Claude Code MCP Setup

Advanced command-line integration for FrameworX with Claude Code.
AI Integration | Claude Skill and MCP Setup | Claude Code MCP Setup

> **Avoid double-wiring an MCP backend.** Wire each FrameworX MCP (Designer, Console, Runtime) in EXACTLY ONE client on a given machine. If you have the FrameworX MCP enabled in **Claude Desktop** via Extensions (`.mcpb`), do NOT also `claude mcp add` it here — and vice versa. Each backend wired twice spawns two host processes per Claude session, and both leak as orphan processes when the session crashes or force-quits. Pick one client per backend.

Overview

Claude Code connects to FrameworX from the terminal — no GUI needed. Two MCP servers are available depending on your workflow:

AI Designer (`DesignerMCP`) connects Claude Code to the live Designer IDE on the same machine. Every tool call produces immediate visual changes — the same live AI Designer experience as Claude Desktop, but driven from the command line. Use this when you want live building with Designer open on your screen.

AI Console (`ConsoleMCP`) connects Claude Code to a file-based engineering workflow. No running Designer needed — Claude generates JSON configuration files following the ExportFormat 1.2 standard. Use `create_solution_file` (for a new solution) or `update_solution_file` (to update an existing `.dbsln` in place with auto-backup) to produce or refresh a deployable `.dbsln` — or open the workspace inside Designer. Use this for analyzing existing projects, generating solutions from specifications, or batch-building.

AI Designer Setup (Live in the IDE)
AI Console Setup (File-Based Engineering)
AI Designer Setup (Split Host / HTTP)
AI Console Setup (Split Host / HTTP)
AI Runtime Setup (Live Runtime Reader, stdio)
AI Runtime Setup (Split Host / HTTP)
write_objects — three input shapes
Test Your Setup
Troubleshooting
Quick Reference: Config Files
Next Steps
Community

You can install both. They register as separate MCP servers and do not conflict.

Scenario	Use
Claude Code on the same Windows machine as Designer — want live building in Designer	**AI Designer**
Generate configs without a running Designer	**AI Console**
Analyze existing projects, audit configurations	**AI Console**
Batch-build or generate solutions from specifications	**AI Console**
Interactively migrate solutions from other platforms	**AI Designer**
Batch/background migration from other platforms	**AI Console**

Two Vocabularies, One Pattern

The two modes use parallel-but-distinct tool names so it is always obvious which mode you are in:

AI Designer (live IDE)	AI Console (file-based)
`list_solutions`	`list_workspaces`
`open_solution`	`open_workspace`
`create_solution`	`create_workspace`
`get_solution_info`	`get_workspace_info`

AI Designer operates on `.dbsln` files (the live solution). AI Console operates on `-json/` workspace folders that hold the same configuration as separate JSON files. The `create_solution_file` and `update_solution_file` tools (AI Console) and the `from_workspace` parameter on AI Designer's `open_solution` / `create_solution` are the bridges between the two.

Prerequisites

- FrameworX v10.1.3+
- FrameworX 10.1.4 — requires .NET Desktop Runtime 8.0 (installed under `fx-10\net8.0\`)
- FrameworX 10.1.5 and later: requires the .NET Runtime 10.0 (the .NET Desktop Runtime 10.0 also works, since it includes it). The MCP servers are installed under `fx-10\net10.0\`. On Windows on ARM64, see below.

- The HTTP MCP servers (DesignerMCPHttp, ConsoleMCPHttp, RuntimeMCPHttp) also need the ASP.NET Core Runtime 10.0 on the machine that runs them; the .NET Desktop Runtime does not include it.
- Claude Code

Path note for the command lines below. The examples target **10.1.5+** with `fx-10\net10.0\`. If you are still on 10.1.4, substitute `fx-10\net8.0\`; on a side-by-side install with both runtimes present, pick the folder matching the FrameworX version you want this MCP server to talk to.

