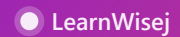


AI Coding Exercise

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 the IconDesk project and give it a command panel with Open, Save, Print and Delete buttons, each carrying its icon through a different mechanism. Load `btnOpen` from a server-side `System.Drawing.Image`, assign a named theme image to `btnSave.ImageSource`, and point `btnPrint` and `btnDelete` at one shared `ImageList` through `ImageKey`. Add a diagnostics label that inspects each button through `IImage` and reports which mechanism is actually in play, then change the image stored under a key in the `ImageList` and confirm that every control using that key follows the change.

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

- ✓ IconDesk command panel with Open, Save, Print and Delete buttons
- ✓ `btnOpen` carrying a `System.Drawing.Image` loaded on the server and `btnSave` a named theme image on `ImageSource`
- ✓ `btnPrint` and `btnDelete` sharing one `ImageList` through `ImageKey` rather than `ImageIndex`
- ✓ Diagnostics label that reads each button through `IImage` and names the mechanism behind its icon
- ✓ Lab note recording what happened when one `ImageList` key was replaced and when two image properties were set on the same button

Why it is shaped this way

Wisej.NET deliberately offers more than one way to put a graphic on a control, and every one of them works. The trouble starts when a team uses all of them by accident: a toolbar where one button reads a file from the server disk, the next points at a theme name, and a third quietly lost its icon because two properties were set on the same control. Nobody can explain, six months later, where a given picture actually comes from or why it disappeared after deployment.

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
Image property	A System.Drawing.Image object that lives in server memory; Wisej.NET sends it down to the browser as PNG data, so the picture travels as bytes your code already decoded.
ImageSource string	A string that identifies a resource for the client to resolve: a relative URL, an absolute URL, a named theme image, a resource.wx path or an item inside an icon-pack assembly.
IconSource	The same string idea on controls that own a dedicated icon slot beside their main image, so you check the API of the control you are using instead of assuming every control names the property the same way.
IImage contract	The common image family that image-capable controls expose — Image, ImageAlign, ImageIndex, ImageKey, ImageList and ImageSource — which lets one helper method decorate a Button, a Label or anything else uniformly.
ImageList	A reusable collection of image objects that several controls can point at, addressed by position or by name, and able to establish a common image size for the controls that use it.
ImageKey versus ImageIndex	A key such as "delete" carries meaning and survives reordering; the index 3 silently becomes the wrong picture the day somebody inserts an image above it.
Mutually exclusive assignment	All of these properties feed one visual slot, so assigning one can reset the others — which is why the wrong icon is usually a second assignment, not a broken file.

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 1 is about. You can open [module-01.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 1 – "The Wisej.NET Image Model: Image, ImageSource and ImageList" – 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-01.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 the IconDesk project and give it a command panel with Open, Save, Print and Delete buttons, each carrying its icon through a different mechanism. Load `btnOpen` from a server-side `System.Drawing.Image`, assign a named theme image to `btnSave.ImageSource`, and point `btnPrint` and `btnDelete` at one shared `ImageList` through `ImageKey`. Add a diagnostics label that inspects each button through `IImage` and reports which mechanism is actually in play, then change the image stored under a key in the `ImageList` and confirm that every control using that key follows the change.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - IconDesk command panel with Open, Save, Print and Delete buttons
 - `btnOpen` carrying a `System.Drawing.Image` loaded on the server and `btnSave` a named theme image on `ImageSource`
 - `btnPrint` and `btnDelete` sharing one `ImageList` through `ImageKey` rather than `ImageIndex`
 - Diagnostics label that reads each button through `IImage` and names the mechanism behind its icon
 - Lab note recording what happened when one `ImageList` key was replaced and when two image properties were set on the same button
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 diagnostics label says `btnDelete` uses an image source, but you only ever set its `ImageKey`. What must have run, and where would you look for it?
- Both print and delete changed when you replaced one entry in the `ImageList`. Would that still be true if the buttons used `ImageIndex`, and what would happen if a fifth image were inserted at the top of the list?
- The Open button reads its file on the server. Which part of that arrangement breaks when the application is deployed to a different machine, and which part would not break if the same picture were an image-source string?

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 diagnostics label says btnDelete uses an image source, but you only ever set its ImageKey. What must have run, and where would you look for it?
- ✓ Both print and delete changed when you replaced one entry in the ImageList. Would that still be true if the buttons used ImageIndex, and what would happen if a fifth image were inserted at the top of the list?
- ✓ The Open button reads its file on the server. Which part of that arrangement breaks when the application is deployed to a different machine, and which part would not break if the same picture were an image-source string?

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-01.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.