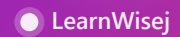


Eigene Wisej-Controls, Client-Klassen und Theming

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

Controls mit nativem Look-and-Feel aus Serverklassen, Client-Widget-Klassen, Appearance-Keys und eingebetteten Ressourcen erstellen. Was das für Entwickler bedeutet, die mit Wisej.NET integrieren – und wie Sie dieses Modul lesen sollten.



Fokus: Produktivbetrieb

Controls mit nativem Look-and-Feel aus Serverklassen, Client-Widget-Klassen, Appearance-Keys und eingebetteten Ressourcen erstellen.

1. Das Integrationsproblem

Manche Integrationen müssen sich wie vollwertige Wisej.NET-Controls verhalten: wiederverwendbar, themefähig, im Designer bearbeitbar und über viele Anwendungen hinweg wartbar.

2. Zentrale Konzepte

Konzept	So setzen Sie es ein
Control-Klasse	Ein serverseitiges Control verwenden, wenn eine nativ wirkende Integration gefordert ist.
AppearanceKey	Die Optik des Controls mit dem Styling auf Theme-Ebene verbinden.

3. Lektion aus der Praxis

Ein eigenes Wisej.NET-Control ist die stärkste Form der Integration. Es sollte eingesetzt werden, wenn das Widget Teil der Produktplattform ist, wenn Theming wichtig ist oder wenn das Control wie ein natives Wisej.NET-Control aussehen und sich so verhalten muss. Das Muster ähnelt dem Erstellen eines eingebauten Controls: Eine Serverklasse stellt .NET-Eigenschaften und Events bereit, und eine Client-Klasse empfängt den gerenderten Zustand und besitzt das browserseitige Objekt. Die Client-Klasse kann je nach gewünschtem Verhalten `wisej.web.Control` oder `wisej.web.Widget` erweitern. Sie kann das Drittanbieter-Objekt erzeugen, apply-Methoden für Eigenschaften implementieren, Größen- oder Appearance-Änderungen behandeln und Callbacks des Anbieters in Wisej.NET-Events übersetzen. Die Serverklasse kann `OnWebRender` überschreiben, um nur die Eigenschaften zu senden, die der Client braucht. Dieser Ansatz erfordert mehr Code, ergibt aber langfristig die sauberste Architektur. Er ist die richtige Wahl für Controls, die als Teil eines Frameworks, einer Erweiterungsbibliothek oder eines Unternehmensprodukts

ausgeliefert werden – überall dort, wo der Wrapper in IntelliSense erscheinen, im Designer funktionieren und über mehrere Releases der zugrunde liegenden Anbieterbibliothek hinweg stabil bleiben muss.

Ein Server-Control für eine engere Integration

Wer von `Wisej.Web.Control` statt von `Wisej.Web.Widget` ableitet, ändert, was die Klasse ist. Ein Widget ist ein Host für ein Skript; ein Control ist ein vollwertiges Mitglied des Frameworks, mit demselben Verhalten bei Layout, Fokus, Theme und Designer wie ein eingebauter Button oder ein Panel. Das `SimpleGaugeControl` im Lab geht diesen Weg, damit sich die Gauge wie jedes andere Control ziehen, dimensionieren, verankern und mit einem Theme versehen lässt.

Die Serverklasse stellt wie zuvor der Wrapper typisierte Eigenschaften und .NET-Events bereit, trägt aber kein InitScript und keinen Options-Beutel mehr. Stattdessen deklariert sie, welche Client-Klasse sie implementiert, und rendert ihren Zustand über `OnWebRender`. Eigenschafts-Setter markieren das Control als aktualisierungsbedürftig, und das Framework entscheidet, wann die Änderung gesendet wird.

Wählen Sie diese Tiefe, wenn das Widget Teil der Produktplattform ist, wenn Theming wichtig ist oder wenn sich das Control für die Entwickler, die damit Masken bauen, nativ anfühlen muss. Der zusätzliche Code zahlt sich mit jeder Maske aus, die das Control verwendet, ohne darüber nachdenken zu müssen.

Eine Client-Klasse unter Platform

Die Browserhälfte eines eigenen Controls ist eine JavaScript-Klasse, die mit dem Client-Klassenmodell von Wisej.NET definiert wird – es basiert auf qooxdoo – und im Ordner `/Platform` der Anwendung liegt, damit das Framework sie mit dem Client lädt. Die Klasse erweitert `wisej.web.Control`, oder `wisej.web.Widget`, wenn das Verhalten eines Skript-Hosts gewünscht ist, und deklariert ihre Eigenschaften mit apply-Methoden.

Eine apply-Methode läuft immer dann, wenn der Server die zugehörige Eigenschaft ändert – genau dort gehört der Aufruf des Anbieters hin: Die apply-Methode für `value` ruft die Set-Value-Methode des Anbieters auf, die apply-Methode für `maximum` aktualisiert die Optionen des Anbieters. Der Konstruktor der Klasse erzeugt die Instanz des Anbieters im Content-Element des Controls, der Destruktor zerstört sie. Größen- und Appearance-Änderungen kommen als Events an, die die Klasse an einer Stelle behandeln kann.

Halten Sie die Client-Klasse frei von Anwendungslogik. Sie übersetzt zwischen Framework und Anbieter und sonst nichts, damit sie von jedem Server-Control wiederverwendet werden kann, das an sie rendert.

Zustand mit OnWebRender rendern

`OnWebRender` ist die Gelegenheit des Servers, genau festzulegen, was der Client sieht. Die Überschreibung erhält ein dynamisches config-Objekt, und das Control schreibt darauf nur die Eigenschaften, die die Client-Klasse deklariert: `config.value`, `config.minimum`, `config.maximum`, `config.caption`. Die Namen stehen in camelCase, passend zu den Eigenschaftsnamen auf dem Client, und Wisej.NET sendet nur die Werte, die sich seit dem letzten Rendering geändert haben.