Windows on ARM64

On Windows on ARM64 (for example Windows 11 in Parallels on a Mac), install both the ARM64 and the x64 .NET Runtime 10.0:

- **ARM64**: the MCP servers start with `dotnet`, which on ARM64 is the ARM64 .NET host. That host does not use an x64 runtime.
- **x64**: Multiplatform solutions started from Designer or Solution Center run on it.

```
winget install --id Microsoft.DotNet.Runtime.10 --architecture arm64
winget install --id Microsoft.DotNet.Runtime.10 --architecture x64
```

Check. Run `dotnet --list-runtimes` and look for a `Microsoft.NETCore.App 10.` line whose path has no `x64` folder. A 10.x runtime only under `C:\Program Files\dotnet\x64\` is the x64 runtime, which `dotnet` does not use on ARM64.

Symptom. Without the ARM64 runtime, the AI Designer, AI Console and AI Runtime servers close as soon as they start, Claude reports "Connection closed", and no FrameworX tool is available. Running `dotnet ConsoleMCP.dll` from `fx-10\net10.0\` prints "You must install or update .NET to run this application" with "Architecture: arm64". **Fix:** install the ARM64 runtime, then restart Claude Code. If you cannot install it, register the x64 host instead: `claude mcp add --scope user consolemcp "C:\Program Files\dotnet\x64\dotnet.exe" "C:\Program Files\Tatsoft\FrameworX\fx-10\net10.0\ConsoleMCP.dll"`.

AI Designer Setup (Live in the IDE)

This connects Claude Code to the running FrameworX Designer IDE on the **same Windows machine**. Claude Code launches DesignerMCP directly as a subprocess — the simplest possible setup.

Step 1: Add the MCP Server

Open a terminal and run:

```
claude mcp add FrameworX-Designer -- dotnet "C:\Program Files\Tatsoft\FrameworX\fx-10\net10.0\DesignerMCP.dll"
```

This saves the configuration at project level by default. For other scopes:

- `-s project` — saves to `.mcp.json` in the project root (shared via version control)
- `-s user` — saves to global/user-level config, available in all projects

For a global install:

> For a global install (works in any folder):
>
> ```
> claude mcp add -s user FrameworX-Designer -- dotnet "C:\Program Files\Tatsoft\FrameworX\fx-10\net10.0\DesignerMCP.dll"
> ```

Step 2: (Optional) Add the Filesystem Server

The **filesystem** MCP server allows Claude to directly read files produced by `get_screenshot` and `get_solution_export_files`. Without it, those tools still work but you must manually provide the files. All other Designer tools work independently.

The filesystem server requires **Node.js** — install from https://nodejs.org if not already present.

```
 claude mcp add filesystem -- npx -y -p @modelcontextprotocol/server-filesystem@2025.8.21 -p zod@3.25.76 -p
@modelcontextprotocol/sdk@1.32.1 mcp-server-filesystem "C:\Users\Public\Documents\FrameworX\Exchange"
```

The package versions are pinned on purpose, because later releases of the filesystem server fail in Claude Code with an `invalid outputSchema` error. FrameworX 10.1.6 and later ship the FrameworX File Access extension with the same pinned versions.

Step 3: Verify

```
claude mcp list
```

You should see `FrameworX-Designer` (and `filesystem` if added) in the list.

Alternative: Manual Configuration

If you prefer editing config files directly, add to your `.mcp.json` or Claude Code MCP settings:

```
{
  "mcpServers": {
    "FrameworX-Designer": {
      "command": "dotnet",
      "args": [
        "C:\\Program Files\\Tatsoft\\FrameworX\\fx-10\\net10.0\\DesignerMCP.dll"
      ],
      "transport": "stdio"
    },
    "filesystem": {
      "command": "npx",
      "args": [
        "-y",
        "-p",
        "@modelcontextprotocol/server-filesystem@2025.8.21",
        "-p",
        "zod@3.25.76",
        "-p",
        "@modelcontextprotocol/sdk@1.32.1",
        "mcp-server-filesystem",
        "C:\\Users\\Public\\Documents\\FrameworX\\Exchange"
      ]
    }
  }
}
```

Claude Code handles tool authorization inline, it prompts on first use and remembers the decision for the session. No pre-configuration of permissions is needed.

