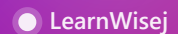


Das mentale Modell der Control-Bibliothek

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

Der Control-Baum auf dem Server und das Widget-Modell im Browser, die Control-Familien, gemeinsame Properties und Namenskonventionen sowie die Shell der Operations Console, die im Lauf des Kurses wächst. Was das für Wisej.NET-Entwickler bedeutet – und wie Sie dieses Modul lesen.



Fokus: Produktivbetrieb

Der Control-Baum auf dem Server und das Widget-Modell im Browser, die Control-Familien, gemeinsame Properties und Namenskonventionen sowie die Shell der Operations Console, die im Lauf des Kurses wächst.

1. Das Problem im Produktivbetrieb

Wisej.NET bringt mehr als hundert Controls mit, und ein Team ohne gemeinsames mentales Modell dafür endet mit zwanzig verschiedenen Suchbildschirmen, fünf Arten, einen Validierungsfehler zu melden, und eigenem JavaScript, wo ein natives Control genügt hätte. Dieses Modul legt das Vokabular, die Entscheidungsregeln und die Konventionen fest, auf denen alle weiteren Module aufbauen, und beginnt mit der Shell der Operations Console.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
Server-Control-Baum	Die Hierarchie der .NET-Control-Objekte, die der Server für jede Session hält; Ihr Code setzt hier Properties und behandelt Events, niemals im Markup.
Browser-Widget	Das JavaScript-Objekt, das Wisej.NET erzeugt, um ein Server-Control darzustellen; es wird über serialisierte Zustandsänderungen synchron gehalten, nicht durch Neuladen der Seite.
Control-Familie	Eine Gruppe von Controls mit gemeinsamem Zweck (Editoren, Container, Listen, Grids, Inhalte, Menüs, Benachrichtigungen); auf dieser Ebene fallen die Entscheidungen.
Extender	Eine Komponente, die vorhandenen Controls zur Entwurfszeit Properties oder Verhalten hinzufügt, ohne von ihnen abzuleiten.
Widget und Erweiterung	Das Widget -Control beherbergt ein eigenes JavaScript-Widget oder eines von Drittanbietern innerhalb des Wisej.NET-Event-Modells; eine Erweiterung ist eine paketierte Integration, meist aus NuGet.
UserControl	Eine wiederverwendbare serverseitige Komposition von Controls mit fachlicher Bedeutung und einer kleinen öffentlichen Schnittstelle.
Leitfaden zur Control-Auswahl	Ein lebendes Team-Dokument, das pro Bildschirmmuster Standard-Controls, Docking, Validierungs- und Benachrichtigungsstil sowie Grid-Konventionen festlegt.

3. Der rote Faden der Lektion

Zuerst Serverobjekte, dann Browser-Widgets

Das mentale Modell, von dem alles andere abhängt: Der Anwendungscode bearbeitet .NET-Objekte auf dem Server, und Wisej.NET serialisiert die Zustandsänderungen, die diese Objekte benötigen, an JavaScript-Widgets im Browser. Wenn die Operations Console `statusLabel.Text` setzt, wird kein Markup geschrieben; die Laufzeitumgebung schickt dem Widget nur den geänderten Zustand. Ein **Button** ist kein HTML-String, sondern ein Serverobjekt mit Properties, Events, einem Lebenszyklus, Designer-Unterstützung, Theming und einer clientseitigen Darstellung, und jede Browser-Session besitzt ihren eigenen Baum solcher Objekte.

Daraus folgt unmittelbar der erste typische Fehler: Controls als HTML-Schnipsel zu behandeln und zu Markup und Client-Skripten zu greifen, wo es bereits eine Property, ein Event oder eine Theme-Einstellung gibt. Der Browser ist die zweite Grenze: Dateisystemzugriff, Drucken, Uploads, Downloads, iframes und JavaScript-Bibliotheken verhalten sich anders als bei Desktop-Controls; deshalb gibt es `Application.Download` und das `Upload` -Control statt eines Dateidialogs.

Controls nehmen am Theme teil: Die Control-Dokumentation nennt die Widget-Klasse und die Appearance-Keys des Themes, die sie verwendet, und das wird wichtig, sobald die Gestaltung mehr braucht als `BackColor`, `ForeColor` oder `Font`. Sie sollten wissen, wann eine einfache Property genügt, wann das Theme der richtige Ort ist, wann ein Extender ausreicht und wann ein wiederverwendbares

`UserControl` oder ein eigenes `Widget` gerechtfertigt ist. Appearance-Keys bis spät im Projekt zu ignorieren ist ein typischer Fehler, der in Dutzenden lokal überschriebenen Farben endet.

