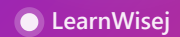


Icon Fonts, HTML Content and Production Icon Strategy

Lesson Guide · Lab / Exam Guide



Lesson Guide

The last mechanism and the policy that binds all of them: an icon-font stylesheet loaded from Default.html, AllowHtml on the controls that should interpret markup, icon glyphs inside DataGridView cells with role-based element clicks, and a production checklist covering accessibility, licensing, caching, external dependencies and visual consistency. 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 last mechanism and the policy that binds all of them: an icon-font stylesheet loaded from Default.html, AllowHtml on the controls that should interpret markup, icon glyphs inside DataGridView cells with role-based element clicks, and a production checklist covering accessibility, licensing, caching, external dependencies and visual consistency.

1. The production problem

After six modules IconDesk can draw an icon five different ways, and every one of them works. What the project still lacks is a decision rule: when a new screen needs a Save symbol, nobody should have to guess whether it comes from the theme, from an official Wisej-4 pack or from IconDesk.Icons. This module adds the last option, icon fonts rendered as HTML, and then writes the policy that says which one to reach for and what to check before a screen ships.

2. Key concepts

Concept	Why it matters
icon font	A web font whose glyphs are pictures instead of letters, chosen with CSS class names and drawn by the browser inside HTML content rather than through any image property.
AllowHtml	The per-control switch that tells Wisej.NET to interpret a control's text as markup; without it the tags appear as literal characters, because text is plain text by default.
Default.html head	The application's host page, where one stylesheet link makes the icon font's CSS and font files available to every screen the session opens.
role-based element click	A click inside HTML content can report which inline element was hit, so one grid cell can offer an Edit target and a Delete target without two separate controls.
CDN versus self-hosting	Pointing the link at a public CDN is a production decision about availability, privacy, Content Security Policy and licensing; internal applications often host the files themselves.
layered icon policy	A written rule giving every kind of symbol one home: theme images for semantics, one primary pack for commands, the custom pack for product artwork, raster files for photos.
icon QA pass	A walk through the finished screens in a light and a dark theme at more than one display scale, checking that each icon is still visible and still has a text alternative.

3. Lesson narrative

Loading the stylesheet and turning markup on

An icon font is not part of the Wisej.NET image system at all. Nothing is assigned to `Image`, nothing is written into `ImageSource`, and the server never produces a PNG. Instead the browser downloads a stylesheet and a font file, and a class name on an inline element tells the font which glyph to draw. That makes loading a page concern rather than a control concern: the stylesheet link belongs in the head of the application's `Default.html`, where it is fetched once for the session and applies to every window and dialog the user opens.

Where that link points is a production decision. A public CDN is convenient, but the interface then depends on a third party being reachable, it reveals your users' requests to that host, it has to pass your Content Security Policy, and it is governed by the icon library's licence. For an internal application, serving the CSS and font files yourself is often the better call: availability and privacy then depend only on your own deployment.

With the stylesheet loaded, nothing appears yet. Wisej.NET controls treat `Text` as plain text, so the markup arrives escaped and the user sees angle brackets. Setting `AllowHtml` to `true` is what changes that, and it is deliberately per control: Label, Button, DataGridView cells, CheckBox, RadioButton and LinkLabel are the HTML-capable examples. Turning the switch on also moves a trust boundary: from that moment the control's text is markup, so keep it application-owned and put the user's words in a separate plain-text control.

Glyphs in a toolbar, a card and a grid cell

In IconDesk the first icon-font surface is a small toolbar or summary card. Each button or label gets `AllowHtml = true` and a text value that puts an inline element in front of a word, which is where icon fonts feel natural: the glyph flows with the label instead of sitting in a separate image slot, and it inherits the surrounding font size and colour. The same idea scales into a `DataGridView` whose `AllowHtml` is on, where a compact action column holds two glyphs in one cell instead of two hosted controls.

A single cell with two symbols only works if the application can tell which one was clicked. That is what a `role` attribute on the inline element is for: the click reports the element that was hit, and the handler branches on it. Inspect the mouse-event data your framework version actually exposes before you rely on it, and do not invent a contract the event does not offer.

```
labelActions.AllowHtml = true;
labelActions.Text =
    "<i class='fa-solid fa-pen' role='edit'></i> " +
    "<i class='fa-solid fa-trash' role='delete'></i>";

labelActions.MouseClick += labelActions_MouseClick;
```

Compactness has a cost. A cell full of markup is harder to reach with the keyboard than a real button, screen readers get a glyph with no name, and an icon that fails to load leaves an empty column. Give each action a tooltip or a visible word, and reach for an ordinary image column or a hosted control when the interaction is more than a two-icon shortcut.

Icon fonts next to SVG and icon packs

Neither mechanism wins outright. Wisej.NET's own system is resource and SVG based, and that is where it is strongest: a source string can name a theme image, a `resource.wx` path or a pack asset; monochrome SVGs accept the `?color=` suffix with a theme colour name; the designer's image selector browses every installed pack; caching is built around those resources; and any control with an image property can use them directly.

