a single long-running MCP instance. The initial target can be supplied via `/solution` + `/profile` startup args or deferred until the first `runtime_connect(solution, profile)` call — tool calls before an initial target return `ERR_NO_TARGET`.

One Claude Code session per retargetable instance. Retarget state is held in the server process and shared across all connected clients — if two sessions hit the same retargetable instance, the second session's `runtime_connect(solution, profile)` silently retargets the first session's tools. For multi-session setups, run multiple `RuntimeMCPHttp` instances on distinct ports in pinned mode.

Pinned

Bound at startup via required `/solution` + `/profile` args, immutable for the process lifetime. The with-args branch of `runtime_connect` is rejected with `ERR_PINNED` in this mode — agents cannot retarget. The no-args branch (session-info query) remains available. Designed for A/B-style test setups: launch N `RuntimeMCPHttp` instances on N distinct listen ports, each pinned to its own `(solution, profile)`, and register each as a separate MCP client entry.

Startup argv

```
RuntimeMCPHttp.exe [/mode:retargetable|pinned]
                   [/solution:"<name>"] [/profile:0-3]
                   [/listen:<port>]         # default 10120
                   [/AllowTestTools:true] [/AllowControlActions:true]
```

`/mode` defaults to `retargetable` if omitted. In `pinned` mode both `/solution` and `/profile` are required; missing either is a startup error.

From FrameworX 10.1.6a, two options list tools that are off by default, on both `RuntimeMCP` and `RuntimeMCPHttp`. `/AllowTestTools:true` lists the Development-only test tools: the client session tools and, on `RuntimeMCP`, `runtime_set_test_value`. `/AllowControlActions:true` lists `runtime_force_switchover`, which commands a redundancy switchover and is marked as destructive, so AI clients that confirm destructive actions ask before it runs. On `RuntimeMCPHttp`, each option takes effect only when the server accepts local connections only.

Tool Catalog

Category	Tools
Tag read & browse	`runtime_get_value`, `runtime_browse_object_model`
Semantic search	`runtime_search_uns`, `runtime_describe_object` (accepts `iri=…`), `runtime_list_instances`
Alarms	`runtime_get_active_alarms`, `runtime_query_alarm_history`
Historian	`runtime_get_tag_history`, `runtime_get_aggregated_history`
Wait	`runtime_wait_for` (from FrameworX 10.1.6, stdio and HTTP. Blocks until a tag, alarm or module condition is true, or times out. Read-only. Parameters and result fields: AI Runtime Connector.)
Test stimulus (Development only)	`runtime_set_test_value` (from FrameworX 10.1.6, stdio only. Sets one tag value in a Development test runtime. See *runtime_set_test_value* below.)
Session / target control	`runtime_connect` (no-args: session info; with-args: bind or retarget on HTTP retargetable instances)
Pre-auth introspection	`runtime_describe_binding` (anonymous; reads `/health`)
Diagnostics	`runtime_debug_display_bindings`, `runtime_debug_module_states`, `runtime_debug_process_info`, `runtime_debug_recent_alarm_fires`, `runtime_debug_recent_events`

Note. RuntimeMCP does NOT expose any write to solution configuration — no `write_objects`, no `delete_objects`, no `rename_objects`. Configuration writes belong to DesignerMCP or ConsoleMCP. RuntimeMCP has no `set_object_value` tool and no general write to live Object Model values. `set_object_value` is an AI Designer tool, described in designer_action Reference. From FrameworX 10.1.6, the Development-only `runtime_set_test_value` sets tag values in test runtimes.

10.1.5 additions. `runtime_get_aggregated_history` sits alongside `runtime_get_tag_history` and returns time-bucketed aggregates (min, max, avg, count) computed server-side over a query range — the same shape exposed by the Runtime API `/aggregates` endpoint and the `TK.GetTagHistorianAggregated` script API.

---

runtime_set_test_value (Development only)

From FrameworX 10.1.6, `runtime_set_test_value` sets the value of one tag in a Development test runtime, the same action a human tester performs by setting a tag value in a running solution. Automated tests use it to drive end-to-end behavior, for example to push a value past an alarm limit and confirm the alarm fires, or to feed a sequence of readings and confirm the scripts and the historian react. The value goes to the object model exactly as an operator write does. It is not device-level simulation.

