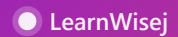


AI Coding Exercise

Build this module's IconDesk sample with ChatGPT or Claude from a ready-made prompt that points the model at this module's specification, then review, run and extend what it gives you.



You are not going to type this sample. You are going to direct it.

Almost nobody writes Wisej.NET code from a blank file any more — an assistant writes the first draft and the developer decides whether it is right. That only works if you know the material, which is what the rest of this module gave you. Here you hand that knowledge to ChatGPT or Claude, get this module's sample back as a working Visual Studio solution, and review it the way you would review a colleague's pull request.

What you are asking for

Create IconDesk.Icons as a class library beside the application, put at least twelve SVG icons in a Resources directory and mark them Embedded Resource. Add a static ApplIcons catalog that exposes each icon as a named resource.wx string so callers never type a path, reference the library from IconDesk, then assign some icons through the designer and others from the catalog in C#. Verify the colour suffix on three icons and write down the SVG cleanup any of them needed. Finish with the naming and versioning rules that decide how a new icon enters the pack.

The module asks for these deliverables, and the prompt below asks the assistant for exactly them:

- ✓ IconDesk.Icons class library with at least twelve SVG files in a Resources directory, all Embedded Resource
- ✓ Static ApplIcons catalog exposing every icon as a named resource.wx string
- ✓ IconDesk page assigning some icons through the designer and others from the ApplIcons catalog
- ✓ Three icons verified with the colour suffix plus a note on the SVG cleanup they needed
- ✓ Naming and versioning rules describing how a new icon enters the pack and how an application overrides one

Why it is shaped this way

By the fifth module IconDesk shows an embedded logo, a theme image and icons from two official packs, and that is when the mess begins: resource strings appear inline in event handlers, two screens spell the same file differently, and nobody can say which icons the product owns. A custom pack is the cheap fix, and it buys a boundary: one assembly holds the artwork, one C# catalog names it, and the rest of the application never types a path again.

These are the terms the assistant has to get right. If what comes back uses them loosely, it has not understood the module:

Term	What it means here
Resource assembly	A plain.NET class library whose only job is to carry icon files as embedded resources; it needs no reference to the application and no Wisej.NET code of its own.
The four-step recipe	Create a class library, create a directory named Resources, put the icon files in it, set their Build Action to Embedded Resource. Everything after that is convenience.
Icon catalog	A static class such as Applcons that exposes each icon as a named constant string, so the compiler and IntelliSense replace copy-and-paste paths.
Semantic name versus file name	The catalog exposes your product's vocabulary — save, delete, customer, warning — while the file keeps a simple name, so renaming it changes one line, not forty screens.
Qualified resource address	An icon in a separate assembly is addressed as resource.wx/ plus the assembly name plus the resource path, which is why every catalog entry shares one private root constant.
Recoloring contract	The?color= suffix only recolors suitable monochrome artwork; an SVG with hard-coded fill values is the usual reason an icon refuses to change color.
Theme sets and overrides	Parallel packages sharing one catalog API over different artwork, plus deployed files that take precedence over an embedded resource, replace icons without touching application code.

Build it — pick your assistant

Both buttons open a new tab with the **same ready-made prompt** already filled in. The prompt points the model at this module's specification — a single Markdown file holding the lesson, the lab brief and the deliverables — so it works from the real material instead of guessing what Module 6 is about. You can open [module-06.md](#) yourself to see exactly what it will read.



Use ChatGPT

Opens [chatgpt.com](#) with this module's prompt loaded. Works with GPT-5 or Codex; any plan that can browse the web will fetch the spec.



Use Claude

Opens [claude.ai](#) with the same prompt loaded. Claude Code users can paste it into a terminal in an empty folder instead.



Send it as-is and you get a project, not a snippet

The prompt ends by asking for the whole solution as a single [.zip](#) with build-and-run steps. Download it, open the solution in Visual Studio, press F5 and IconDesk comes up on [localhost](#). You need the .NET SDK and the [Wisej-4](#) NuGet package; everything else the prompt asks the assistant to write itself.

The prompt, in full — read it, then edit it if you want a different sample

I am working through Module 6 – "Building Your Own Wisej.NET Icon Pack" – of the LearnWisej Icons and Images in Wisej.NET course, and I want to build the module sample with you.

