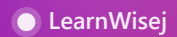


Embedded Resources and resource.wx

Lesson Guide · Lab / Exam Guide



Lesson Guide

Shipping images inside the assembly: Build Action set to Embedded Resource, the resource.wx handler with and without an assembly qualifier, the css, js, html, jpg, jpeg, png, gif, svg and bmp extensions it will serve, and the deployed file that can override an embedded resource without a rebuild. What this means for a developer who knows basic C# and has done some Wisej.NET control and designer work — and how to read this module.



Production focus

Shipping images inside the assembly: Build Action set to Embedded Resource, the resource.wx handler with and without an assembly qualifier, the css, js, html, jpg, jpeg, png, gif, svg and bmp extensions it will serve, and the deployed file that can override an embedded resource without a rebuild.

1. The production problem

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.

Embedding turns the artwork into part of the build output: the picture travels with the assembly, versioned by the same build number, impossible to lose in a copy step. It is also the mechanism every Wisej.NET icon pack is built on, which is what makes Module 6 possible.

2. Key concepts

Concept	Why it matters
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	<code>resource.wx/company-logo.svg</code> asks Wisej.NET to search the loaded assemblies for a resource with that name, which is convenient and slightly ambiguous.
Assembly-qualified resource URL	<code>resource.wx/MyCompany.Branding/company-logo.svg</code> names the assembly (without <code>.dll</code>) so the lookup cannot land on someone else's file of the same name.
Allowed resource types	The handler documents exactly nine extensions: <code>.css</code> , <code>.js</code> , <code>.html</code> , <code>.jpg</code> , <code>.jpeg</code> , <code>.png</code> , <code>.gif</code> , <code>.svg</code> and <code>.bmp</code> ; 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.

3. Lesson narrative

Embedding is a build action, not an API call

There is no method to call and no attribute to decorate. You add the file to the project, open the Properties window, and change Build Action from its default to **Embedded Resource**. The compiler writes the bytes into the assembly manifest, the file stops appearing in the output folder, and from then on the asset is part of the binary. The lab does this five times: `logo.svg`, `status-ok.svg`, `status-warning.svg`, `photo.png` and `badge.gif`, all inside a `Resources` folder in the IconDesk application.

The same property works in a class library, and that is the interesting case. A library whose only job is to carry artwork is a redistributable bag of icons: reference it, and every screen that references it can address those files. Wisej.NET icon packs are exactly this, several hundred SVGs in one assembly, and Module 6 has you build your own. Nothing about the technique changes between an application and a library; only the name you may have to put in the URL does.

Contrast this with the alternatives. A Content file with *Copy to Output Directory* stays loose: easy to replace, easy to lose. An external URL moves the problem to another host. Embed when the asset belongs to the code and should be versioned with it; the decision is operational, not stylistic.

Addressing it: resource.wx with and without an assembly name

Once the file is embedded, the browser reaches it through the `resource.wx` handler. The value you assign to `ImageSource` is a string, and the client resolves it as a URL; the server never materialises a `System.Drawing.Image` for it. That distinction matters most for SVG, which is not a `System.Drawing` format at all: it is markup the browser renders, which is precisely why it stays sharp at 16, 24 or 96 pixels.

```
// IconDesk - ResourceLab.cs
// every file below has Build Action = Embedded Resource

// unqualified: Wisej.NET searches the loaded assemblies
picStatusOk.ImageSource = "resource.wx/status-ok.svg";
picPhoto.ImageSource    = "resource.wx/photo.png";
picBadge.ImageSource     = "resource.wx/badge.gif";

// qualified with the assembly name, no ".dll" - no ambiguity
picLogo.ImageSource      = "resource.wx/IconDesk/logo.svg";

// the same URL works outside control properties, e.g. in CSS:
// .brand-mark { background-image: url('resource.wx/IconDesk/logo.svg'); }
```

Leave the assembly segment out and Wisej.NET searches the loaded assemblies for a matching resource. That is fine while one assembly owns the name. Add a second icon library, or a customer's branding assembly, and two files called `logo.svg` can both answer. Qualifying the path removes the question and documents ownership for the next reader. The lab has you qualify exactly one source so the difference is visible in your own code.