The runtime checks each condition below before it writes, and refuses the call when one of them fails:

- The runtime runs on the Development profile with its `QaTestMode` gate active.
- The solution is a disposable test solution whose name starts with `ai-test-` or `QA-TEST-`.
- The caller reaches the runtime over the local loopback address (`127.0.0.1` or `::1`). Connect by loopback address, not by machine name or network address.
- The tag path is in the `Tag.*` namespace.

Arguments: `name` is the full tag path, for example `Tag.Line1/Tank1/Level`. `value` is a string, number or boolean, converted to the tag's type by the runtime. On success the response carries `wrote: true`, and the runtime has committed the value. A refusal returns an `errorCode` such as `QA_TEST_MODE_REQUIRED`, `SANDBOX_REQUIRED`, `LOOPBACK_REQUIRED`, `INVALID_NAMESPACE` or `WRITE_REJECTED`, and changes nothing. Each accepted write adds one row to the audit trail, with the value masked. The tool is available in the stdio `RuntimeMCP` only, not over HTTP.

A value written to a tag mapped to a device output reaches the device. Run test solutions with the Devices module disabled.

Diagnostic Tools

The `runtime_debug_*` family exposes internal runtime state for troubleshooting workflows. These tools are **read-only diagnostics** — they do not mutate runtime state, do not require the AI Runtime-Write Gate, and do not open `DataAccess` connections beyond what the bound TServer already serves. Use them when a tag, alarm, display, or module is behaving unexpectedly and the configuration looks correct.

All diagnostic tools require the MCP to be bound to a target (same as any other authenticated `runtime_*` tool) and inherit the bound TServer's bearer-token authentication. They return `ERR_NO_TARGET` on a retargetable instance that has not yet been bound.

Tool	What it returns	Typical use case
`runtime_debug_display_bindings`	Live Display binding state — which Display elements are bound to which tags or expressions, last evaluation result, and binding errors surfaced by the renderer.	Triage a Display that renders blank, shows #ERR, or fails to update when the underlying tag changes. Confirms whether the binding resolved at all and what value the renderer last received.
`runtime_debug_module_states`	TModule lifecycle state for all loaded modules (Alarm, Device, Historian, Dataset, Script, Report, Security, Display, TagProviderService, Info, etc.) with one of `Started` / `Stopped` / `Faulted` / `Loading` per module.	Confirm which modules came up cleanly after a runtime start. A `Faulted` module usually points at the root cause of a feature being silently absent (no alarms firing, no historian writes, no device polling).
`runtime_debug_process_info`	Process-level diagnostics: PID, listen port, working-set memory, uptime, .NET runtime version, and the bound solution / profile.	Confirm **which TServer process** the MCP is actually talking to when several runtimes are up side by side, or capture a memory / uptime snapshot for a stability investigation.
`runtime_debug_recent_alarm_fires`	Recent alarm fire events from the runtime's in-memory ring buffer (last N entries, newest first). Each entry carries the alarm name, fire timestamp, severity, and the tag value that crossed the threshold.	Investigate "did this alarm actually fire?" without scanning the persistent alarm history. Faster than `runtime_query_alarm_history` for recent-events triage, and surfaces fires that never made it to history because the Alarm module faulted mid-flight.
`runtime_debug_recent_events`	Recent system events from the runtime's in-memory ring buffer (last N entries, newest first) — module start/stop, configuration reload, client connect/disconnect, security events, and other lifecycle signals.	Reconstruct what the runtime was doing in the seconds before a symptom appeared. Pairs with `runtime_debug_module_states` to distinguish "module faulted at startup" from "module faulted later after a reload."

Privilege gating. None of the `runtime_debug_*` tools require the AI Runtime-Write Gate — they are read-only. They do inherit the bound TServer's authentication: an HTTP variant rejects unauthenticated calls with `401`, same as any other authenticated `runtime_*` tool. The diagnostic surface intentionally exposes internal runtime state, so do not expose the HTTP endpoint beyond a trusted subnet without a reverse proxy.

