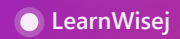


Bitmaps, SVGs and PictureBox: Formats, Sizing and Loading

Lesson Guide · Lab / Exam Guide



Lesson Guide

Two format lists because there are two pipelines: bitmap, metafile, icon, JPEG, GIF and PNG through System.Drawing and PictureBox.Image, against the jpg, jpeg, png, gif, svg and bmp that resource.wx and theme images accept as strings, together with the PictureBox size modes and the synchronous Load and asynchronous LoadAsync entry points. 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

Two format lists because there are two pipelines: bitmap, metafile, icon, JPEG, GIF and PNG through System.Drawing and PictureBox.Image, against the jpg, jpeg, png, gif, svg and bmp that resource.wx and theme images accept as strings, together with the PictureBox size modes and the synchronous Load and asynchronous LoadAsync entry points.

1. The production problem

"Which image formats does Wisej.NET support?" is the question every team asks in week one, and the honest answer depends on which API path the picture takes. Answer it with one vague list and you get a stretched logo on a high-DPI laptop, a page that freezes while a product photo downloads from a slow host, and a bug report because an `.ico` file will not resolve through a resource string. This module replaces the vague list with two precise ones, each attached to the control property that consumes it.

2. Key concepts

Concept	Why it matters
System.Drawing pipeline	The <code>Image</code> property holds a <code>System.Drawing.Image</code> in server memory; Wisej.NET sends it to the browser as PNG data. Its documented sources are bitmap, metafile, icon, JPEG, GIF and PNG files.
Image-source pipeline	A string the client resolves itself: a relative or absolute URL, a named theme image, or a <code>resource.wx</code> path. The documented <code>resource.wx</code> image extensions are .jpg, .jpeg, .png, .gif, .svg and .bmp.
PictureBox.Image	The entry point for pixels your C# code already owns or has just generated, such as a chart drawn with <code>Graphics</code> .
PictureBox.ImageSource	The entry point for a picture the browser can fetch itself. A short string travels instead of image bytes, which is why SVG belongs here.
PictureBox.SizeMode	The six values <code>Normal</code> , <code>AutoSize</code> , <code>CenterImage</code> , <code>StretchImage</code> , <code>Zoom</code> and <code>Cover</code> , which decide whether a picture is clipped, centred, distorted, fitted or cropped.
Load and LoadAsync	<code>Load(url)</code> updates the picture from a URL synchronously; <code>LoadAsync(url)</code> performs the same fetch asynchronously, so a slow host does not hold the screen.
Vector versus raster	An SVG is markup drawn at whatever size the layout asks for, and suitable monochrome artwork can be recoloured; a raster file is a fixed grid of pixels that only ever gets resampled.

3. Lesson narrative

Two format lists, because there are two pipelines

The first objective of this module is to stop quoting one list. For `PictureBox`, the Wisej.NET control documentation describes graphics loaded from bitmap, metafile, icon, JPEG, GIF and PNG files. That is the familiar `System.Drawing.Image` world: you hand the control an object, and the server turns it into PNG data on the wire. For string resources, the documented `resource.wx` image extensions are .jpg, .jpeg, .png, .gif, .svg and .bmp, and theme image definitions support png, gif, svg, jpg and bmp.

The interesting entries are the ones that appear in only one list. Icon and metafile belong to the System.Drawing side and are not part of the `resource.wx` allowed-image list. SVG is the reverse: it is an excellent image-source value and a first-class theme image, but it is not a `System.Drawing.Image` format in the same sense as a Bitmap or a PNG. Saying "Wisej.NET supports SVG" without naming the pipeline is what produces a failed `Image` assignment later in the sprint.

In the lab you write this down explicitly: the format note beside your ImageLab page lists every asset and names the pipeline that carried it.

PictureBox has two entry points, and you choose one per control

The second objective is about picking the right property. `PictureBox.Image` receives a `System.Drawing.Image`. `PictureBox.ImageSource` receives a string. The rule that keeps a project readable is simple: if your C# code is creating or transforming the pixels, use `Image`; if the picture is a file the browser could just as well fetch on its own, use `ImageSource` and keep the string meaningful.

```
pictureBox1.ImageSource = "Images/product.png";    // a string the client resolves
pictureBox1.SizeMode = PictureBoxSizeMode.Zoom;

pictureBox2.ImageSource = "resource.wx/IconDesk/logo.svg";
pictureBox2.SizeMode = PictureBoxSizeMode.Cover;

// a URL the user chose: fetch it without holding the page
pictureBox3.LoadAsync(remoteJpegUrl);
```

The URL entry points sit alongside those properties. `Load(url)` refreshes the picture from a URL synchronously; `LoadAsync(url)` starts an asynchronous load instead. Your ImageLab page uses `LoadAsync` for the remote JPEG precisely so that a slow third-party host degrades into "the picture arrives a moment later" rather than "the page is frozen".

Six size modes, and what each one does to the picture

Wisej.NET documents six `PictureBoxSizeMode` values, and the third objective is to choose between them deliberately instead of accepting whatever the designer left behind. `Normal` draws the image at its original size from the top-left corner and clips whatever does not fit. `AutoSize` inverts the relationship: the PictureBox grows or shrinks to the image. `CenterImage` keeps the original size but centres it, so the clipping happens evenly on all sides. `StretchImage` fills the box whether or not the aspect ratio survives. `Zoom` scales the image with its aspect ratio preserved until the whole picture fits. `Cover` also preserves the aspect ratio but fills the rectangle, cropping the edges that overflow.

