AI Designer Best Practices

Best Practices for working with FrameworX AI Designer.

AI Integration Best Practices

AI Designer works best when you treat it as a co-engineer: give it specific context, confirm each step, and correct it when it goes wrong. These five practices cover the areas where new users most often get stuck.

Role, Context, and Instructions
Use FrameworX Terminology
Define what you know. Ask for the rest.
Follow the pillars in order and split large builds across sessions
Run and verify in runtime
When something doesn't work, tell Claude what's wrong
Known limitations

Role, Context, and Instructions

This is the single most reliable way to improve first-attempt results. Before describing your project, set the structure with three parts.

Part	What to include	Example
Role	Tell Claude what it is and what it specializes in.	You are an automation engineer working in FrameworX by Tatsoft, specializing in SCADA systems.
Context	Describe your solution: its name, structure, and purpose.	I have a bottling line solution called BottomLine with areas for filling, washing, and capping.
Instructions	State exactly what to create or change, with all relevant specifics.	Create a device called Filler_PLC1 using Modbus TCP at 192.168.1.10 with tags for run status (Boolean) and line speed (Float).

Once you set the role in the first prompt of a session, you do not need to repeat it. Context and instructions update with each prompt as your build progresses.

See below an example of a starter prompt.

Starter Prompt

You are an automation engineer working in FrameworX by Tatsoft, specializing in SCADA and HMI development.

About me: I am a [automation engineer / system integrator / student — choose one] with [X] years of experience in industrial automation.

Context: I am working on a FrameworX solution called [solution name]. [Add one or two sentences describing the solution and its purpose.]

Work with me like this:

- Build one module at a time following the FrameworX pillars: tags, devices, alarms, historian, scripts, displays. Do not move to the next module until I confirm the current one is working.

- After each step, list the objects you created and their paths so I can verify them in Designer.

- If you are uncertain about a path, driver name, data type, or any other detail, ask me. Do not guess.

- If I tell you something is not working, diagnose and fix it in this session. Do not suggest starting over unless genuinely necessary.

- Do not sugarcoat results. If the first attempt has limitations, tell me what they are and what I need to do manually.

I am ready to describe my first task.

Use FrameworX Terminology

Claude understands general automation terms, but using FrameworX-specific names gives it less to interpret and produces more accurate results.

- Use **device** instead of PLC, controller, or field device.

- Use **UNS** instead of tag database or tag tree.
- Use **display** instead of screen or page.
- Use **historian tag** instead of data logging or trend tag.
- Use **area** instead of folder or group.

Define what you know. Ask for the rest.

You do not need to specify everything upfront. Define the things you know. For everything else, ask Claude to suggest options.

Define these things yourself:

- Protocol and driver name
- Full UNS folder path
- Data type (Boolean, Integer, Float, String)
- Device IP address and port
- Alarm type and setpoint values
- Historian logging interval or deadband

Ask Claude to suggest these things:

- Color schemes and visual layout
- Icon choices and label formatting
- Object positioning and sizing
- Naming conventions for tags and devices
- Tag lists for equipment types you are not familiar with

Example of asking Claude to suggest a tag list:

"I'm building a water pumping station with two centrifugal pumps. I don't have the tag list yet. Suggest the most common tags for this equipment type, including data type and engineering units, and create them in FrameworX under Tag.Plant1/Utilities/PumpStation1."

Follow the pillars in order and split large builds across sessions

FrameworX solutions are built in layers. Each layer depends on what came before. Build in this sequence and confirm each layer works before moving to the next.

#	Pillar	Confirm before moving on
1	**Tags (UNS)**	Tags appear in the asset tree with correct paths, types, and initial values.
2	**Devices and protocols**	Channels and nodes are created. Protocol matches your hardware. Communication is confirmed in runtime.
3	**Alarms**	Alarm items reference the correct tags. Conditions and setpoints are correct. Alarm group is set.
4	**Historian**	Historian tags reference the correct UNS tags. Logging interval or deadband is set as needed.
5	**Scripts**	Scripts compile without errors. Logic runs as expected in runtime.
6	**Displays**	Displays open in runtime. Navigation works. Tag bindings show live values, not tag path strings.

For larger builds, complete pillars 1–5 in one session, then start a fresh session for displays (pillar 6). This gives you a clean context window and prevents running out of tokens mid-build. Re-state your context briefly at the start of the display session so Claude knows the solution structure.

When building multiple displays, include this instruction in your prompt:

"After each display, stop and wait for my review before proceeding to the next."

Run and verify in runtime

Once a layer is built, ask Claude to launch your solution in runtime and verify it visually. FrameworX runs in two clients — the .NET RichClient and the HTML5 WebClient — and Claude can launch either, then inspect the result with you.