Ring-buffer scope. The two `runtime_debug_recent_*` tools read from in-memory ring buffers sized at runtime start and bounded by available memory; older entries are evicted silently. For audited or long-window queries, use `runtime_query_alarm_history` (alarms) or the Historian / Events module surfaces — the debug tools are for live triage, not the system of record.

---

runtime_describe_binding

Pre-auth introspection tool. Returns the runtime's `/health` payload without performing authentication, so an automated verifier can confirm **which solution a TServer is actually bound to** before issuing authenticated `runtime_get_value` / `runtime_browse_object_model` calls.

Why pre-auth matters: authenticating against the wrong solution produces confusing downstream failures (tag-not-found, namespace-empty, asset-tree mismatch) that look like runtime bugs but are bench-misalignment. Catching the mismatch pre-auth is one MCP call vs. minutes of false-positive triage.

```
# No args: resolve the target via this MCP's bound (host, port).
runtime_describe_binding()

# Or probe an arbitrary runtime by URL (cross-bench check; this MCP's
# binding is NOT changed).
runtime_describe_binding(tserver_url="host:3201")
runtime_describe_binding(tserver_url="http://host:3201")
runtime_describe_binding(tserver_url="http://host:3201/health")
```

Response shape:

Error rendering macro 'code': Invalid value specified for parameter 'com.atlassian.confluence.ext.code.render.InvalidValueException'

```
{
  "success":         true,
  "url":             "http://localhost:3201/health",
  "resolved_from":   "current_binding",       // or "argument" | "tserver_cmdline_scan"
  "http_status":     200,
  "status":          "healthy",               // /health "status" field
  "version":         "10.1.5",
  "solution":        "SolarPanels MCP Demo",
  "started_at":      "2026-05-15T18:02:11Z",
  "uptime_seconds":  4231
}
```

From FrameworX 10.1.6, every response, success or error, also carries an `mcp_host` block describing this MCP server, not the probed runtime: `transport` (`stdio` or `http`), `listen_port` (null on stdio), `listen_scope` (`stdio`, `loopback` or `any_interface`), `local_only` and `instance`. A second instance of the HTTP server reports its own instance number. `transport` names the MCP transport, not the URL scheme, so do not build a URL from these fields.

Resolution order when `tserver_url` is omitted: (1) this MCP's bound `CurrentHost` / `CurrentPort`, (2) the MCP's startup args (`/Host+/Port` or legacy `/ip1+/port1`), (3) in 10.1.5 only, a command-line scan of running TServer processes on the host. From FrameworX 10.1.6, the tool never scans for a runtime. With no bound target and no startup args, it returns `ERR_NO_TARGET`. `resolved_from` tags which path produced the answer so the caller can see whether the response reflects the MCP's binding or a fallback.

The tool is **pure** — no `DataAccess` connection is opened, no search cache is touched, no log is written, the MCP's binding is not changed. The cross-bench form (with `tserver_url`) leaves the MCP pointed at its existing target.

Error code	When
`ERR_INVALID_URL`	`tserver_url` is malformed, OR no `tserver_url` was supplied and the resolution chain returned nothing. From FrameworX 10.1.6, the second case returns `ERR_NO_TARGET` instead.
`ERR_TIMEOUT`	The `/health` GET exceeded the 5-second timeout.
`ERR_UNREACHABLE`	Connection refused, TLS handshake failed, or the host is otherwise unreachable. The inner exception message is surfaced in `error`.
`ERR_INVALID_RESPONSE`	The endpoint returned non-JSON or a shape the tool could not parse. First 512 bytes of the body are surfaced in `raw_body_first_512` for diagnostics.
`ERR_INTERNAL`	Any other unexpected failure.

503 unhealthy. A runtime that is up but mid-startup or shutting down returns `http_status: 503` with `status: "unhealthy"` and a `reason` field; `success` is then `false` but the tool returns normally so the caller can branch on the precise status.

---

runtime_connect

Single entry point for session and target control. Two branches based on argument presence; both are exposed under the same tool name to keep agent transcripts uncluttered.

