Skill Authoring Guide

How to write, test, and publish custom FrameworX AI skills — markdown playbooks that teach the AI assistant your company's specific conventions, workflows, and automation patterns.

Why Write a Skill?

The AI assistant already knows FrameworX tables, protocols, and properties from the built-in documentation. A custom skill adds **your** knowledge on top:

- **Company naming standards** — enforce tag hierarchy, description format, unit conventions
- **Preferred architectures** — your standard alarm group structure, historian setup, device templates
- **Project-specific patterns** — which protocols to use, how to organize folders, compliance requirements
- **Reusable recipes** — common tasks your team repeats across solutions

Without a skill, the AI makes reasonable generic choices. With a skill, it makes **your** choices.

Skill File Format

A skill is a single markdown file (`.md`) with two parts: YAML frontmatter for metadata, and a markdown body with implementation instructions.

Filename Convention

```
skill-<kebab-case-name>.md
```

Examples: `skill-alarm-pipeline.md`, `skill-company-naming.md`, `skill-modbus-setup.md`

The filename becomes the skill's **slug** — a unique identifier used for deduplication across tiers.

YAML Frontmatter

Every skill starts with a YAML block between `---` markers:

Required and optional fields

```
---
title: "Your Skill Title — Descriptive Subtitle"
tags: [keyword1, keyword2, keyword3, module-name]
description: "One-line summary of what this skill teaches the AI to do"
version: "1.0"
author: "Your Company Name"
---
```

Field	Required	Description
`title`	Yes	Human-readable name shown in search results
`tags`	Yes	Comma-separated keywords for discovery — include module names, technologies, concepts
`description`	Yes	One-line summary — the AI reads this when deciding whether to load the full skill
`version`	No	Track changes to your skill over time
`min_product_version`	No	Minimum FrameworX version required (e.g., "11.0")
`author`	No	"Tatsoft" for official skills, your company name for custom

⚠ **Tags matter for discovery.** When the AI calls `search_docs('alarm setup', labels='skill')`, it matches against title, tags, and description. Include synonyms and related module names. For example, an alarm skill should have tags like `alarm, alarms, alert, notification, limit, monitoring`.

Markdown Body Structure

The body follows a recommended structure. Not every section is required — use what's relevant.

Recommended sections

```
# Skill Title

## What This Skill Does          ? one-paragraph summary for the AI
## When to Use This Skill        ? trigger conditions (use/don't use)
## Prerequisites                 ? what must exist before applying
## MCP Tools and Tables Involved ? quick reference table
## Implementation Steps          ? the core: step-by-step with JSON
   ### Step 1: ...
   ### Step 2: ...
## Code Examples                 ? C# scripts if relevant
## Verification                  ? concrete checks
## Common Pitfalls               ? mistakes to avoid (table format)
## Variations                    ? alternative approaches
## Related Skills                ? links to other skills
```

Writing Effective Steps

The Implementation Steps section is the heart of a skill. Each step should be **self-contained** — the AI executes it, verifies, then moves on.

Good step pattern:

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

```
### Step 1: Create Alarm Groups

Groups define severity levels and shared behavior. Create at least Critical, Warning, and Info.

```
get_table_schema('AlarmsGroups')
```

```json
{
  "table_type": "AlarmsGroups",
  "data": [
    { "Name": "Critical", "Description": "Immediate action required" },
    { "Name": "Warning", "Description": "Attention needed" }
  ]
}
```

Key decisions:
- Group names reflect severity, not equipment
- Keep groups to 3-5 maximum
```

What makes this effective:

- Starts with `get_table_schema` so the AI confirms field names before writing
- Provides exact JSON the AI can adapt (not vague prose like "create some groups")
- Explains **why** these choices were made (Key decisions)
- The AI can run this step independently and verify before continuing

Where to Save Your Skill

Location	Scope	Best For
`ProductPaths.Skills/`	Machine-wide (Tier 1)	Company standards that apply to ALL solutions on this machine
Confluence (docs.tatsoft.com)	Online (Tier 3)	Skills shared across the entire organization

Machine-wide (Tier 1) is the simplest option — just save the `.md` file and the AI discovers it within 30 seconds.