Die Control-Familien und die Entscheidungsregeln

Die Bibliothek gliedert sich in Familien, und auf Ebene der Familie fallen die Entscheidungen. Editoren (`TextBox` , `TypedTextBox` , `MaskedTextBox` , `DateTimePicker` , `NumericUpDown` , `ComboBox`) erfassen typisierte Eingaben. Buttons und Menüs (`Button` , `MenuBar` , `ToolBar` , `ContextMenu`) lösen Befehle aus. Container (`Panel` , `SplitContainer` , `TabControl` , `Accordion` , Layout-Panels) geben dem Bildschirm Struktur. Listen, Bäume, Repeater und Grids (`ListView` , `TreeView` , `DataRepeater` , `DataGridView`) zeigen Daten an. Inhalts-Controls stellen Text, Bilder, HTML, PDFs, Dateien und im Browser gehostete Inhalte dar. Benachrichtigungen (`Toast` , `AlertBox` , `MessageBox`) teilen einen Status mit. Extender fügen vorhandenen Controls Verhalten hinzu, ohne von ihnen abzuleiten, und Erweiterungen und Widgets holen externe Web-Funktionen in das Komponenten- und Event-Modell.

Der Kurs behandelt die Bibliothek als Designsystem, nicht als Liste zum Auswendiglernen: Entscheidend ist, welche Familie zur Aufgabe passt, wie sie befüllt wird, wie sie sich bei großen Datenmengen, im Layout und beim Theming verhält, wie sie mit dem Browser kommuniziert und wie sie in einer Mehrbenutzer-Serveranwendung wartbar bleibt. Eine gute Entscheidung beginnt bei der Aufgabe des Benutzers: ein Editor, um einen Wert zu bearbeiten; eine `ListView` , um aus einer Sammlung mit Symbol-, Kachel- oder Detailansicht auszuwählen; eine `TreeView` für hierarchische Daten mit Auf- und Zuklappen; eine `DataGridView` für tabellarische Datensätze mit Sortieren, Bearbeiten, eigenen Spalten oder großen Datenmengen; und ein `UserControl` , wenn eine Gruppe von Controls eine wiederverwendbare fachliche Bedeutung hat.

Drei Regeln halten die Entscheidung ehrlich. Wählen Sie das einfachste Control, das die Aufgabe ohne eigenen Code abbildet, und bevorzugen Sie Standard-Controls und Extender vor jedem eigenen JavaScript-Widget; ein Widget zu bauen, bevor die native Bibliothek ausgeschöpft ist, ist der teuerste typische Fehler im Kurs. Betrachten Sie die Datenmenge, bevor Sie eine Befüllungsstrategie wählen: Manuell befüllte `Rows` und `Items` sind bei kleinen Mengen in Ordnung, große Mengen brauchen Binding, Virtual Mode, Caching oder verzögertes Laden. Verwenden Sie Theme- und Appearance-Konventionen für die anwendungsweite Gestaltung und einzelne Properties nur für lokale Ausnahmen.

Gemeinsame Properties und Konventionen, die alle Module zusammenhalten

Namen sollten die fachliche Rolle benennen, nicht nur den visuellen Typ: `customerGrid` sagt mehr als `dataGridView1` , und `StatusPanel` , `SearchTextBox` und `SaveButton` machen Event-Handler lesbar, ohne den Designer zu öffnen. Wer die Standardnamen des Designers stehen lässt, macht aus `button3_Click` ein Fall für die Archäologie, und die Kursleitung zieht dafür deutlich Punkte ab.

Docking und Anchoring sollten bewusst eingesetzt werden: `Dock` füllt und stapelt, `Anchor` folgt den Rändern, wenn sich die Browsergröße ändert, und absolute Positionierung gehört nur dorthin, wo nichts anderes passt; Layout-Strategien auf einem Bildschirm willkürlich zu mischen ist ein typischer Fehler, der bei der ersten Größenänderung zerbricht. Controls im Tastatur-Arbeitsablauf brauchen einen bewusst gesetzten `TabIndex` , Validierungsfehler brauchen klare Meldungen, Bilder sollten einer `ImageList` - oder Ressourcen-Konvention folgen, aufwendige Vorgänge brauchen sichtbares Feedback, und barrierefreie Beschriftungen sollten der fachlichen Rolle entsprechen.

Eine produktive Anwendung hält all das in einem Leitfaden zur Control-Auswahl fest: einem lebenden Dokument, das pro Bildschirmmuster die Standard-Controls, das Standard-Docking, den Validierungs- und Benachrichtigungsstil, die Platzierung von Menüs, das Icon-Pack, die Grid-Konventionen und die Erwartungen an die Barrierefreiheit festlegt. Das ist keine Bürokratie, sondern der Weg, auf dem ein Team konsistent bleibt. Die `ControlSelection.md` aus Ihrem Lab ist die erste Seite dieses Leitfadens und beantwortet für jeden Bereich diese Fragen:

- Passt das gewählte Control zur Aufgabe des Benutzers, und gibt es ein einfacheres natives Control?
- Sind die Daten flach, hierarchisch, tabellarisch, wiederholt oder visuell, und wie groß werden sie?
- Braucht der Bildschirm eine UserControl-Grenze, und ist ein eigenes Widget wirklich gerechtfertigt?

