Solution Capabilities Reference

Reference for the Solution Capabilities surface — the per-solution table of big-module capabilities (master enable + JSON config), with the basic Local AI configuration as the worked example.

Technical Reference Solution Capabilities Reference

What Solution Capabilities are

A **capability** is a big-module feature a solution can carry, gated by a master enable and configured by a JSON blob. Capabilities live in the `SolutionCapabilities` table — a **fixed-slot, name-keyed** table: the platform seeds the capability rows, so you do not create or delete them. Each row carries a master `Enabled` switch and a per-capability `Settings` JSON. The first shipped capability is **Local AI** (the on-device AI engine); the platform seeds additional capability slots as new engines ship.

The canonical edit surface is **Solution Capabilities** in the Designer. Each capability appears as a tile with an Enable checkbox, a reachability / status indicator where applicable, and an Edit Configuration dialog for its `Settings`. The Local AI tile is also surfaced on the Data Servers page (sibling of the OPC UA, DataHub, and MQTT Broker tiles) as a convenience shortcut — both routes edit the same row.

The table

Three author-facing columns:

Column	Type	Role
`Name`	String	Capability row identifier (for example `LocalAI`). Seeded by the platform — fixed.
`Enabled`	Boolean	Master enable / kill-switch for the capability. Default **off** — capabilities are opt-in. When off, the capability's surfaces short-circuit (for Local AI, every call returns `status = "disabled"`).
`Settings`	String (JSON)	Per-capability configuration. An empty or missing value resolves to the platform defaults. The JSON schema is specific to each capability.

Additional internal columns exist on the row but are platform-managed and not edited directly.

Local AI — basic configuration

The **Local AI** capability (`SolutionCapabilities[LocalAI]`) is the on-device LLM engine. Operators chat from Display panels via the `ChatRequest` action; server-side scripts call the model atomically with `AI.Execute`. Two fields on the capability row drive it.

Enabled — the master kill-switch

Ships **off**. A solution can be fully configured and shipped with `Enabled = false`; no LLM traffic flows until an engineer ticks the box in **Solution Capabilities**. While off, both the `ChatRequest` Display action and the `AI.Execute` script API return `status = "disabled"` with no HTTP traffic.

Settings — the endpoint JSON

An empty or missing `Settings` resolves to the platform defaults: an Ollama endpoint seeded at `http://192.168.1.50:11434/v1/chat/completions` (replace `192.168.1.50` with the IP of the machine running your model — this is a placeholder pointing at a separate LAN host, not the TServer machine itself) using the recommended default model. The full JSON shape:

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

```
{
  "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	Notes
`URL`	OpenAI-compatible chat-completions endpoint. Local Ollama, LM Studio, vLLM, llama.cpp's server, or any conforming cloud endpoint.
`Name`	Model name placed in the POST body's `"model"` field. Must match a model the endpoint can serve.
`Authorization`	`NoAuth` / `BearerToken` / `BasicAuth` / `CustomAuth`. Accepts `/secret:<Name>` tokens for SecuritySecrets resolution.

`Headers`	Optional extra HTTP headers, one `Key: Value` per line. Accepts `/secret:<Name>` tokens.
`Info`	Free-text description shown to anyone editing the configuration.
`TimeoutSeconds`	Per-call wall-clock budget. Integer, valid range `30`–`600`; default `60`.

Full field-by-field reference, alternate-endpoint examples (remote Ollama, cloud with Bearer token, custom headers), and the safety-net behavior are on Local AI Configuration.

Which model

FrameworX standardizes on the **Qwen 2.5 instruct** family. Pick the tier that matches your hardware — `qwen2.5:7b-instruct` is the recommended default.

Model	Tier	Disk	When to use
`qwen2.5:7b-instruct`	**Recommended default**	~4.7 GB	Most deployments; the model used for new solutions, demos, and templates. Best JSON tool-call reliability. Expects a machine with a GPU.
`qwen2.5:3b-instruct`	Limited-hardware fallback	~2 GB	No GPU / very low spec. Lower speed and quality; not recommended for interactive chat — acceptable for atomic reporting, classification, and summary tasks.
`qwen2.5:32b-instruct`	Maximum performance	~20 GB	Best reasoning and multi-step tool logic. Requires a strong GPU.

Fully offline — the model runs with no internet connection, everything local. The LLM should run on a separate GPU machine; CPU-only on the same box produces only ~2–4 tokens/sec — too slow even for a demo. For real use run `qwen2.5:7b-instruct` on a separate GPU machine — the floor even for demos. CPU-only is not recommended for any interactive or operator use. Per-OS install steps are on Local AI - Installing Models (Windows, macOS, Linux).

Tool gating — ModelOptions

The tool surface the LLM may call during a chat turn (UNS reads, alarm queries, historian queries, solution-authored MCP tools) is gated separately by the `SolutionSettings.ModelOptions` bitmask — not by this capability row. The master tool bit and per-category bits are documented on Local AI Configuration.

Editing a capability

1. Open **Solution Capabilities** (or the Local AI tile under **Unified Namespace Data Servers** — both edit the same row).
2. Tick **Enabled** to flip the master kill-switch.
3. Click **Settings** (Edit Configuration) to set the endpoint `URL`, model `Name`, timeout, and authorization — or to point at a remote / cloud LLM.
4. The reachability status indicator turns green once the endpoint responds.

See also

- Local AI — the Local AI capability landing page.
- Local AI Configuration — full endpoint and tool-bit reference.
- Local AI - Installing Models (Windows, macOS, Linux) — install the runtime and pull the models per OS.
- SecuritySecrets Authentication for Local AI — store API keys for remote / cloud endpoints.

In this section...