AI Designer — from_workspace parameter

Both `open_solution` and `create_solution` accept an optional `from_workspace` parameter. When supplied, all `*.json` files in the workspace folder are imported into the solution after it is opened or created. Use this to apply a workspace authored in AI Console to an existing or fresh solution from inside AI Designer.

- `open_solution(name, from_workspace="C:\\Path\\Foo-json")` — opens the existing `.dbsln`, auto-backs it up (3-file retention), then imports the workspace files using always-replace upsert semantics.
- `create_solution(name, template="HeaderLayout", from_workspace="C:\\Path\\Foo-json")` — creates a fresh `.dbsln` from the template, then imports the workspace files.

Missing folder, empty folder, or pipe exception become warnings — the open / create still succeeds. The response includes a `workspaceImport` section with per-file results.

AI Designer — designer_action visual targeting

Beyond the existing `navigate` / `find_object` / runtime control actions, three new actions point the user's attention at a specific object:

Action	Purpose	Example
`select_row`	Select a specific row in a config grid by name and scroll it into view. Visual only, no mutation.	`designer_action('select_row', 'UnsTags.Plant1/Temperature')`
`scroll`	Alias of `select_row`. Same behavior.	`designer_action('scroll', 'UnsTags.Plant1/Temperature')`
`expand`	Expand a tree node in the AssetTree, select it, and scroll into view. V1 supports the AssetTree only.	`designer_action('expand', 'AssetTree.Plant1/Motor1')`

AI Console Setup (File-Based Engineering)

AI Console connects Claude Code to FrameworX without a running Designer. Claude generates JSON configuration files in a workspace folder that is later turned into a deployable `.dbsln` via the `create_solution_file` tool, or refreshed in place on an existing `.dbsln` via `update_solution_file` (auto-backup, always-upsert) — or by opening the workspace from inside Designer.

Step 1: Add the MCP Server

Local machine (stdio):

```
claude mcp add FrameworX-Console -- dotnet "C:\Program Files\Tatsoft\FrameworX\fx-10\net10.0\ConsoleMCP.dll"
```

> For a global install (works in any folder):
>
> ```
> claude mcp add -s user FrameworX-Console -- dotnet "C:\Program Files\Tatsoft\FrameworX\fx-10\net10.0\ConsoleMCP.dll"
> ```

Step 2: Verify

```
claude mcp list
```

You should see `FrameworX-Console` in the list.

Alternative: Manual Configuration

Local (stdio)

```
{
  "mcpServers": {
    "FrameworX-Console": {
      "command": "dotnet",
      "args": [
        "C:\\Program Files\\Tatsoft\\FrameworX\\fx-10\\net10.0\\ConsoleMCP.dll"
      ],
      "transport": "stdio"
    }
  }
}
```

How the File-Based Workflow Works

1. Claude calls `create_workspace` or `open_workspace` — creates or opens a `<name>-json/` workspace folder under `Documents/FrameworX/Exchange/`.
2. Claude calls `get_table_schema` to fetch column definitions, then `write_objects` to generate JSON files.
3. Each JSON file follows the ExportFormat 1.2 envelope structure.
4. Claude calls `create_solution_file(solution_name, workspace=...)` to produce a fresh `.dbsln`, or `update_solution_file(solution=..., workspace=...)` to refresh an existing one (auto-backup kept, always-upsert semantics). Both spawn `SolutionCreator.exe` headlessly.
5. (Alternate path) Open the workspace from inside Designer using DesignerMCP's `open_solution(name, from_workspace=...)` — Designer auto-backs up the target solution and imports the workspace files in place.