No-args branch — session info

Returns the current binding (solution, profile, host, port), the server variant (stdio / HTTP retargetable / HTTP pinned), and version metadata. Always available, regardless of mode. Useful as a first call to confirm what the MCP is talking to.

```
runtime_connect()
```

With-args branch — bind or retarget

Binds (or rebinds) a retargetable `RuntimeMCPHttp` instance to a running TServer identified by `(solution, profile)`. Disconnects from the current target, resolves the new target through InfoServer, reconnects, and invalidates the in-process search cache. Atomic from the caller's perspective — on failure the previous target is left intact.

In pinned mode the with-args branch is rejected with `ERR_PINNED`; run a separate `RuntimeMCPHttp` instance for each pinned target.

```
runtime_connect(
  solution="SolarPanels MCP Demo",
  profile=1            # or "dev"
)
```

Parameter	Required	Notes
solution	With-args branch only	Solution name (case-sensitive) as reported by InfoServer for a running TServer.
profile	With-args branch only	Integer 0–3 or case-insensitive alias (see Connection Identity above).

With-args response includes the new bound target and, if the server was previously bound, the target it was on before:

Error rendering macro 'code': Invalid value specified for parameter 'com.atlassian.confluence.ext.code.render.InvalidValueException'

```
{
  "solution": "SolarPanels MCP Demo",
  "profile": 1,
  "profileName": "Development",
  "host": "localhost",
  "port": 3201,
  "connected": true,
  "previousTarget": {
    "solution": "Industrial Ontology Demo",
    "profile": 0,
    "profileName": "Production",
    "host": "localhost",
    "port": 3101
  }
}
```

In 10.1.5, all resolved targets use `host = "localhost"`; remote TServer binding is not supported. The field is returned for forward compatibility.

Replaces. The earlier standalone `runtime_set_target` tool is now the with-args branch of `runtime_connect`; the earlier `runtime_get_info` tool is now the no-args branch. Existing automation that called either should switch to `runtime_connect` with the same payload (with-args) or no arguments (no-args). See the deprecation table on AI Runtime Connector.

---

runtime_describe_object

Returns a single live object as the runtime sees it — inheritance chain, derived types, cross-references, and effective values. Two argument shapes select the object: by `name` (UNS-pathed or short name resolved against the runtime's namespace) or by `iri` (the runtime's canonical Internationalized Resource Identifier for an object). Exactly one of the two must be supplied.

```
# By name (UNS-pathed or short)
runtime_describe_object(name="Tag.Plant.Line1.Temperature")

# By canonical IRI
runtime_describe_object(iri="urn:tatsoft:fx:tag:Plant/Line1/Temperature")
```

The response shape is identical across both branches — a single resolved object with its derived types in `derivedTypes`, its inheritance chain in `inheritsFrom`, and the values the runtime is currently serving. The branch is a convenience over how the caller knows the object, not a difference in what is returned.

Replaces. The earlier standalone `runtime_find_by_iri` tool is now the `iri=…` branch of `runtime_describe_object`; the earlier `runtime_get_object_context` tool merged into this tool as well; the earlier `runtime_list_derived_types` tool was removed entirely — its data is now in the `derivedTypes` field of this response. See the deprecation table on AI Runtime Connector.

MCP Client Registration

For MCP client configuration (Claude Desktop, Claude Code, VS Code Copilot), register each `RuntimeMCPHttp` instance as its own HTTP MCP entry. Name the entries semantically — `runtime_{solution}_{profile}` for pinned instances; `runtime` or `runtime_retargetable` for a retargetable instance. Never encode the port in the entry name: ports can change when profile offsets are edited in the solution, and a port-named entry becomes a lie the moment that happens.

Example: side-by-side Dev vs. Production (pinned)

```
# Launch two RuntimeMCPHttp instances pinned to different profiles
RuntimeMCPHttp.exe /mode:pinned /solution:"SolarPanels" /profile:0 /listen:10120
RuntimeMCPHttp.exe /mode:pinned /solution:"SolarPanels" /profile:1 /listen:10121

# Register in the MCP client as two separate entries:
#   runtime_solarpanels_prod  -> http://localhost:10120
#   runtime_solarpanels_dev   -> http://localhost:10121
```

Tools from each entry carry that entry's name as a prefix in the agent's transcript, so `runtime_solarpanels_dev.runtime_search_uns` is unambiguously the Development runtime, and side-by-side comparisons are legible.