Das ist beabsichtigter statt zufälliger Zustand. Eine Eigenschaft, die auf dem Server für den Designer oder für Geschäftslogik existiert, wird einfach nicht in das config-Objekt geschrieben und erreicht den Browser nie. Ein Wert, dessen Serialisierung teuer wäre, etwa ein großes Verlaufs-Array, lässt sich vor der Zuweisung

auf das reduzieren, was die Gauge braucht. Der Selbstcheck im Lab fragt nach der Rolle von `OnWebRender`, und die Antwort lautet: Es ist der Vertrag zwischen den beiden Hälften des Controls.

Events fließen in die andere Richtung über `OnWebEvent`, das Modul 7 behandelt. Die beiden Überschreibungen zusammen definieren das gesamte Übertragungsprotokoll des Controls.

Appearance-Keys, Themes und Entwurfszeit

Ein Wisej.NET-Theme gestaltet Controls über ihren Appearance-Key. Erhält das Gauge-Control einen `AppearanceKey` wie `gauge` und bekommen das Theme oder ein Theme-Mixin einen Eintrag mit diesem Schlüssel, sind Farben, Schriften und Abstände des Controls mit dem Theme der Anwendung verbunden. Ein Theme-Wechsel gestaltet die Gauge dann zusammen mit allen anderen Controls um – das ist der Unterschied zwischen einem Control, das nativ aussieht, und einem, das wie hineinkopiert wirkt.

Theme-Ressourcen umfassen mehr als Farben. Bilder und Icons, die das Control braucht, lassen sich als Theme-Ressourcen deklarieren, sodass ein dunkles Theme ohne Codeänderung dunkle Varianten liefert. Die Client-Klasse liest ihr Styling aus der Appearance, statt es fest zu codieren, und die Instanz des Anbieters wird in den apply-Methoden mit diesen Werten konfiguriert.

Das Rendering zur Entwurfszeit ist der letzte Test. Der Designer instanziiert das Control und rendert es live; die Client-Klasse muss also mit Platzhalterdaten, einem fehlenden Server und wiederholter Erzeugung zurechtkommen, während der Entwickler die Größe ändert. Deshalb verlangt das Lab einen Screenshot zur Entwurfszeit: Wird das Control im Designer korrekt gerendert, wird es angenommen; wirft es Fehler, wird es gemieden.



Gewohnheit für den Produktivbetrieb

Halten Sie klar getrennt, welchen Zustand Wisej.NET besitzt und wie sich das Client-Widget verhält. Dokumentieren Sie den Client-Server-Vertrag, bevor Sie sich im Produktivbetrieb auf die Integration verlassen.



Typischer Fehler

Demo-JavaScript in Masken kopieren, ohne Lebenszyklus, Ressourcen, Event-Verträge, Aufräumen oder Sicherheitsgrenzen zu dokumentieren.



Lernziele

Am Ende dieses Moduls können Sie:

- ✓ Ein serverseitiges, von `Wisej.Web.Control` abgeleitetes Control erstellen, wenn eine engere Integration nötig ist.
- ✓ Eine clientseitige Klasse unter ``/Platform`` mit dem Klassenmodell von `Wisej.NET/qx` erstellen.
- ✓ Zustand über ``OnWebRender`` rendern und auf Client-Eigenschaften oder -Optionen abbilden.
- ✓ `AppearanceKey`, Theme-Ressourcen und die Besonderheiten des Renderings zur Entwurfszeit verstehen.

Lab- und Prüfungsleitfaden

Was Sie im praktischen Lab erstellen, der empfohlene Arbeitsablauf, die geforderten Ergebnisse und der Selbstcheck vor der Abgabe.



Praktisches Lab

Erstellen Sie ein eigenes `SimpleGaugeControl`` mit einer Serverklasse und einer Client-Klasse unter ``/Platform``. Fügen Sie einen einfachen Appearance-Key hinzu und sorgen Sie dafür, dass es im Designmodus gerendert wird.

Empfohlener Arbeitsablauf

- ✓ Öffnen Sie das vorhandene Projekt des Wisej.NET-Integrationskurses und starten Sie es lokal.
- ✓ Ermitteln Sie die Server-Komponente, das Client-Widget, die Ressourcendatei oder den Endpunkt, um die es in diesem Modul geht.
- ✓ Nehmen Sie jeweils nur eine kleine Änderung vor und testen Sie sie im Browser, bevor Sie weitermachen.
- ✓ Halten Sie anbieterspezifisches JavaScript im Adapter oder Wrapper, statt es über mehrere Seiten zu verstreuen.
- ✓ Passen Sie die UI so an, dass Erfolg, Fehler, Fortschritt oder Änderungen in der Kommunikation klar erkennbar sind.
- ✓ Halten Sie mit Screenshots oder Notizen fest, dass die Integration funktioniert, und begründen Sie die Designentscheidung.

Geforderte Ergebnisse

- ✓ Server-Control-Klasse
- ✓ Grundgerüst der qx-Client-Klasse
- ✓ Theme-/Appearance-Eintrag
- ✓ Screenshot oder Notizen zur Entwurfszeit

Selbstcheck vor der Abgabe

- ✓ Wann lohnt sich der Mehraufwand für ein eigenes Control?
- ✓ Welche Rolle spielt ``OnWebRender``?
- ✓ Warum ist die Theme-Integration für die Akzeptanz im Unternehmen wichtig?