Available AI Console Tools

Category	Tools
Workspace management	`list_workspaces`, `open_workspace`, `create_workspace`, `get_workspace_info`, `delete_workspace` (from FrameworX 10.1.6)
Build & deploy	`create_solution_file`, `update_solution_file`, `verify_solution_file`
Object operations	`write_objects`, `get_objects`, `delete_objects`, `browse_object_model`
Schema queries	`get_table_schema`, `list_elements`, `list_dynamics`, `list_protocols`
Documentation	`search_docs`, `get_solution_export_files`

AI Console uses `*_workspace` tool names (file-based JSON folders). AI Designer uses `*_solution` (live Designer IDE). The underlying operations are parallel but the vocabulary distinguishes the two modes.

From FrameworX 10.1.6, the table lists the main AI Console tools without a tool count. The tool list in Claude Code shows the complete list for the installed version. AI Console has no `rename_objects`: rename an object by editing its `Name` with `write_objects`.

delete_workspace. From FrameworX 10.1.6, `delete_workspace(name)` deletes a disposable test workspace folder in one step. It accepts only names starting with `ai-test-` or `QA-TEST-` (`SANDBOX_REQUIRED` otherwise) and refuses with `WORKSPACE_IN_USE` while a runtime runs under the workspace name. A `.dbsln` built from the workspace stays in place.

Context Package on repeat calls. From FrameworX 10.1.6, `list_workspaces`, `open_workspace`, `create_workspace` and `get_solution_export_files` accept `context`: `full` (default), `summary` or `none`. Pass `summary` or `none` only when the conversation already holds the full package. The first bootstrap call of a server process always returns the full package. Details: AI Designer Connector.

create_solution_file — turn a workspace into a new .dbsln

Once a workspace has been authored with `write_objects`, call `create_solution_file` to produce the deployable `.dbsln`. The tool invokes SolutionCreator.exe headlessly and waits for it to finish.

```
create_solution_file(
  solution_name="Foo",
  workspace="C:\\Users\\Public\\Documents\\FrameworX\\Exchange\\Foo-json",
  template="HeaderLayout",
  platform="Windows"
)
```

If `workspace` is omitted, the active workspace folder set by `open_workspace` / `create_workspace` is used. Default template is `HeaderLayout`, default platform is `Windows`, default output folder is the standard Solutions folder. The response includes the produced `.dbsln` path, exit code, per-file import attribution (created / modified / skipped / errors), and the headless log. 10.1.5 additionally carries the shared `build` block with per-Script /Display compile status.

update_solution_file — refresh an existing .dbsln in place

Use `update_solution_file` when the target `.dbsln` already exists and you want to apply the workspace changes to it rather than produce a new file. The tool auto-backs up the existing `.dbsln` (keep-last-3 retention, `<name>.dbsln__backup__<timestamp>.backup`), then spawns SolutionCreator.exe headlessly. SolutionCreator detects the existing target and switches to open-existing mode. Workspace JSON files are imported using **always-upsert** semantics — existing objects with the same name are overwritten by the workspace version.

```
update_solution_file(
  solution="C:\\Solutions\\Foo.dbsln",
  workspace="C:\\Users\\Public\\Documents\\FrameworX\\Exchange\\Foo-json"
)
```

The `solution` argument must be an absolute path ending in `.dbsln` to an existing file; the folder and filename are decomposed internally. If `workspace` is omitted, the active workspace folder is used. The response includes the target path, the backup path, exit code, per-file import attribution, the headless log, and the shared `build` block (10.1.5+).

verify_solution_file — inspect a .dbsln after the fact (10.1.5)

Opens an existing current-version `.dbsln` and returns a per-table Name inventory plus the shared `build` block drawn from the `*Contents` tables. Optionally diffs the inventory against a caller-supplied list of expected object names — the response then includes `missing[]` (expected but not found) and `unexpected[]` (found but not expected) per table type. Useful for asserting that `create_solution_file` / `update_solution_file` produced the solution a specification called for.

