Local AI Configuration Reference

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 configuration from the Solution Capabilities row for LocalAI: the master enable, the endpoint JSON and the Gateway IP.

Surface	Type	Role	Default
`Enabled`	Boolean	Canonical master kill-switch for ALL Local AI features in this solution. When `false`, every `ChatRequest` action and every `AI.Execute` call (canonical surface from the `T.Toolkit.LocalAI` namespace; legacy alias `TK.AIExecute` forwards to it) short-circuits with `status="disabled"`.	`false` — off until customer opts in
`Settings`	String (JSON blob)	Seven-key endpoint configuration: `Provider`, `URL`, `Name`, `Authorization`, `Headers`, `Info`, `TimeoutSeconds`.	`NULL` — defaults apply (`qwen2.5:7b-instruct` at the LAN-IP placeholder endpoint). The model must be reachable at the configured URL; it must run on a **separate networked machine** (ideally GPU), not on the FrameworX/TServer host. See the Settings JSON section below.
`Settings Ext`	String (IP /host)	**Gateway IP** (optional, from FrameworX 10.1.6). Names an egress FrameworX runtime server as `host` or `host:port`. When set, this solution's interactive AI chat goes to the egress server, which holds the endpoint and credentials and makes the outbound call. `AI.Execute` calls still go directly from this runtime to the endpoint. Surfaced in the Designer and in JSON import/export as `GatewayIP`. See the *Routing AI chat through an egress runtime* section below.	`NULL` — direct call (no gateway)

Editing in the Designer

The canonical edit surface is **Solution Capabilities**. The **Local AI** capability row carries:

- **Enabled** checkbox — toggles `SolutionCapabilities[LocalAI].Enabled`. 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. The URL points to the chat-completions endpoint (`/v1/chat/completions`), which accepts POST requests only and returns a browser error if opened directly. The link icon next to the URL opens `/v1/models` in your browser instead — a GET endpoint supported by Ollama, LM Studio, vLLM, and all OpenAI-compatible servers that lists installed models and confirms the server is reachable.
- **Settings** link, labeled **Edit Configuration** in the Designer — opens the **Edit Local AI Endpoint** dialog for `SolutionCapabilities[LocalAI].Settings`.
- **Model name** — read-only display of the configured `Name` field.
- **Gateway IP**, in the **Edit Local AI Endpoint** dialog. Optional egress routing for interactive AI chat, from FrameworX 10.1.6. Enter the egress runtime server as `host` or `host:port`, or leave the field blank to call the endpoint directly. See *Routing AI chat through an egress runtime* below and Remote and Cloud LLM Models.

The same Local AI tile is also surfaced on the Data Servers page (sibling of the OPC UA, DataHub, MQTT Broker, and MCP for Runtime tiles) as a convenience shortcut for engineers who reach for Data Servers first — both routes edit the same row.

The Settings JSON

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

Replace `192.168.1.50` with the IP of the machine running your model. **The model must not run on the FrameworX/TServer host** — run it on a separate networked machine (ideally with a GPU).

```
{
  "URL": "http://192.168.1.50:11434/v1/chat/completions",
  "Name": "qwen2.5:7b-instruct",
  "Authorization": "NoAuth",
  "Headers": "",
  "Info": "Recommended default model. Apache 2.0, ~4.7 GB.",
  "TimeoutSeconds": 60
}
```

Key	Default	Notes

`Provider`	absent — the OpenAI-compatible dialect applies	Selects the request dialect. The Designer suggests `Llama (OpenAI-compatible)`, `Claude (Anthropic)`, and `Mock (no model)`, but the combo is free-text and accepts any string. The rule that matters at runtime: a value **containing `anthropic`** (case-insensitive) engages the Anthropic Messages API dialect; anything else is treated as OpenAI-compatible. The value also decides which model IDs the **Model Name** combo offers.
`URL`	`http://192.168.1.50:11434/v1/chat/completions`	Must speak OpenAI-compatible chat-completions JSON. Ollama on a separate GPU host, LM Studio (in OpenAI mode), vLLM, llama.cpp's server, or any cloud endpoint that conforms. Replace `192.168.1.50` with the IP of the machine running your model (the model must not run on the FrameworX/TServer host).
`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. In the **Edit Local AI Endpoint** dialog this single value is split across two controls: the **Authorization** dropdown carries line 1 and the **Auth Value** box carries the remaining lines. There is no separate `AuthValue` key in the JSON. 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 `AI.Execute`.

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 on `SolutionSettings` 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"`. `AI.Execute` 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. `AI.Execute` 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. **`SolutionCapabilities[LocalAI].Enabled`** — if `false`, return `status="disabled"` immediately. No HTTP traffic. `latencyMs = 0`.
2. **`SolutionSettings.ModelOptions` bit `0x02`** — `ChatRequest` only: if the master tool-surface bit is OFF, return `status="disabled"`. `AI.Execute` 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

> **Recommended default and limited-hardware fallback**
>
> FrameworX recommends `qwen2.5:7b-instruct` as the default (Apache 2.0, ~4.7 GB, best reasoning and tool-call reliability) — the model used for new solutions, demos, and templates. It expects a GPU on a **separate networked machine**. For real use run `qwen2.5:7b-instruct` on a separate GPU machine — the floor even for demos. CPU-only produces only ~2–4 tokens/sec, too slow even for a demonstration; `qwen2.5:3b-instruct` is at most a last-resort for single-shot atomic tasks, never interactive chat. For maximum reasoning on a strong GPU, `qwen2.5:32b-instruct` is the performance tier. To switch any solution, update the `Name` field in the JSON examples below (and pull the matching model with `ollama pull <name>`). For cloud and remote GPU options, see Remote and Cloud LLM Models.