Two clients, one solution. Every FrameworX solution can run as a RichClient (the .NET WPF window that opens on Windows desktops) or as a WebClient (HTML5 displays served from a browser). Your solution's RuntimeStartup configuration decides which one launches by default. Claude respects that default unless you ask for a specific client.

What you say	What Claude does
"Run the runtime and check the new dashboard."	Launches whichever client your solution is configured for. Same as clicking "Run Startup" in the Designer Ribbon.
"Test this display in the WebClient."	Forces the HTML5 client. Returns a runtime URL Claude can navigate to in a browser.
"Test this display in the RichClient."	Forces the .NET WPF client.
"Launch both clients."	Starts the RichClient AND opens the WebClient in a browser tab.

None of these change your solution file. They're per-launch overrides — the next time you click "Run Startup" yourself, your solution's configuration applies as before.

Visual verification. When Claude launches the WebClient, it gets back the URL the browser is serving (typically `http://127.0.0.1:<port>/html5/`). If Claude has access to a browser tool, it can navigate to that URL, take a screenshot, read the page, and verify your display visually — without you opening Chrome yourself.



Browser tools available to Claude. Two complementary options:

- **Sandboxed preview** — Claude opens its own isolated browser, navigates, screenshots, captures console logs and network traffic. Good default for runtime smoke tests.
- **Real Chrome** — Claude drives your installed Chrome, sees your logged-in sessions and extensions. Use when you need a realistic browser environment (auth, customer-environment repro).

Both compose with the runtime URL Claude gets back from "test this display in the WebClient". Availability depends on your Claude setup — see the FrameworX MCP setup pages.

Example prompts.

"Launch the WebClient and screenshot the Overview display so I can see how it looks. Walk me through anything that looks off."

"I just changed the alarm color binding. Run the RichClient, navigate to the Alarms display, and tell me whether the inactive alarm rows look right."

"Open the WebClient. Navigate to the trend display and capture the console logs after 10 seconds — I want to see if there are any binding errors."

Iterate fast. Once Claude is launching runtime and inspecting visually, the loop tightens: edit a display, ask Claude to relaunch and screenshot, review, repeat. This is the feedback loop the visual-polish work in the next section assumes.

When something doesn't work, tell Claude what's wrong

AI Designer is conversational. If the first result isn't right, describe the problem. You do not need to restart the session or rebuild from scratch.

Problem	What to tell Claude
The display shows the tag path as text instead of a value.	The tag binding on the display is showing the tag path string instead of the value. Likely cause: the `@` prefix is missing from the `LinkedValue` (a bare `Tag.X` parses as a literal identifier, not a binding), OR a TextBlock has the binding on the `Text` property instead of `LinkedValue` (`Text` is design-time fallback and does not interpolate). For composite-template values, use `{@Tag.X}` inside the curly braces — the `@` is required inside. Apply the four-family rule (see Skill Display Construction - Basics, § Four-family rule for *Link properties).
A navigation button doesn't go to the correct display.	The button isn't navigating correctly. The ActionDynamic is missing or pointing to the wrong display. Fix it.
A script throws a compile error.	The script has a compile error on line 12. The error is: [paste error]. Fix the syntax.
An alarm isn't appearing in the alarm viewer.	The alarm for this tag isn't showing when the value exceeds the setpoint. Check the alarm item configuration and fix it.
Claude created things you didn't ask for.	You created displays and scripts when I only asked for tags. Remove what you added beyond what I specified and wait for my instructions before building the next module.

If a session runs long and responses become degraded, start a new chat, re-state your context briefly, and continue. Most issues resolve in one to two follow-up messages.

Known limitations

AI Designer handles tag creation, device configuration, alarms, historian, scripts, and display structure reliably. Two areas currently require manual work after AI generation.

Area	What to do
Display visual polish	AI-generated displays are structurally correct: elements bind to the right tags, navigation works, alarms display. The visual layout — element sizing, alignment, color consistency — typically needs manual refinement in Designer. Plan for 30 to 60 minutes of polish on a typical display after AI generation.
Image placement	AI does not reliably place images on canvas displays. Place images manually in Designer, then ask Claude to build around them. If you ask AI to modify an image you placed manually, it may corrupt the placement. Make all image changes manually.

Get Started

Setup instructions: MCP and Claude Setup

Overview of all AI features: AI Integration

New Solution Prompts: New Solution Prompts

According to Anthropic's Claude

What Makes Our AI Unique

First-time build walkthrough: Download the AI Designer Guide

In this section...
