Local AI Configuration

Reference for configuring the Local AI endpoint, master enable, tool-category bits, and pointing at alternative LLM endpoints (cloud or alternate local).

AI Integration Local AI Configuration

Where the configuration lives

Local AI reads its full configuration from three columns on the existing `SolutionSettings` table. No new tables, no new columns — one solution has one Local AI configuration.

Column	Type	Role	Default
`ModelEnabled`	Boolean	Master kill-switch for ALL Local AI features in this solution. When `false`, every `ChatRequest` action and every `TK.AIExecute` call short-circuits with `status="disabled"`.	`false` — off until customer opts in
`ModelSettings`	String (JSON blob)	Six-key endpoint configuration: `URL`, `Name`, `Authorization`, `Headers`, `Info`, `TimeoutSeconds`.	`NULL` — defaults apply (local Ollama + `qwen2.5:7b-instruct`)
`ModelOptions`	Int (bitmask)	Master tool-surface bit and per-category tool-enable sub-bits. Same bitmask the AI Runtime Connector reads.	`0` — no tools exposed; `ChatRequest` returns `disabled` until master bit is set

Editing in the Designer

The Local AI configuration is edited from the **Local AI** tile on the Data Servers page (sibling of the OPC UA, DataHub, MQTT Broker, and MCP for Runtime tiles).

- **Enable Local AI** checkbox — toggles `ModelEnabled`. Master kill-switch.
- **Status** indicator — reachability probe against the configured endpoint URL. Cached for 30 seconds.
- **Endpoint URL** — read-only display of the resolved URL.
- **Settings** link — opens the structured 5-field editor for `ModelSettings`.
- **Model name** — read-only display of the configured `Name` field.

The `ModelSettings` JSON

Six fields, all defensive — an empty/missing/malformed `ModelSettings` column transparently resolves to defaults. Unknown extra keys are preserved across edit cycles, so future revisions stay forward-compatible.

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

```
{
  "URL": "http://localhost:11434/v1/chat/completions",
  "Name": "qwen2.5:7b-instruct",
  "Authorization": "NoAuth",
  "Headers": "",
  "Info": "Default local model. Apache 2.0, ~4.7 GB.",
  "TimeoutSeconds": 60
}
```

Key	Default	Notes
`URL`	`http://localhost:11434/v1/chat/completions`	Must speak OpenAI-compatible chat-completions JSON. Local Ollama, LM Studio (in OpenAI mode), vLLM, llama.cpp's server, or any cloud endpoint that conforms.
`Name`	`qwen2.5:7b-instruct`	Goes into the POST body's `"model"` field. Must match a model the configured endpoint can serve.
`Authorization`	`NoAuth`	Multi-line wire format. Line 1 = scheme (`NoAuth` / `BearerToken` / `BasicAuth` / `CustomAuth`); subsequent lines carry the value. Scheme matching is case-insensitive; the legacy literal `None` from pre-10.1.5 configurations is still accepted as an alias for `NoAuth`. Accepts `/secret:<Name>` tokens for SecuritySecrets resolution. See SecuritySecrets Authentication for Local AI.
`Headers`	empty	Multi-line `key: value` per line. Same format the WebData connector uses for custom HTTP headers. Accepts `/secret:<Name>` tokens.
`Info`	self-documenting block	Free-text description visible to anyone editing the configuration. Distinct from `SolutionSettings.Description`.

`TimeoutSeconds`	`60`	Wall-clock budget per LLM call, in seconds. Integer; valid range `30`–`600`. The complete turn — parse, POST, tool-loop, reply build — must finish inside this window or the call returns a `truncated` / `error` envelope. Out-of-range, missing, or malformed values fall back to the default. Re-read fresh on every call so edits via the Designer dialog take effect on the next call. Used by both `ChatRequest` and `TK.AIExecute`.

The configuration is parsed defensively on every Local AI call — the parse cost is negligible compared with the LLM round-trip, and there is no caching layer to invalidate when the JSON changes.

The `ModelOptions` bitmask

An integer column carrying independent enable bits. The bitmask is shared with the AI Runtime Connector and the AI Designer connector — the same bits gate the same tool categories regardless of which transport the LLM uses to call them.