AI Designer Setup (Split Host / HTTP)

Use this variant when Claude Code runs on a different machine from the FrameworX Designer. Examples: Claude Code on a Mac terminal driving a Designer inside a Parallels VM, or a laptop connecting to a shared Designer on the LAN.

The Windows machine runs `DesignerMCPHttp.exe` as a local HTTP MCP server (default port **10150**). Claude Code uses the `mcp-remote` bridge (an npm package) to forward MCP calls to that HTTP endpoint.

Prerequisites. FrameworX Designer installed on the Windows machine. Node.js LTS installed on the Claude Code host (for `mcp-remote`). Port reachable from the Claude Code host.

> **Updating to FrameworX 10.1.6: a split-host connection now needs an API key.** From 10.1.6, DesignerMCPHttp, ConsoleMCPHttp and RuntimeMCPHttp accept connections **only from the same computer** while `X_API_KEY` is empty. A split-host setup that worked in 10.1.5 without a key — including a Mac reaching a Windows VM through port forwarding — **stops connecting after the update**. To keep it working: set `X_API_KEY` in the server's `.json` settings file (32 or more letters and digits), restart the server, and add the key to the `mcp-remote` command as shown in Step 2. Setups where Claude Code and FrameworX run on the same computer are not affected. The connection sends the key over plain HTTP, so use it only on a trusted network.

Step 1: Start DesignerMCPHttp on the Windows machine

On the Windows machine where FrameworX is installed, open File Explorer and navigate to `Documents\FrameworX\Utilities`, then double-click **StartDesignerMCPHttp.bat**. A console window confirms the server is listening on port **10150**. Leave it running.

Step 2: Add the MCP Server on the Claude Code host

```
claude mcp add FrameworX-Designer -- npx -y mcp-remote http://127.0.0.1:10150/mcp
```

Use `127.0.0.1` for a local VM, or the LAN IP of the Windows box (for example, `192.168.1.50`). For a global install add `-s user`.

From FrameworX 10.1.6, a connection from another machine needs the server's key (`X_API_KEY` in `DesignerMCPHttp.json`), and `mcp-remote` needs `--allow-http` for a plain-HTTP address other than localhost:

```
claude mcp add FrameworX-Designer -- npx -y mcp-remote http://192.168.1.50:10150/mcp --allow-http --header X-API-KEY:<your key>
```

To override the port (when `DesignerMCPHttp.json` has a custom `ListenPort`), substitute in the URL. The default **10150** matches the shipped config.

Step 3: Verify

```
claude mcp list
```

You should see `FrameworX-Designer` in the list.

> Make sure port **10150** is open in Windows Firewall on the Designer machine. If Claude Code runs on the same host as the VM (Parallels / VMware), configure port forwarding from host to guest for **10150**.

AI Console Setup (Split Host / HTTP)

Same split-host pattern as the Designer HTTP variant, applied to a file-based workspace. Use this when Claude Code runs on a different machine than the one hosting the workspace folder, or when you want a shared ConsoleMCP server that multiple Claude Code sessions connect to.

Prerequisites: same as the Designer Split Host / HTTP section above. Node.js LTS must be installed on the Claude Code host because `mcp-remote` is an npm package.

The Windows machine runs `ConsoleMCPHttp.exe` on default port **10151**.

Step 1: Start ConsoleMCPHttp on the Windows machine

Run `ConsoleMCPHttp.exe` manually, or install it as a Windows Service for unattended operation. Port and API key live in `ConsoleMCPHttp.json` alongside the executable (default `ListenPort`: **10151**).

Step 2: Add the MCP Server on the Claude Code host

```
claude mcp add FrameworX-Console -- npx -y mcp-remote http://127.0.0.1:10151/mcp
```

Replace `127.0.0.1` with the LAN IP of the workspace host when they are on different machines. From FrameworX 10.1.6 that requires `X_API_KEY` in `ConsoleMCPHttp.json` and the key in the command: append `--allow-http --header X-API-KEY:<your key>` after the URL.