Start by reading the module specification. It is a single Markdown file holding the full lesson, the lab brief and the deliverables:

<https://learnwisej.com/assets/courses/icons-and-images/specs/module-06.md>

Then build a complete Wisej.NET 4 project I can open in Visual Studio:

1. Project "IconDesk", C#, net8.0 or later, NuGet package Wisej-4 4.1.x, Kestrel host calling `app.UseWisej()`.
2. Build what the specification's lab asks for: Create `IconDesk.Icons` as a class library beside the application, put at least twelve SVG icons in a `Resources` directory and mark them `Embedded Resource`. Add a static `AppIcons` catalog that exposes each icon as a named `resource.wx` string so callers never type a path, reference the library from `IconDesk`, then assign some icons through the designer and others from the catalog in C#. Verify the colour suffix on three icons and write down the SVG cleanup any of them needed. Finish with the naming and versioning rules that decide how a new icon enters the pack.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - `IconDesk.Icons` class library with at least twelve SVG files in a `Resources` directory, all `Embedded Resource`
 - Static `AppIcons` catalog exposing every icon as a named `resource.wx` string
 - `IconDesk` page assigning some icons through the designer and others from the `AppIcons` catalog
 - Three icons verified with the colour suffix plus a note on the SVG cleanup they needed
 - Naming and versioning rules describing how a new icon enters the pack and how an application overrides one
4. Follow the specification wherever it names a Wisej.NET class, property or event – use exactly that one. Where it leaves a choice open, tell me which API you picked and why.
5. Keep the server authoritative: state and validation live in .NET, and the browser renders and reports. Never move a decision into JavaScript because it is easier there.
6. Make the result demonstrable: I should be able to run it and see the behaviour this module is about, including what happens when something goes wrong.

Before you write any code, restate in your own words what you are going to build and where each piece will live, so I can correct the design while it is still cheap.

When the code is done, package the whole solution as a single .zip I can download, and give me the exact steps to build and run it.

Finally, answer these questions from the specification, using your own implementation as the evidence:

- Your pack builds and the designer shows nothing in the Image Selector. Which step of the recipe would you check first, and what evidence would tell you the resources really are in the assembly?
- Two of your three test icons recolour and the third stays grey. What is in that SVG, and when would you leave an icon un-recolourable on purpose?
- A customer wants three icons replaced in their deployment only. Which of the override routes would you choose, and what does that choice cost you at the next pack version?

Ask me about anything that is ambiguous before you start.

Then review what came back

Code that compiles is not yet code that is right. The interesting questions are design questions: is this the Wisej.NET control the job actually calls for, does it follow the lifecycle the framework gives you, is the authoritative state on the server rather than parked in the browser because that was convenient? Read the result as a reviewer, not as a customer.

This module gives you the checklist. Answer each one against the code you just received:

- ✓ Your pack builds and the designer shows nothing in the Image Selector. Which step of the recipe would you check first, and what evidence would tell you the resources really are in the assembly?
- ✓ Two of your three test icons recolour and the third stays grey. What is in that SVG, and when would you leave an icon un-recolourable on purpose?
- ✓ A customer wants three icons replaced in their deployment only. Which of the override routes would you choose, and what does that choice cost you at the next pack version?

Anything you had to correct is worth writing down. A short note of the conventions and decisions you settled on, pasted into the first message of your next session, is worth more than any single clever prompt.



No reference sample for this course yet — yours is the first

The [LearnWisej samples repository](#) carries worked builds for most courses, but not yet for this one. That means there is nothing to compare against and no shortcut: the specification and your own review are the whole safety net. Keep what you build — and if it turns out well, we would like to see it.

Three prompts to take it further

Point your assistant at the project you just built and try these. Each one is a review-and-extend exercise, not a rewrite.

- ✓ **Prove it actually works.** *"Go through this project deliverable by deliverable against the module specification at <https://learnwisej.com/assets/courses/icons-and-images/specs/module-06.md> and show me the evidence that each one is genuinely done — the code path, what to click, and what I should see. Where a deliverable is only partly met, say so plainly instead of arguing it through. Then add whatever is missing."*
- ✓ **Review it like a pull request.** *"Walk me through this IconDesk code as if you were reviewing a colleague's pull request and did not write it. Where does state live, what happens end to end on one user action, what would you send back for changes, and which parts would you not be comfortable putting in front of a customer? Be specific and quote the lines you mean — do not summarise the design back to me."*
- ✓ **Swap a piece without moving the seams.** *"Pick the part of this IconDesk project that is most obviously a stand-in for something real — a fake data source, an in-memory store, a simplified service — and replace it with a production-shaped implementation. The contract either side of it must not change: same types, same calls, same behaviour under failure. Tell me what the real implementation forced you to handle that the stand-in did not."*



Carry it into the lab

The hands-on lab that closes this module asks for the same deliverables you just had the assistant produce. Use what you have here as the shape of your answer and the evidence behind it — not as the answer itself. The lab is graded on your reasoning, and that part has to be yours.