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

Ollama on a separate host (LAN)

The model must run on a separate machine — replace `192.168.1.50` with the IP of that machine. Do not use `localhost` or `127.0.0.1` as the recommended value; loopback means the model would need to run on the FrameworX/TServer host, which is not supported.

```
{
  "URL": "http://192.168.1.50:11434/v1/chat/completions",
  "Name": "qwen2.5:7b-instruct",
  "Authorization": "NoAuth",
  "Headers": ""
}
```

Remote Ollama on a GPU server

```
{
  "URL": "http://gpu-server-01.lan:11434/v1/chat/completions",
  "Name": "qwen2.5:32b-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

```
{
  "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:

```
{
  "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.

Anthropic (Claude) endpoint

In the **Edit Local AI Endpoint** dialog, set **Provider** to **Claude (Anthropic)**. This is the field that engages the Anthropic Messages API dialect: the combo is free-text, and any value containing `anthropic` (case-insensitive) selects it. Leave it on the OpenAI-compatible preset and the request is built in the wrong shape no matter how the headers are set. Anthropic authenticates with an `x-api-key` header rather than a bearer token, and requires an API version header on every request, so both go in the **Headers** box while **Authorization** stays `NoAuth` and **Auth Value** is left empty. The `BearerToken` shape shown in the previous examples does not apply here.

From FrameworX 10.1.6, `AI.Execute` also uses this dialect. In 10.1.5, only AI chat does.

Type one header per line in the **Headers** box, exactly as below. This is the form you enter in the Designer:

```
x-api-key: /secret:AnthropicApiKey
anthropic-version: 2023-06-01
anthropic-workspace-id: wrkspc_01ABC...
```

The same endpoint as a `SolutionCapabilities[LocalAI].Settings` blob, for anyone editing the JSON directly rather than using the dialog. The line breaks typed in the **Headers** box appear here as `\n` escapes, because a JSON string cannot contain a literal newline:

```
{
  "Provider": "Claude (Anthropic)",
  "URL": "https://api.anthropic.com/v1/messages",
  "Name": "claude-opus-4-8",
  "Authorization": "NoAuth",
  "Headers": "x-api-key: /secret:AnthropicApiKey\nanthropic-version: 2023-06-01\nanthropic-workspace-id:
wrkspc_01ABC...",
  "Info": "Anthropic Claude endpoint; key held in SecuritySecrets.",
  "TimeoutSeconds": 60
}
```

The third header, `anthropic-workspace-id`, is needed only for some keys. Anthropic issues three kinds, and the kind the site holds decides whether the header is required:

Anthropic API key type	`anthropic-workspace-id`
Workspace key (legacy) — belongs to the workspace it was created in	Not required
Personal or service account key, bound to one workspace when created	Not required
Personal or service account key, **not** bound to a workspace (identity-linked)	**Required on every request**

Omitting the header with an unbound key returns `HTTP 400`: *anthropic-workspace-id is required when authenticating with an identity-linked API key; send the id of the workspace this request acts in.* The Local AI status indicator surfaces this as a failed test, and no chat or `AI.Execute` call succeeds until it is set.

Find the workspace ID in the Anthropic Console under **Settings Workspaces**, in the **ID** column; it always carries the `wrkspc_` prefix. The Default Workspace does not list an ID there, so name one of the other workspaces. If the API key is linked to a service account, that service account must be a member of the workspace named in the header, or the call returns `HTTP 404 Workspace not found` instead.

The alternative is to create an API key bound to a single workspace, which needs no header at all. Both approaches work; the header is the one that leaves an existing organization-wide key in place.

Routing AI chat through an egress runtime (Gateway IP)

From FrameworX 10.1.6, a solution sends its interactive AI chat to a designated *egress* runtime server instead of calling the LLM endpoint from each runtime. Set **Gateway IP** (the `GatewayIP` field, persisted in `SolutionCapabilities[LocalAI].SettingsExt`) to the egress server as `host` or `host:port`. The chat goes directly from this runtime to the egress runtime, which holds the endpoint configuration and credentials and makes the outbound LLM call on this solution's behalf. This centralizes outbound chat traffic on one server and keeps the provider API key for chat off the individual solutions. This is the same per-solution `ServerIP` pattern used by Datasets database connections and remote Device channels. Leave the field blank to call the endpoint directly from this runtime (the default).

- **Port.** With no port, the egress is reached on port 3101, the Production profile port. For an egress on another port, such as the Development profile port 3201, enter `host:port`.
- **Egress server.** The egress is a FrameworX runtime server, not the Solution Center web service. On the egress, set its own **Gateway IP** to a network interface address of the egress machine itself. A loopback address does not qualify. The egress accepts forwarded chat only when it names itself this way, and otherwise answers *caller is not an authorized forwarding peer*.
- **Scope.** Only interactive AI chat is routed. `AI.Execute` calls from server-side scripts still go directly from this runtime to the endpoint, with this solution's own endpoint settings and key.

For the zone-crossing rationale and topologies, see Remote and Cloud LLM Models.

Configuration safety nets

The platform applies several safety nets to prevent silent misconfiguration:

- **Defensive defaults.** Empty / null / malformed `SolutionCapabilities[LocalAI].Settings` 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 `SolutionCapabilities [LocalAI].Enabled = false`. No LLM traffic flows until the customer toggles it ON in **Solution Capabilities**.
- **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 `AI.Execute` script API.** See AI.Execute API Reference.

In this section...