Step 3: Verify

```
claude mcp list
```

> The workspace folder must be reachable from the ConsoleMCPHttp host, not from the Claude Code host. The workspace path on the ConsoleMCP server side is what matters for `open_workspace` and `create_workspace` path resolution.

AI Runtime Setup (Live Runtime Reader, stdio)

AI Runtime (`RuntimeMCP`) connects Claude Code to a running FrameworX TServer to read live tag values, browse the operational namespace, inspect alarms, and query the historian. Use this when you need to answer questions about what is happening in a running solution, as opposed to configuring it. Same machine as the runtime.

Step 1: Add the MCP Server

```
claude mcp add FrameworX-Runtime -- dotnet "C:\Program Files\Tatsoft\FrameworX\fx-10\net10.0\RuntimeMCP.dll" /Host:localhost /Port:3101 /UserName:guest /Password:""
```

Runtime connection parameters:

- `/Host`: TServer host (default `localhost`).
- `/Port`: TServer port (default `3101`).

- `/UserName` and `/Password`: runtime login (use `guest` with empty password for anonymous access).

Add `-s user` for a global install. 11 platform tools ship in 10.1.5: `runtime_connect` (session bootstrap), `runtime_get_value`, `runtime_browse_object_model`, `runtime_search_uns`, `runtime_describe_object` (accepts a `path=` or `iri=` selector — absorbs the former `runtime_find_by_iri`), `runtime_list_instances`, `runtime_get_active_alarms`, `runtime_query_alarm_history`, `runtime_get_tag_history`, `runtime_get_aggregated_history` (10.1.5 new — server-side bucketed aggregation: Average / Minimum / Maximum / Total / Count / StandardDeviation / Range), and `runtime_describe_binding` (10.1.5 new — pre-auth health and binding probe). Plus any solution-authored Custom Tools.

Step 2: Verify

```
claude mcp list
```

AI Runtime Setup (Split Host / HTTP)

Use this when Claude Code and the running FrameworX TServer are on different machines. Example: an operations console on a laptop connecting to a production TServer on the plant floor.

Prerequisites: same as the Designer Split Host / HTTP section above. Node.js LTS must be installed on the Claude Code host because `mcp-remote` is an npm package.

The Windows machine runs `RuntimeMCPHttp.exe` on default port **10120**. That HTTP server holds the runtime connection to TServer. Claude Code bridges via `mcp-remote`.

Step 1: Configure and start RuntimeMCPHttp on the Runtime machine

Edit `RuntimeMCPHttp.json` alongside the executable to set the target TServer connection (Runtime Host / Port / Username / Password) and the MCP listen port (default **10120**). Start `RuntimeMCPHttp.exe` or install `RuntimeMCPHttpService` as a Windows Service.

Step 2: Add the MCP Server on the Claude Code host

```
claude mcp add FrameworX-Runtime -- npx -y mcp-remote http://127.0.0.1:10120/mcp
```

Replace `127.0.0.1` with the LAN IP of the Runtime machine. From FrameworX 10.1.6, set `X_API_KEY` in `RuntimeMCPHttp.json` and append `--allow-http --header X-API-KEY:<your key>` after the URL. To override the port when `RuntimeMCPHttp.json` has a custom `ListenPort`, substitute in the URL.

Step 3: Verify

```
claude mcp list
```

> The TServer connection is configured on the RuntimeMCPHttp side, not on the Claude Code side. Claude Code only needs to reach the RuntimeMCPHttp port. The HTTP server handles the downstream TServer link.
>
> Inside `RuntimeMCPHttp.json`, the TServer port (default **3101**) is separate from the MCP listen port (default **10120**). Two ports, two roles: MCP bridge = Claude-side, TServer = runtime-side.

