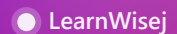


Wisej.NET Icon Packs and NuGet

Lesson Guide · Lab / Exam Guide



Lesson Guide

What an icon pack really is: a .NET assembly carrying hundreds of SVG files as embedded resources, addressed individually through resource.wx and cached by the browser one icon at a time, installed from the official Wisej-4 NuGet packages, browsed in the Visual Studio image explorer and referenced from C# through the pack's icon catalog. 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

What an icon pack really is: a .NET assembly carrying hundreds of SVG files as embedded resources, addressed individually through resource.wx and cached by the browser one icon at a time, installed from the official Wisej-4 NuGet packages, browsed in the Visual Studio image explorer and referenced from C# through the pack's icon catalog.

1. The production problem

Three modules in, IconDesk can already show a picture five different ways, but every icon it owns was drawn or downloaded by hand. A real application needs a Save, a Delete, a Search, a User and a Settings icon on day one, and another forty by the end of the quarter, all in the same visual language. Copying SVG files in one at a time produces a folder of unrelated artwork, inconsistent stroke weights and no licence trail.

2. Key concepts

Concept	Why it matters
icon pack	An ordinary .NET assembly whose icon files, mostly SVG, are compiled in as embedded resources; referencing it adds one file to your deployment no matter how many icons it holds.
embedded resource	A file compiled into the assembly rather than copied beside it, so the icons cannot be lost, renamed or forgotten when the application is deployed to another server.
resource.wx	The Wisej.NET handler that serves one embedded resource over HTTP; an icon-pack image source is a <code>resource.wx</code> URL naming the pack assembly and the file inside it.
icon catalog	The static class a pack publishes so callers get IntelliSense and a compile-time name instead of a typed path; the exact namespace and class name vary from pack to pack.
per-icon request	Pack icons are fetched individually, on first use, and then served from the browser cache, which is what makes a catalog of thousands of icons affordable.
theme image preload	Theme images travel with the theme and are already in the client theme cache when the application starts, so they behave very differently from pack icons at startup.
primary icon family	The one pack a product commits to for its everyday commands, chosen deliberately for coverage, style and licence rather than accumulated pack by pack.

3. Lesson narrative

An icon pack is an assembly, not a folder

The first thing to unlearn is the idea that an icon pack is a zip of pictures. A Wisej.NET icon pack is a normal .NET assembly. Its icons, in the official packs overwhelmingly SVG, are compiled in as embedded resources, so a pack holding several thousand icons still arrives in your `bin` folder as a single DLL. Nothing is copied next to `Default.html`, nothing has to be published with the site, and nothing can be deleted by accident during a deployment.

Each icon nevertheless stays individually addressable. At runtime the browser asks for one icon at a time through the `resource.wx` handler, which reads that one embedded resource out of the referenced assembly and returns it. The image source you assign is therefore just a string: the handler name, the assembly the resource lives in, and the file inside it. Because it is an ordinary HTTP response, the browser caches it like any other file, so the second control that uses the same icon, and the next page load, cost nothing.

That per-icon delivery is why a pack of thousands of icons is affordable: referencing the assembly sends nothing to the browser, using an icon sends one file. A named theme image works the other way round, preloaded with the theme before the first screen paints.

Installing the official Wisej-4 packages

Wisej.NET publishes its icon families as NuGet packages, and for a Wisej.NET 4 application you want the `Wisej-4-*` family: `Wisej-4-FontAwesome`, `Wisej-4-MaterialDesign`, `Wisej-4-ModernUI`, `Wisej-4-`

`ElegantIcons`, `Wisej-4-VisualStudioIcons`, `Wisej-4-BootstrapIcons`, `Wisej-4-VaadinIcons`, `Wisej-4-FeatherIcons`, `Wisej-4-CSSGGIcons`, `Wisej-4-TablerIcons` and `Wisej-4-IonicIcons`. Wisej.NET 4.1 also brings Lucide into the picture; confirm how it is distributed from the official Wisej NuGet profile and the 4.1 release notes rather than installing an unrelated third-party package that happens to share the name.

Add them the way you add any dependency, from the Visual Studio NuGet UI or with `dotnet add package Wisej-4-BootstrapIcons`. Keep the pack's major line aligned with the framework's: a Wisej 4 application uses Wisej-4 packs, and examples copied from an older Wisej 2 or Wisej 3 package should be re-checked against the current one rather than pasted in. In the IconDesk lab you will reference two of these packages, which is exactly two `PackageReference` entries in the project file and two more assemblies in source control's dependency story.

One step catches nearly everyone: build after adding the reference. The design-time image explorer discovers icon-pack assemblies that the project actually references and has compiled against, so a picker opened before the first build will simply not list the new family.