The exact path for `ProductPaths.Skills` depends on your installation. It's typically inside the FrameworX library folder. The folder is auto-created on first scan if it doesn't exist.

Testing Your Skill

After saving your skill file, verify it works using the AI assistant. No restart required — local skills have a 30-second cache.

Test 1 — Can the AI find it?

Ask the AI:

> *Search for skills about ‹your topic›*

Behind the scenes, the AI calls `search_docs('your topic', labels='skill')` and your skill should appear in the results.

Test 2 — Can the AI read it?

Ask the AI:

> *Show me what the ‹your skill name› skill does*

The AI calls `search_docs(fetch_url='local://skill/skill-your-name')` and reads the full content.

Test 3 — Does it produce the right output?

Create a test solution and ask the AI to apply your skill:

> *Create a new solution and apply ‹your task description›*

Verify the AI follows your skill's steps and produces the expected objects with your conventions.

Publishing to Confluence (Online Skills)

If you want your skill discoverable by anyone in your organization (Tier 3), publish it as a Confluence page:

1. Create a new page under **AI Skills Library Official Skills** or **User Skill Examples**
2. Paste the wiki markup version of your skill (use **Insert Markup Confluence Wiki**)
3. Add the **`skill`** label to the page — **this is mandatory**
4. Add module labels (e.g., `alarm`, `mqtt`, `script`) for additional filtering
5. Optionally add an `excerpt` macro for search result previews

> Without the `skill` label, the AI cannot discover your page. This is the single most important step when publishing to Confluence.

Overriding Official Skills

If a Tatsoft skill doesn't match your needs, you can override it:

1. Find the official skill's filename (e.g., `skill-alarm-pipeline.md`)
2. Create a file with the **same name** in `ProductPaths.Skills/`
3. Customize the content to match your conventions

Because Tier 1 (user) has higher priority than Tier 2 (cached) and Tier 3 (online), your version wins. The AI will use your customized alarm pipeline instead of Tatsoft's default.

> **Start by copying.** Don't write from scratch — copy the official skill and modify only what's different. This preserves the verification steps and pitfalls that Tatsoft has tested.

Checklist

Before you ship a skill, verify:

1. Title is unambiguous — doesn't sound like a generic doc page
2. Tags include synonyms and module names — at least 5-8 keywords
3. Description is a single sentence — the AI uses it to decide whether to load the skill
4. Every step starts with `get_table_schema` or `list_protocols` — never assume field names
5. JSON examples use realistic values, not placeholder text
6. Verification section has concrete checks (`get_objects`, `get_state`, `browse_object_model`)
7. Every example call passes `solution_id` where the tool requires it — `get_objects`, `write_objects`, `get_state`, `browse_object_model`, `designer_action` and their peers reject the call with `MISSING_SOLUTION_ID` without it. `get_table_schema`, `list_protocols` and `search_docs` do not take it

8. Common Pitfalls captures at least 3 mistakes you've seen in practice
9. If publishing to Confluence: the `skill` label is added to the page

---

Examples

See these published skills as references for different styles and complexity levels:

Skill	Complexity	Good Example Of
Skill New Solution	Beginner	Simple multi-step flow, good for template
Skill Acme Naming Conventions	Beginner	User-created conventions skill, no code
Skill ML.NET Anomaly Detection	Advanced	Scripts, ML pipeline, multi-module integration
Skill MQTT TagProvider	Advanced	Interactive steps, decision guides, dynamic discovery

See also

- Skill Template Reference. Canonical template for creating a new skill page — duplicate and customize.
- Skill Acme Naming Conventions. User-created example: company tag-naming standards as a skill.