Because the handler speaks plain HTTP, the same URL form can appear in a stylesheet, a fragment of HTML or a piece of JavaScript, not only in an image property. Note too that the `[assembly: WisejResources]` attribute belongs to a broader discovery and bundling story for embedded scripts, styles and themes; a resource URL works with or without it, so do not add attributes mechanically just to show one picture.

What the handler will and will not serve

The documented list is short and worth memorising, because the failure mode is silent. The handler serves `.css`, `.js`, `.html`, `.jpg`, `.jpeg`, `.png`, `.gif`, `.svg` and `.bmp`. Six of the nine are images; the other three exist because a library assembly often carries a stylesheet or a script beside its artwork. Everything else, an icon file, a font, a PDF, comes back as a 404 no matter how correctly it was embedded.

So the extension is a decision taken before the file enters the project: a vector logo becomes `.svg`, a photograph `.png` or `.jpg`, an animated marker `.gif`. If someone hands you a format the handler does not serve, converting it is part of the work.

When a deployed file wins, and how to diagnose a 404

Wisej.NET lets a file deployed with the application take precedence over the embedded resource at the same logical path. This is deliberate and useful: a customer installation can drop in its own `logo.svg` and the application re-brands itself without anyone rebuilding or re-releasing the resource assembly, and the same route hotfixes a wrong icon. The lab has you perform one override and capture the before and after.

The risk is that the deployment directory is now part of your trust boundary: anyone who can write files there can change what the application displays. The second risk is drift, because the running application no longer matches the assembly it was built from. Treat override capability as configuration, with an owner and a log.

When an image does not appear, do not guess at manifest names. Work down a fixed list instead; each step rules out one cause:

- Confirm Build Action is Embedded Resource, not Content or None.
- Confirm the extension is one of the nine the handler serves.
- Confirm the assembly is actually loaded in the running process.
- Confirm the assembly and path segments in the URL, character by character.
- Check whether a default namespace, folder or logical-name change moved the manifest resource name.
- Rebuild, then request the exact URL in the browser and read the status code.



Name the mechanism first

Before you assign any image, say which mechanism you are using: a `System.Drawing.Image` object the server renders as PNG, an image-source string the browser resolves, a named theme image, a resource.wx embedded resource, or an icon font drawn as HTML. Assign one image property per control so the others cannot quietly reset it, keep the source string meaningful instead of a pasted magic path, prefer SVG for interface icons so they stay sharp at every size and scale, and give every icon that carries meaning a text alternative.



Learning objectives

By the end of this module you'll be able to:

- ✓ Embed supported image files in an application or class-library assembly.
- ✓ Use resource.wx with and without an assembly name.
- ✓ List the resource types supported by the Wisej.NET embedded-resource handler.
- ✓ Explain how deployed files can override embedded resources and why that can be useful.

Lab / Exam Guide

What you'll build in the hands-on lab, the suggested approach, and the required deliverables.



Hands-on lab

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.

Starting point

Open IconDesk as it stands after Module 4: the command panel, the ImageLab page, the designer-driven IconGallery and the IconCompare page that puts two official icon packs side by side. Every asset so far arrives from the theme, a loose project file, a URL or a pack assembly; nothing yet ships inside IconDesk's own binary, and there is no **Resources** folder in the project.

Suggested approach

- ✓ Add the **Resources** folder and the five files first, then set Build Action to Embedded Resource on each one and rebuild before writing any C#.
- ✓ Add a **ResourceLab** page with five PictureBox or Label controls and assign each **ImageSource** a **resource.wx** string, qualifying only the logo with the **IconDesk** assembly name.
- ✓ Run the page, open the browser network panel, and confirm each asset returns 200 and the SVGs stay sharp when you enlarge them.
- ✓ Publish or copy the application to a deployment folder, drop a replacement file over one embedded asset, reload without rebuilding, and screenshot both states.
- ✓ Break one URL on purpose, then walk the troubleshooting list in order and record which step revealed the cause.

Required deliverables

- ✓ 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

Student review questions

- ✓ 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?



Name the mechanism first

Before you assign any image, say which mechanism you are using: a `System.Drawing.Image` object the server renders as PNG, an image-source string the browser resolves, a named theme image, a resource.wx embedded resource, or an icon font drawn as HTML. Assign one image property per control so the others cannot quietly reset it, keep the source string meaningful instead of a pasted magic path, prefer SVG for interface icons so they stay sharp at every size and scale, and give every icon that carries meaning a text alternative.