In practice, `Zoom` is the safe choice for logos and photos that must remain completely visible, `Cover` is the right one for thumbnails and banners where filling the rectangle matters more than showing every edge, and `StretchImage` should be a decision somebody made on purpose, because it is the mode that quietly distorts a brand mark. The lab shows one raster asset four times, in `Normal`, `Zoom`, `Cover` and `StretchImage`, so that the clipping and the distortion are visible side by side rather than described.

SVG for interface icons, raster for photographs

The fourth objective is the design rule that follows from the two pipelines. SVG is usually the first choice for interface icons: it is vector markup, so it stays sharp at 16, 24, 32 and 48 pixels and at every display scale factor, simple monochrome artwork has a small footprint, and Wisej.NET can recolour a suitable SVG through the image-source string. Photographs and screenshots go the other way. PNG is the general-purpose raster choice when transparency or exact pixels matter, JPEG is normally better for photographs where lossy compression is acceptable, GIF remains useful for simple or animated graphics, and BMP is supported but is usually far larger than a compressed web format.

High-DPI is where the difference stops being theoretical. A vector icon needs no second copy for 150% or 200% scaling; a 16-pixel bitmap enlarged to 32 pixels simply looks enlarged. If you must ship raster icons,

start from enough resolution for the largest size you will draw, and test in the browser at a raised display scale.

- Command, navigation and status glyphs: SVG through `ImageSource` .
- Product photos, screenshots and scans: PNG or JPEG, usually with `Zoom` or `Cover` .
- Pixels your own code draws: a `System.Drawing.Image` on `Image` .

Sizing is a design decision, not an accident

An image resource has intrinsic dimensions, but your interface also has a target icon size, and the two are not the same number. Wisej themes typically control icon metrics, and an `ImageList` can establish an explicit common size and resize the images the controls draw. For a `Label`, the documentation notes that the original image size is used unless an `ImageList` participates.

The last deliverable closes the loop back to Module 1: a report handler walks every `PictureBox` on the `ImageLab` page and states whether its picture arrived through `Image` or through `ImageSource` . That handler is the fastest way to find a control where somebody set both properties and only one of them is winning.



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:

- ✓ Identify the image formats Wisej.NET accepts through `PictureBox`, themes and `resource.wx`.
- ✓ Use `PictureBox.Image` and `PictureBox.ImageSource` correctly.
- ✓ Choose `PictureBox` size modes including `Normal`, `AutoSize`, `CenterImage`, `StretchImage`, `Zoom` and `Cover`.
- ✓ Explain why SVG is usually preferred for scalable UI icons and why photos are usually raster images.

Lab / Exam Guide

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



Hands-on lab

Build an ImageLab page in IconDesk that makes the two pipelines visible side by side. Show one raster asset four times with SizeMode set to Normal, Zoom, Cover and StretchImage so the cropping and distortion differences are obvious, add a PictureBox that pulls a remote JPEG with LoadAsync and keeps the page responsive while it arrives, and place one SVG assigned through ImageSource at 16, 24, 32 and 48 pixels to prove it stays sharp. Finish with a handler that reports, for every PictureBox on the page, whether its picture came from Image or from ImageSource.

Starting point

Open the IconDesk solution you created in Module 1, where the command panel already carries four button icons through four different mechanisms. Add a new page named **ImageLab** to the project and put one raster asset, one photograph-style JPEG that lives on a remote host, and one monochrome SVG icon within reach before you start placing controls.

Suggested approach

- ✓ Lay out four PictureBox controls of identical size on ImageLab and give them the same raster asset, then set **SizeMode** to Normal, Zoom, Cover and StretchImage and label each one with the mode it demonstrates.
- ✓ Add a fifth PictureBox and call **LoadAsync** with the remote JPEG URL from the page's load path, leaving a visible placeholder or caption so the user can see that the page stayed usable while the picture was on its way.
- ✓ Place four more PictureBox controls sized 16, 24, 32 and 48 pixels, assign the same SVG to each through **ImageSource**, and compare them against the raster asset scaled to the same four sizes.
- ✓ Write a **btnReport_Click** handler that walks the page's controls, and for every PictureBox writes one line naming the control and whether its picture came from **Image** or from **ImageSource**.
- ✓ Write the format note: list every asset the page uses, state which travels through System.Drawing and which stays a resource string, and record which size mode you would defend in a review for a logo, a thumbnail and a photograph.

Required deliverables

- ✓ ImageLab page showing one raster asset in Normal, Zoom, Cover and StretchImage size modes
- ✓ PictureBox that fetches a remote JPEG with LoadAsync without blocking the page
- ✓ One SVG assigned through ImageSource and rendered at 16, 24, 32 and 48 pixels
- ✓ Report handler that names, for each PictureBox, whether Image or ImageSource supplied the picture

- ✓ Format note listing which lab assets travel through `System.Drawing` and which stay resource strings

Student review questions

- ✓ Your raster asset is wider than it is tall and the box is square. Which of `Zoom` and `Cover` loses part of the picture, which leaves empty space, and which one would you defend for a customer logo?
- ✓ The remote JPEG host is unreachable for ten seconds. What does the user of your page experience with `LoadAsync`, and what would they have experienced with the synchronous `Load`?
- ✓ Your report handler says a control's picture came from `ImageSource`, but the designer file also assigns `Image` on the same control. Which one is actually drawn, and how would you prove it rather than guess?



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.