Example: single retargetable instance for automated tests

```
# Launch one RuntimeMCPHttp in retargetable mode, no initial target
RuntimeMCPHttp.exe /mode:retargetable /listen:10120

# Register as a single entry: runtime_retargetable -> http://localhost:10120
# Scripted tests call runtime_connect(solution, profile) before each scenario.
```

Ad-hoc HTTP Client Handshake

`RuntimeMCPHttp` uses stateful Streamable HTTP. Any ad-hoc client — bare-shell `curl`, Postman, or a custom HTTP wrapper — must perform a three-step handshake before issuing tool calls:

1. POST `initialize` and capture the `Mcp-Session-Id` response header.
2. POST `notifications/initialized` with that `Mcp-Session-Id` header.
3. POST `tools/list` or `tools/call` with the same `Mcp-Session-Id` header.

Skipping the `initialize` step returns:

```
-32000 Bad Request: A new session can only be created by an initialize request
```

Claude Code, Claude Desktop, and any other MCP-aware client perform this handshake transparently — the rule applies only when a script or operator drives the HTTP endpoint directly. The same `Mcp-Session-Id` must be reused for the lifetime of the session; new POSTs without it are treated as fresh sessions and rejected on subsequent tool calls.

Error Codes

Code	Where	Meaning
`ERR_NO_TARGET`	Any `runtime_*` tool	Retargetable instance has not been bound yet. Call `runtime_connect(solution, profile)` first. From FrameworX 10.1.6, every stdio or HTTP instance started without a named runtime answers `ERR_NO_TARGET` until `runtime_connect(solution, profile)` binds it, and it never attaches to another running runtime.
`ERR_PINNED`	`runtime_connect` (with-args branch)	Instance is in pinned mode and does not accept retargeting. The pinned target is reported in the error message. Run a separate instance for the desired target.
`ERR_SOLUTION_NOT_RUNNING`	`runtime_connect` (with-args branch)	InfoServer has no record of `solution` currently running under any profile.

`ERR_PROFILE_NOT_RUNNING`	`runtime_connect` (with-args branch)	Solution is running but not under the requested profile.
`ERR_BIND_FAILED`	`runtime_connect` (with-args branch)	Connection to the resolved `(host, port)` failed. The previous target (if any) is left intact.

Version History

FrameworX	Notes
10.1.4	Initial ship of RuntimeMCP (stdio) with the runtime tool surface.
10.1.6	New Development-only `runtime_set_test_value` (stdio only) for automated tests in disposable test solutions. An instance connects only to a runtime named at launch or by `runtime_connect(solution, profile)`, and never discovers one. `RuntimeMCPHttp` listens on the local machine only until an API key is configured. New `runtime_wait_for`. `runtime_describe_binding` returns an `mcp_host` block.
10.1.6a	`/AllowTestTools:true` lists the Development-only test tools, and `/AllowControlActions:true` lists `runtime_force_switchover`. Both are off by default.
10.1.5	HTTP variant (`RuntimeMCPHttp`) on port `10120`, Windows Service wrapper (`RuntimeMCPHttpService`), target rebinding via `runtime_connect(solution, profile)`, retargetable and pinned startup modes, and the pre-auth `runtime_describe_binding` tool. Tool catalog refresh: `runtime_set_target` merged into `runtime_connect` (with-args branch); `runtime_get_info` merged into `runtime_connect` (no-args branch); `runtime_find_by_iri` merged into `runtime_describe_object` (`iri=…` branch); `runtime_get_object_context` merged into `runtime_describe_object`; `runtime_browse_uns` renamed to `runtime_browse_object_model`; `runtime_list_by_type` renamed to `runtime_list_instances`; `runtime_search_tags` renamed to `runtime_search_uns`; `runtime_list_derived_types` removed (folded into `runtime_describe_object.derivedTypes`). See AI Runtime Connector for the full deprecation table.

In this section...