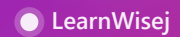


The Wisej.NET Image Model: Image, ImageSource and ImageList

Lesson Guide · Lab / Exam Guide



Lesson Guide

The two image pipelines: an Image property holding a System.Drawing.Image that the server sends to the browser as PNG, versus an ImageSource or IconSource string the client resolves as a URL, a theme name, a resource.wx path or an icon-pack item, plus the shared IImage contract and the ImageList that lets many controls share one keyed set of images. 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

The two image pipelines: an Image property holding a System.Drawing.Image that the server sends to the browser as PNG, versus an ImageSource or IconSource string the client resolves as a URL, a theme name, a resource.wx path or an icon-pack item, plus the shared IImage contract and the ImageList that lets many controls share one keyed set of images.

1. The production problem

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.

2. Key concepts

Concept	Why it matters
Image property	A <code>System.Drawing.Image</code> 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 <code>resource.wx</code> 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 — <code>Image</code> , <code>ImageAlign</code> , <code>ImageIndex</code> , <code>ImageKey</code> , <code>ImageList</code> and <code>ImageSource</code> — 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 <code>"delete"</code> carries meaning and survives reordering; the index <code>3</code> 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.

3. Lesson narrative

Object or string: the first question to answer

Before you type anything, decide whether the server owns an image *object* or whether the browser should resolve an image *identifier*. Those are the two pipelines, and almost every later confusion in this course comes from mixing them up. When you assign `Image`, you hand Wisej.NET a `System.Drawing.Image`: real decoded pixel data sitting in your session's memory, which the framework then sends to the browser as PNG. That is the right route when your C# genuinely produced or received an image object — a chart you rendered, a thumbnail you resized, a photo that arrived from a service as a stream.

When you assign `ImageSource` or `IconSource`, you hand Wisej.NET a string instead. Nothing is decoded on the server. The string names something the client can fetch and cache on its own: `"Images/open.png"` relative to the application, an absolute address, a theme image name such as `"icon-save"`, or a `resource.wx` path into an assembly. Because the resource keeps its identity, an SVG stays an SVG all the way to the browser, which is why interface icons stay crisp at any size and can later be recoloured. In IconDesk you will see both: `btnOpen` receives an image object the server loaded, `btnSave` receives a theme name as a string, and the diagnostics label has to tell them apart.

One family of properties, named IImage

Wisej.NET does not give each control its own vocabulary for pictures. Image-capable controls share a contract called `IImage` that gathers `Image`, `ImageAlign`, `ImageIndex`, `ImageKey`, `ImageList` and

`ImageSource` into one family. Recognising the family is more valuable to a beginner than casting to it: when you meet a new control, you already know which properties to look for and which four of them compete for the same slot.

The cast is still useful for infrastructure code. A method that accepts a `Control`, tests `is IImage` and then sets `ImageSource` can apply the project's save icon to a Button, a Label or a menu item without a switch statement. The IconDesk diagnostics label in the lab does the reverse: it reads each button through `IImage`, checks which of the four properties is populated, and prints the mechanism behind the icon rather than guessing from the picture.

Picking the mechanism for the asset in front of you

Four cases cover nearly everything. A file deployed beside the application is an image source with a relative URL. An image hosted somewhere else is an image source with an absolute URL. An icon the active theme already defines is an image source holding the theme's name for it, which means the client takes it from the theme cache it has loaded anyway. A picture your code created, transformed or received as a .NET object is an `Image`.

The fifth case is the shared one, and it is what `ImageList` exists for. An `ImageList` holds a collection of image objects; controls reference an entry by numeric `ImageIndex` or by string `ImageKey`. Two properties of that indirection matter in a real application: many controls can share one collection, and the picture stored under a key can be replaced without touching a single control that refers to that key. The list can also establish an image size for the controls that use it, which is handy when the theme would otherwise decide the icon size for you.

```
var icons = new ImageList();
icons.Images.Add("print", Image.FromFile("Images/print.png"));
icons.Images.Add("delete", Image.FromFile("Images/delete.png"));

btnPrint.ImageList = icons;
btnPrint.ImageKey = "print";
btnDelete.ImageList = icons;
btnDelete.ImageKey = "delete";

// btnSave takes its icon from a string instead - no ImageList involved.
btnSave.ImageSource = "icon-save";
```

Note what the first lines are *not* doing. `Image.FromFile` reads a file on the *server*, never on the user's machine, so an absolute path that worked on your laptop is a deployment defect waiting to happen. An image-source string, by contrast, is addressed the way the browser addresses everything else.

Why the icon you set is not always the icon you see

A control has one visual slot for its picture, and `Image`, `ImageSource`, `ImageIndex` and `ImageKey` all compete to fill it. Wisej.NET resolves that by making them mutually exclusive: setting one can reset the others, so the configuration is never ambiguous. If `btnDelete` is happily showing the list entry under `ImageKey = "delete"` and a later line sets `ImageSource`, the string becomes the active representation and the key stops mattering.

That single rule explains most "the wrong icon is showing" bug reports. The designer set one property, the form constructor set another, and a theme or a helper method set a third. When the picture is wrong, do not start by opening the file — inspect every image-related property on the control and find the last assignment that ran.

- One image property per control, chosen deliberately.
- Keys, not indices, wherever the identity has meaning.
- A text alternative for any icon that carries meaning on its own.



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 the difference between a `System.Drawing.Image` value and an image string such as `ImageSource` or `IconSource`.
- ✓ Use the common `Image` model to recognize `Image`, `ImageSource`, `ImageList`, `ImageIndex` and `ImageKey` properties.
- ✓ Choose the correct image mechanism for a local bitmap, a URL, a theme image or a reusable icon.
- ✓ Understand why assigning one of several mutually exclusive image properties can reset the others.

Lab / Exam Guide

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



Hands-on lab

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.

Starting point

This is the first module, so you start from nothing: create a new Wisej.NET 4 application named `IconDesk` and add a page for the command panel. Drop a small folder of images beside the application — an open, save, print and delete graphic is enough — and confirm the empty project builds and opens in the browser before you assign a single icon.

Suggested approach

- ✓ Lay out the four buttons first with text only, so the panel is readable before any picture exists.
- ✓ Give `btnOpen` a `System.Drawing.Image` the server loads, and `btnSave` a theme image name on `ImageSource`; say out loud which pipeline each one uses.
- ✓ Build one `ImageList`, add the print and delete images under meaningful keys, then point both buttons at the list through `ImageKey` rather than `ImageIndex`.
- ✓ Write the diagnostics label: walk the panel's controls, test each for `IImage`, and report which of `Image`, `ImageSource`, `ImageList` plus `ImageKey` is actually populated.
- ✓ Replace the picture stored under the delete key, refresh the page, and then deliberately set a second image property on one button so you can record what the first assignment did.

Required deliverables

- ✓ 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

Student review questions

- ✓ 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?



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.