---

write_objects — three input shapes

The `write_objects` tool's `data` parameter accepts three shapes (in both AI Designer and AI Console). Pick the one that matches how you are authoring:

Shape	When to use	Example
JSON string literal	Backwards-compatible. Required when the call is built as a string concatenation.	`write_objects(table_type='UnsTags', data='[{"Name":"Foo"}]')`
Parsed JSON array	Single-table without JSON-inside-JSON escaping. Preferred when authoring plan files.	`write_objects(table_type='UnsTags', data=[{"Name":"Foo"}])`
Parsed JSON object	Multi-table without escaping. Preferred for batched plans.	`write_objects(data={"tables":[{"table_type":"UnsTags","data":[...]}]})`

Test Your Setup

AI Designer test: Open Designer, then ask Claude Code: *"Create a new FrameworX solution with a bottling line"* — Claude should call `create_solution` and start building in the live IDE.

AI Console test: Ask Claude Code: *"Create a new FrameworX workspace with 10 motor tags and Modbus communication, then build it"* — Claude should call `create_workspace`, fetch schemas, write JSON files, then call `create_solution_file` to produce the `.dbsln`.

Troubleshooting

Symptom	Likely Cause	Fix
`claude mcp list` doesn't show FrameworX	Add command failed or wrong scope	Re-run `claude mcp add` with `-s user` for global install
DesignerMCP or ConsoleMCP won't start	.NET runtime missing for the version you targeted	Run `dotnet --list-runtimes`. For `fx-10\net8.0\` install .NET 8 from dotnet.microsoft.com/download/dotnet/8.0. For `fx-10\net10.0\` install .NET 10 from dotnet.microsoft.com/download/dotnet/10.0. On Windows on ARM64, the 10.x line must be under a path with no `x64` folder; see **Windows on ARM64** under Prerequisites.
"No active workspace" in AI Console	Workspace not opened	Call `open_workspace` or `create_workspace` first
"Control schemas not found" in AI Console	Missing data file	Ensure `ControlSchemas.json` is in the MCP folder alongside ConsoleMCP.dll
`create_solution_file` or `update_solution_file` returns SOLUTION_CREATOR_NOT_FOUND	Solution Creator.exe not in expected location	Verify FrameworX is installed and the .NET 4.8 binaries are present alongside ConsoleMCP.dll's parent folder
HTTP connection refused (remote)	Firewall or wrong IP	Open the relevant port in Windows Firewall on the server (10120 Runtime, 10150 Designer, 10151 Console) and verify the IP address.
Claude web-searches for FrameworX basics	Skill not loaded	Verify SKILL.md is in `~/.claude/skills/frameworx/` (renamed from the legacy `frameworx-industrial-platform` bundle). To install it, extract `frameworx.skill` from `Documents\FrameworX\AISetup` into `~/.claude/skills/`: the bundle already contains the `frameworx` folder, so extracting it into `~/.claude/skills/frameworx/` nests it one level too deep. On Windows (PowerShell): `tar -xf "$HOME\Documents\FrameworX\AISetup\frameworx.skill" -C "$HOME\.claude\skills"`

Quick Reference: Config Files

File	Location
Claude Code MCP (project-level)	`.mcp.json` in your project root
Claude Code MCP (user-level)	`~/.claude.json`
Claude Code skills	`~/.claude/skills/frameworx/SKILL.md`
AI Console workspace	`Public Documents/FrameworX/Exchange/<WorkspaceName>-json/`

Next Steps

- **Back to main setup:** MCP and Claude Setup
- **Start from a ready workspace:** FrameworX Solution Workspace on GitHub: a CLAUDE.md working agreement, delivery skills and a worked project, ready to clone
- **Session skills:** Claude-Code-Skills on GitHub, installed with `/plugin marketplace add tatsoft-llc/Claude-Code-Skills`
- **Watch:** Claude Builds Industrial Solutions in Minutes (YouTube)
- **See it in action:** AI Designer In Action
- **Technical reference:** AI Designer Connector
- **Download Designer:** FrameworX (free evaluation, unlimited tags, all features)

Community

Questions or feedback:

- Discord: discord.gg/BYhbTfyRyh
- Forum: forum.tatsoft.com
- YouTube: youtube.com/@TATSOFT