From the image explorer to C#

With the packages in place, the Visual Studio image selector gains the new families alongside your project images and the theme images, and you browse and preview icons there instead of guessing filenames. When you pick one, the designer writes the image source into the generated `Designer.cs` for you, and that generated string is worth reading at least once because it shows the whole mechanism in one line.

```
// designer-generated: the picker wrote the resource.wx URL for you
buttonSave.ImageSource =
    "resource.wx/Wisej.Ext.FontAwesome/android.svg?color=Coral";

// from C#: the pack's own icon catalog, so no path is typed by hand
// (the exact catalog namespace and class vary from pack to pack)
buttonSave.ImageSource = FontAwesome.Icons.Save;

// recolour a monochrome pack SVG with a theme colour name
buttonDelete.ImageSource =
    "resource.wx/Wisej.Ext.BootstrapIcons/trash.svg?color=error";
```

Both forms end up at the same place. The catalog is the one to prefer in business code: it gives IntelliSense, it survives a rename inside the pack, and it stops literal filenames from being scattered across handlers where nobody can audit them. The `?color=` suffix works on pack SVGs exactly as it did on your own monochrome SVGs in the previous module, and it works for the same reason, so a filled or multi-colour piece of artwork will ignore it. That is a fact to discover in the lab on two real icons, not an assumption to carry into production.

Pack, theme image, or your own pack

Three mechanisms now compete for the same button. Use a **named theme image** when the icon represents a standard framework semantic and should change when the theme changes. Use a **published icon pack** when you need broad coverage of ordinary application concepts. Build an **application icon pack**, which is Module 6, when the symbols are proprietary, branded or a curated subset you want to

version yourself. The mechanisms coexist happily; what breaks a product is using all three at random for the same kind of command.

Choosing the primary family is a short, answerable review:

- Does the pack cover the business concepts the product actually names?
- Do its outline or fill styles sit comfortably next to the theme?
- Are the icons monochrome SVGs that accept recolouring?
- Does the upstream licence fit how the application is distributed?
- Will designers and developers recognise the naming convention?

Finally, treat the pack like the dependency it is. It is an assembly you ship, so record its source and licence, keep the `PackageReference` in source control, and test a version change before production. Do not install every family because installing is easy. The lab closes with a one-paragraph recommendation for exactly that reason: the network panel evidence, one request per icon and then a cache hit, is what makes it defensible.



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:

- ✓ Explain what a Wisej.NET icon pack is and how the browser obtains its icons.
- ✓ Install and use official Wisej-4 icon-pack NuGet packages.
- ✓ Use icon-pack resources in the Visual Studio designer and from C#.
- ✓ Compare icon-pack resources with theme images and select the right mechanism for an application.

Lab / Exam Guide

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



Hands-on lab

Add two official Wisej-4 icon-pack NuGet packages to IconDesk and build an IconCompare page that shows the same five concepts, Save, Delete, Search, User and Settings, drawn from both families so the visual difference is impossible to miss. Pick at least five of the icons through the Visual Studio image explorer rather than typing the path, assign the rest from the pack catalog in C#, recolor two of them with the `?color=` suffix, and watch the network panel to see icon files arriving one request at a time and then coming from the browser cache. Close with a short recommendation naming the primary icon family for the project.

Starting point

Open the IconDesk solution as you left it after Module 3: the command panel, the ImageLab page and the IconGallery page all work, and every image in the project is either a local file, a theme image or a URL you typed yourself. The project has no icon-pack reference yet, so the designer's image selector currently shows only project and theme images.

Suggested approach

- ✓ Pick two contrasting official Wisej-4 families, add both with the NuGet UI or `dotnet add package`, and build the project before you open any picker.
- ✓ Add an `IconCompare` page with two labelled columns and one row per concept: Save, Delete, Search, User and Settings.
- ✓ Fill the first column from the Visual Studio image explorer, then open `IconCompare.Designer.cs` and read the `resource.wx` strings the picker generated.
- ✓ Fill the second column in C# from each pack's icon catalog, and recolour two icons by appending the `?color=` suffix with a theme colour name.
- ✓ Reload with the browser network panel open, record the per-icon requests and the cache hits on the second load, then write the recommendation from that evidence.

Required deliverables

- ✓ Two official Wisej-4 icon-pack NuGet packages referenced by IconDesk
- ✓ IconCompare page showing Save, Delete, Search, User and Settings from both packs
- ✓ At least five icons chosen through the Visual Studio image explorer rather than typed by hand
- ✓ Two icons recoloured with the colour suffix and a note on which artwork accepted the recolour
- ✓ One-paragraph recommendation naming the project's primary icon family, with the browser-cache evidence behind it

Student review questions

- ✓ Referencing a pack of three thousand icons adds one assembly to the deployment. What does that cost the browser, and when?
- ✓ One of your two recoloured icons ignored the colour suffix. What is it about that artwork, and what would you change in the pack selection criteria because of it?
- ✓ A teammate wants every command icon to follow the theme when the application switches themes. Which of the five IconCompare icons should become theme images instead, and which should not?



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.