Icon fonts earn their place elsewhere. They are familiar to web developers, they drop into an existing HTML and CSS design system, their catalogues are enormous, and they sit inline with text in a way an image property cannot. If the company's design language is already expressed as font classes, matching it costs one stylesheet link instead of redrawing every asset.

So the honest answer is that they coexist, and the choice belongs to the surface rather than the project. Commands on controls with image properties belong to the resource pipeline; inline symbols inside text or a dense grid cell are a reasonable icon-font case. What is not acceptable is drift, where the same Delete concept is a pack SVG on one screen and a font glyph on the next.

The production checklist and the IconDesk policy

The capstone's real deliverable is one page of policy. A layered rule works well: theme images for common semantics, one primary official Wisej-4 pack for broad commands, the IconDesk.Icons custom pack for proprietary artwork, compressed raster files for photographs, and icon fonts only where the HTML

integration is worth it. Because the policy names a first choice for each situation, a new screen is consistent by default rather than by review.

Around that sits the checklist the project runs before a screen ships:

- Every meaningful icon has a visible word, a tooltip or another text alternative, and status is never signalled by colour alone.
- Any external URL or CDN reference is explicitly approved, and the library's licence terms are recorded.
- Icon-font markup is application-owned or encoded; untrusted values never reach an `AllowHtml` control.
- Asset URLs are stable so the browser cache can work, and no random query string defeats it.
- The screen has been opened in a light and a dark theme and at a higher display scale, and the findings are written down.

That QA pass is where the five mechanisms finally stand next to each other. On the IconDesk summary page a theme image, an official pack icon, a custom pack icon, an embedded asset reached through `resource.wx` and a recoloured SVG all appear at once, and switching the theme shows which followed it and which stayed exactly as drawn. Explaining, for every visible picture, where it is stored and which API resolved it is the skill this course has been building towards.



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:

- ✓ Load an icon-font stylesheet and render icons in Wisej.NET controls with `AllowHtml`.
- ✓ Use icon fonts in Label, Button and DataGridView content and understand role-based element clicks.
- ✓ Compare icon fonts with Wisej.NET SVG/icon-pack resources.
- ✓ Apply a production checklist covering accessibility, security, caching, external dependencies and consistency.

Lab / Exam Guide

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



Hands-on lab

Finish the IconDesk capstone. Reference an icon-font stylesheet from `Default.html`, set `AllowHtml` on a small toolbar or card and render its icons as font glyphs, then build a `DataGridView` column whose cells hold icon-font actions and read which element the user clicked so `Edit` and `Delete` do different things. Build a summary page that demonstrates every mechanism the course covered at once: a theme image, an official pack icon, a custom pack icon, an embedded logo and a recoloured SVG. Then write the project's production icon policy and run a QA pass in a light and a dark theme.

Starting point

Open the IconDesk solution as Module 6 left it: the command panel, the ImageLab and IconGallery pages, two official Wisej-4 icon packs, the embedded Resources folder served through `resource.wx`, and the IconDesk.Icons class library with its `AppIcons` catalog. Nothing in the project uses an icon font yet, and the project's `Default.html` still has an empty head apart from the title.

Suggested approach

- ✓ Decide first whether the icon font is referenced from a CDN or hosted beside the application, and record the availability, privacy, Content Security Policy and licence reasons for that choice before you add the link to `Default.html`.
- ✓ Add a small toolbar or card, set `AllowHtml` on each Label or Button, and confirm that a control without the switch shows the markup as literal text so you have seen both behaviours.
- ✓ Turn on `AllowHtml` for the grid, build the action column with two inline elements carrying distinct roles, then inspect the mouse-event data your Wisej.NET version exposes and branch `Edit` and `Delete` on the element that was hit.
- ✓ Build the summary page so each of the five mechanisms appears once, labelled with where the picture is stored and which API resolves it, and keep one control per image property.
- ✓ Run the QA pass: switch between a light and a dark theme, try a higher display scale, check tooltips and keyboard reach on the icon-only actions, and note anything that changed or disappeared.

Required deliverables

- ✓ Icon-font stylesheet referenced from `Default.html` and one `AllowHtml` toolbar or card rendering font glyphs
- ✓ `DataGridView` column whose cells render icon-font actions and respond to the clicked element
- ✓ IconDesk summary page demonstrating a theme image, an official pack icon, a custom pack icon, an embedded asset and a recoloured SVG
- ✓ Written production icon policy covering accessibility, licensing, caching and external dependencies

- ✓ QA pass run in a light and a dark theme with the findings recorded

Student review questions

- ✓ The stylesheet link is in `Default.html` and the class names are right, but one Label shows the raw tags while the Button next to it shows the glyph. What differs between the two controls, and why is the setting per control rather than per application?
- ✓ Your action column works, and a colleague asks to put the customer's own note text into the same cell so it can be styled. What do you tell them, and what would you build instead?
- ✓ On the summary page, which of the five images change when the application switches theme and which stay exactly as drawn? What does that tell you about where each one is stored?



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.