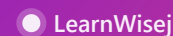


AI Coding Exercise

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.



Same module, built the way you will actually build things.

Every other section of this module explains how "Embedded Resources and resource.wx" works. This one asks you to produce it — by directing an assistant with the module's own specification, then verifying the result runs and does what the lab brief describes. It is the workflow you will use on real work, practised on something whose right answer you already know.

What you are asking for

Add a Resources folder to the IconDesk application holding logo.svg, status-ok.svg, status-warning.svg, photo.png and badge.gif, and set every one of them to Build Action Embedded Resource. Build a ResourceLab page that shows all five through resource.wx image sources, write one of those sources with the application assembly name in front of the file so the lookup cannot be ambiguous, and leave the rest unqualified. Then drop a replacement file next to the deployed application to override one embedded resource and capture the before and after. Finish by deliberately misspelling one resource URL and writing down the steps that identified the failure.

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

- ✓ Resources folder with logo.svg, status-ok.svg, status-warning.svg, photo.png and badge.gif all marked Embedded Resource
- ✓ ResourceLab page displaying every embedded asset through a resource.wx image source
- ✓ One image source qualified with the application assembly name and the rest left unqualified
- ✓ Deployed file that overrides one embedded resource, captured before and after
- ✓ Troubleshooting note listing the steps that identified a misspelled resource URL

Why it is shaped this way

Up to this point every IconDesk asset has been something the application found on disk or asked the theme for. That works until someone deploys the site and forgets a folder: the browser asks for an icon, the server answers 404, and the user sees an empty rectangle where the status badge should be. Nobody catches it in review, because the developer machine still has the file.

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
Build Action = Embedded Resource	The Visual Studio file property that compiles a file into the assembly manifest instead of copying it beside the executable, so the asset ships inside the DLL.
resource.wx handler	The Wisej.NET HTTP handler that serves embedded resources to the browser; what you assign is an image-source string, never a server-side image object.
Unqualified resource URL	resource.wx/company-logo.svg asks Wisej.NET to search the loaded assemblies for a resource with that name, which is convenient and slightly ambiguous.
Assembly-qualified resource URL	resource.wx/MyCompany.Branding/company-logo.svg names the assembly (without.dll) so the lookup cannot land on someone else's file of the same name.
Allowed resource types	The handler documents exactly nine extensions:.css,.js,.html,.jpg,.jpeg,.png,.gif,.svg and.bmp; anything else comes back as a 404.
Deployed-file override	A file deployed with the application can be served in place of the embedded resource at the same path, which lets an installation re-brand or hotfix artwork without rebuilding the assembly.
Manifest resource name	The name.NET actually stores, derived from the assembly name, the default namespace and the folder the file sits in; guessing it is the usual reason a resource URL returns nothing.

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 5 is about. You can open [module-05.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 5 – "Embedded Resources and resource.wx" – 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-05.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: Add a Resources folder to the IconDesk application holding `logo.svg`, `status-ok.svg`, `status-warning.svg`, `photo.png` and `badge.gif`, and set every one of them to Build Action Embedded Resource. Build a ResourceLab page that shows all five through `resource.wx` image sources, write one of those sources with the application assembly name in front of the file so the lookup cannot be ambiguous, and leave the rest unqualified. Then drop a replacement file next to the deployed application to override one embedded resource and capture the before and after. Finish by deliberately misspelling one resource URL and writing down the steps that identified the failure.
3. Produce these deliverables, and tell me where each one lives in the solution:
 - Resources folder with `logo.svg`, `status-ok.svg`, `status-warning.svg`, `photo.png` and `badge.gif` all marked Embedded Resource
 - ResourceLab page displaying every embedded asset through a `resource.wx` image source
 - One image source qualified with the application assembly name and the rest left unqualified
 - Deployed file that overrides one embedded resource, captured before and after
 - Troubleshooting note listing the steps that identified a misspelled resource URL
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:

- Two referenced assemblies both embed a file called `logo.svg`. What does an unqualified source resolve to, and what would you change so the answer stops depending on load order?
- Your override works on the test server but not after the next release. Which parts of the deployment process could have removed it, and where should the override have been recorded?
- A teammate embeds `icon.ico` and reports that `resource.wx` is broken. What is actually happening, and what is the smallest change that fixes it?

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:

- ✓ Two referenced assemblies both embed a file called logo.svg. What does an unqualified source resolve to, and what would you change so the answer stops depending on load order?
- ✓ Your override works on the test server but not after the next release. Which parts of the deployment process could have removed it, and where should the override have been recorded?
- ✓ A teammate embeds icon.ico and reports that resource.wx is broken. What is actually happening, and what is the smallest change that fixes it?

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.

- ✓ **Add the second case.** *"Extend this IconDesk project with a second case that follows the same pattern as the first — another entity, screen or control of the same shape, wired the same way. Factor out what is genuinely shared into a base class or service, but only where it earns its place; two similar things do not always want an abstraction. Show me the diff and justify each thing you extracted."*
- ✓ **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-05.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."*



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.