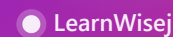


Custom Wisej.NET Controls, Client Classes, and Theming

Lesson Guide · Lab / Exam Guide



Lesson Guide

Build native-feeling controls with server classes, client widget classes, appearance keys, and embedded resources. What this means for a Wisej.NET integration developer — and how to read this module.



Production focus

Build native-feeling controls with server classes, client widget classes, appearance keys, and embedded resources.

1. The integration problem

Some integrations must behave like first-class Wisej.NET controls: reusable, themeable, designable, and maintainable across many applications.

2. Key concepts

Concept	How to use it
Control class	Use a server-side control when native-feeling integration is required.
AppearanceKey	Connect control visuals to theme-level styling.

3. Production lesson

A custom Wisej.NET control is the strongest form of integration. It should be used when the widget is part of the product platform, when theming matters, or when the control must look and behave like a native Wisej.NET control. The pattern is similar to creating a built-in control: a server class exposes .NET properties and events, and a client class receives rendered state and owns the browser-side object. The client class can extend `wisej.web.Control` or `wisej.web.Widget` depending on the desired behavior. It can create the third-party object, implement property apply methods, handle resize or appearance changes, and translate vendor callbacks into Wisej.NET events. The server class can override `OnWebRender` to send only the properties the client needs. This approach requires more code, but it gives the cleanest long-term architecture. It is the right choice for controls shipped as part of a framework, extension library, or enterprise product, where the wrapper must show up in IntelliSense, work in the designer, and stay stable across releases of the underlying vendor library.

A server control for tighter integration

Deriving from `Wisej.Web.Control` instead of `Wisej.Web.Widget` changes what the class is. A Widget is a host for a script; a Control is a first-class member of the framework, with the same layout, focus, theme, and Designer behavior as a built-in button or panel. The lab's `SimpleGaugeControl` takes this route so that the gauge can be dragged, sized, anchored, and themed like any other control.

The server class exposes typed properties and .NET events, just as the wrapper did, but it no longer carries an `InitScript` or an `Options` bag. Instead it declares which client class implements it and renders its state through `OnWebRender`. Property setters mark the control as needing an update, and the framework decides when to send the change.

Choose this depth when the widget is part of the product platform, when theming matters, or when the control must feel native to the people who build screens with it. The extra code is paid back by every screen that uses the control without thinking about it.

A client class under Platform

The browser half of a custom control is a JavaScript class defined with the Wisej.NET client class model, which is built on qooxdoo, and placed under the application's `/Platform` folder so the framework loads it with the client. The class extends `wisej.web.Control`, or `wisej.web.Widget` when script-hosting behavior is wanted, and declares its properties with apply methods.

An apply method runs whenever the server changes the matching property, which is where the vendor call belongs: the apply for `value` calls the vendor's set value method, and the apply for `maximum` updates the vendor's options. The class constructor creates the vendor instance inside the control's content element, and the destructor destroys it. Resize and appearance changes arrive as events the class can handle in one place.

Keep the client class free of application logic. It translates between the framework and the vendor, and nothing else, so that it can be reused by every server control that renders to it.

Rendering state with OnWebRender

`OnWebRender` is the server's chance to decide exactly what the client sees. The override receives a dynamic config object, and the control writes onto it only the properties the client class declares: `config.value`, `config.minimum`, `config.maximum`, `config.caption`. Names are camel-cased to match the client property names, and Wisej.NET sends only the values that changed since the last render.

This is intentional state rather than accidental state. A property that exists on the server for the Designer's benefit, or for business logic, is simply not written to the config and never reaches the browser. A value that would be expensive to serialize, such as a large history array, can be reduced to what the gauge needs before it is assigned. The lab's self-check asks about the role of `OnWebRender`, and the answer is that it is the contract between the two halves of the control.

Events flow the other way through `OnWebEvent`, which Module 7 covers. The two overrides together define the whole wire protocol of the control.

Appearance keys, themes, and design time

A Wisej.NET theme styles controls by appearance key. Giving the gauge control an `AppearanceKey` such as `gauge`, and adding an entry with that key to the theme or a theme mixin, connects the control's colors, fonts, and spacing to the application's theme. Switching themes then restyles the gauge together with every other control, which is the difference between a control that looks native and one that looks pasted in.

Theme resources cover more than colors. Images and icons the control needs can be declared as theme resources so that a dark theme supplies dark variants without code changes. The client class reads its styling from the appearance rather than hard-coding it, and the vendor instance is configured from those values in the apply methods.

Design-time rendering is the final test. The Designer instantiates the control and renders it live, so the client class must cope with placeholder data, an absent server, and repeated creation as the developer resizes it. The lab asks for a design-time screenshot for this reason: if the control renders correctly in the Designer, it will be adopted; if it throws, it will be avoided.



Production habit

Keep Wisej.NET state ownership and client widget behavior clear. Document the client/server contract before relying on the integration in production.



Common mistake to avoid

Copying demo JavaScript into screens without documenting lifecycle, resources, event contracts, cleanup, or security boundaries.



Learning goals

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

- ✓ Create a server-side control derived from `Wisej.Web.Control` when tighter integration is required.
- ✓ Create a client-side class under ``/Platform`` using the `Wisej.NET/qx` class model.
- ✓ Render state through ``OnWebRender`` and map it to client properties or options.
- ✓ Understand `AppearanceKey`, theme resources, and design-time rendering considerations.

Lab / Exam Guide

What you'll build in the hands-on lab, the recommended workflow, the required deliverables, and the self-check before you submit.



Hands-on lab

Create a custom `SimpleGaugeControl` with a server class and a `Platform` client class. Add a basic appearance key and make it render in design mode.

Recommended workflow

- ✓ Open the existing Wisej.NET integration course project and run it locally.
- ✓ Identify the server component, client widget, resource file, or endpoint involved in this module.
- ✓ Make one small change at a time and test it in the browser before moving on.
- ✓ Keep vendor-specific JavaScript inside the adapter or wrapper instead of scattering it across pages.
- ✓ Update the UI so the learner can clearly see success, failure, progress, or communication changes.
- ✓ Take screenshots or notes that prove the integration works and explain the design decision.

Required deliverables

- ✓ Server control class
- ✓ Client qx class skeleton
- ✓ Theme/appearance entry
- ✓ Design-time screenshot or notes

Self-check before submitting

- ✓ When is a custom control worth the extra work?
- ✓ What is the role of `OnWebRender`?
- ✓ Why does theme integration matter for enterprise adoption?