Bit	Name	Effect when ON
`0x02`	`EnableRuntimeMCP` (master)	Master enable for the AI tool surface. Required for the `ChatRequest` action to call any tools. When OFF, `ChatRequest` returns `status="disabled"`. `TK.AIExecute` is unaffected by this bit (atomic calls have no tools).
`0x04`	`EnableUnsTools`	The LLM may read tag values, browse the namespace, and search the UNS during a chat turn.
`0x08`	`EnableAlarmTools`	The LLM may read active alarms and query the alarm history.
`0x10`	`EnableHistorianTools`	The LLM may query historian time-series data.
`0x20`	`EnableCustomTools`	The LLM may call solution-authored MCP Tool class methods.
`0x40`	`EnableDesignerMCP`	Reserved for the AI Designer connector. Do not reuse for Local AI features.
`0x80`	`EnableChatHistory`	Per-Display-panel transcript cache participates in `ChatRequest` calls. Default **ON** in new 10.1.5 solutions. `TK.AIExecute` always bypasses the cache regardless of this bit.

The five tool-category bits (`0x04`–`0x20`) are AND-gated against the master bit. A category bit ON without the master bit ON leaves the category effectively OFF.

Master gate order

Both consumer paths apply the gates in a fixed order:

1. **`ModelEnabled`** — if `false`, return `status="disabled"` immediately. No HTTP traffic. `latencyMs = 0`.
2. **`ModelOptions` bit `0x02`** — `ChatRequest` only: if the master tool-surface bit is OFF, return `status="disabled"`. `TK.AIExecute` skips this gate (no tools to expose).
3. **Per-category bits** — `ChatRequest` only: AND-ed against the master bit when assembling the tool catalog the LLM sees during a chat turn.

Pointing at a different LLM endpoint

Replace the `URL` and `Name` fields. Any OpenAI-compatible chat-completions endpoint works.

Local Ollama (default)

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

```
{
  "URL": "http://localhost:11434/v1/chat/completions",
  "Name": "qwen2.5:7b-instruct",
  "Authorization": "NoAuth",
  "Headers": ""
}
```

Remote Ollama on a GPU server

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

```
{
  "URL": "http://gpu-server-01.lan:11434/v1/chat/completions",
  "Name": "qwen2.5:14b-instruct",
  "Authorization": "NoAuth",
  "Headers": ""
}
```

The remote Ollama must be started with `OLLAMA_HOST=0.0.0.0:11434` and the firewall opened on TCP 11434.

OpenAI-compatible cloud endpoint with Bearer token

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

```
{
  "URL": "https://api.example-llm-provider.com/v1/chat/completions",
  "Name": "the-cloud-model-name",
  "Authorization": "BearerToken\n/secret:CloudLLMApiKey",
  "Headers": ""
}
```

The `/secret:CloudLLMApiKey` token resolves at call time from the SecuritySecrets vault — the actual API key never appears in the configuration. See SecuritySecrets Authentication for Local AI.

Endpoint with extra HTTP headers

Some providers require extra request headers (organization ID, project ID, region). Add them via the `Headers` field, one `Key: Value` pair per line:

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

```
{
  "URL": "https://api.example-llm-provider.com/v1/chat/completions",
  "Name": "the-cloud-model-name",
  "Authorization": "BearerToken\n/secret:CloudLLMApiKey",
  "Headers": "X-Organization-Id: org-12345\nX-Project-Id: /secret:CloudLLMProjectId"
}
```

Header values also accept `/secret:<Name>` tokens.

Configuration safety nets

The platform applies several safety nets to prevent silent misconfiguration:

- **Defensive defaults.** Empty / null / malformed `ModelSettings` falls back to the recommended local Ollama defaults. A solution with a corrupted JSON blob still works against the local default.
- **Status probe.** The Local AI tile in the Designer probes the resolved URL on a 30-second cache, surfacing a red indicator when the endpoint is unreachable. Use it before deploying.
- **Master kill-switch precedes everything.** A solution can be staged with full configuration and shipped with `ModelEnabled = false`. No LLM traffic flows until the customer toggles it ON.
- **Off-server short-circuit.** Secret resolution is a server-side operation. Calls reaching Local AI from a thin-client context cannot resolve secrets and fall through to a normal HTTP error reply — no silent unauthenticated POST.

What this page does NOT cover

- **Bringing up Local AI on a fresh machine.** See Local AI - First Install Walkthrough.
- **SecuritySecrets reference syntax and examples.** See SecuritySecrets Authentication for Local AI.
- **Reply envelope shape and status semantics.** See Local AI Reply Envelope Schema.
- **The `ChatRequest` Display action.** See ChatRequest Action Reference.
- **The `TK.AIExecute` script API.** See TK.AIExecute API Reference.

In this section...