Die Shell der Operations Console

Das Kursprojekt beginnt als Shell, und jedes Modul fügt einen Abschnitt hinzu, bis Sie eine funktionierende Referenzanwendung statt isolierter Beispiele besitzen. Die Shell ist eine Haupt- `Page` (oder ein `Form`) mit vier Bereichen: der Navigation links, einem Inhaltsbereich, der den Rest ausfüllt, einem Befehlsbereich für die Aktionen des aktuellen Abschnitts und einem Statusbereich, der meldet, was die Konsole gerade tut.

Im Lab enthält der Navigationsbereich pro Platzhalterseite (Editors, Layouts, Lists and Trees, DataGridView, Dashboard, Widgets) einen klar benannten `Button`, der Inhaltsbereich ist ein `Panel` mit Dock Fill, und der Statusbereich ist ein andocktes Panel mit einem `Label`. Die eine Methode, die sich zu schreiben lohnt, tauscht den Inhalt aus, dockt die neue Seite mit Fill an und aktualisiert das Status-Label, sodass jeder Navigationswechsel an einer Stelle gemeldet wird und eine Seite, die nicht geladen werden kann, keine leere Fläche hinterlässt.

```
private void Navigate(Control content, string title)
{
    try
    {
        contentPanel.Controls.Clear();
        content.Dock = DockStyle.Fill;
        contentPanel.Controls.Add(content);
        statusLabel.Text = $"Showing {title}";
    }
    catch (Exception ex)
    {
        statusLabel.Text = $"Could not open {title}";
        MessageBox.Show(ex.Message);
    }
}
```

Jede Platzhalterseite ist eine eigene Klasse, später ein `UserControl`, in einem eigenen Ordner, damit das Abschlussprojekt sie wiederverwenden kann, und die Namen tragen durchgängig die Rolle: `navigationPanel`, `contentPanel`, `commandPanel`, `statusLabel`, `editorsButton`. Die Zusatzaufgabe ist ein kleines Diagnose-Panel, das den Namen des ausgewählten Controls, die Datensatz-ID, das aktive Client-Profil und den Zeitpunkt der letzten Aktualisierung anzeigt.



Gewohnheit für die Control-Bibliothek

Wählen Sie das einfachste native Control, das die Aufgabe des Benutzers abbildet, bevor Sie eigenen Code schreiben. Native Controls bringen Designer-Unterstützung, Theming, Events und Serverzustand bereits mit; greifen Sie als Nächstes zu einem Extender oder einem UserControl und bauen Sie ein JavaScript-Widget erst, wenn kein natives Control passt.



Lernziele

Am Ende dieses Moduls können Sie:

- ✓ die Beziehung zwischen dem Server-Control-Baum von Wisej.NET und den Browser-Widgets erklären.
- ✓ die Control-Bibliothek nach Zweck einordnen: Eingaben, Container, Listen, Daten-Grids, Inhalte, Menüs, Benachrichtigungen, Extender und Erweiterungen.
- ✓ gemeinsame Properties, Events, Benennung, Anchoring, Docking, Bilder, Tab-Reihenfolge, barrierefreie Beschriftungen und Designer-Konventionen konsistent anwenden.
- ✓ eine Shell für das Kursprojekt namens Operations Console erstellen, die im Lauf des Kurses wächst.

Lab- und Prüfungsleitfaden

Was Sie im praktischen Lab bauen, das vorgeschlagene Vorgehen und die geforderten Ergebnisse.



Praktisches Lab

Erstellen Sie die Shell des Control-Katalogs der Operations Console: eine Haupt-Page mit Navigationsbereich, Inhaltsbereich, Befehlsbereich und Statusbereich; Platzhalterseiten für Editors, Layouts, Lists and Trees, DataGridView, Dashboard und Widgets; eine Notiz `ControlSelection.md` im Projekt, die festhält, welche Control-Familie Sie für welchen Bereich verwenden; durchgängig klare Control-Namen; und einen Statusbereich, der sich bei jedem Navigationswechsel aktualisiert.

Ausgangspunkt

Öffnen Sie Visual Studio und erstellen Sie eine neue Wisej.NET-Webanwendung namens `OperationsConsole`; diese Projektmappe erweitern alle späteren Labs, halten Sie die Arbeit jedes Moduls also in einem eigenen Ordner oder Namespace. Die Vorlage liefert eine leere Haupt- `Page` und eine `Program.cs`, die sie öffnet; sonst existiert noch nichts.

Vorgeschlagenes Vorgehen

- ✓ Skizzieren Sie die vier Bereiche, bevor Sie den Designer anfassen, und docken Sie sie dann der Reihe nach an: Navigation links, Befehle oben, Status unten und das Inhalts-Panel zuletzt mit Dock auf Fill.
- ✓ Benennen Sie jedes Control nach seiner fachlichen Rolle, sobald Sie es platzieren (`navigationPanel`, `contentPanel`, `commandPanel`, `statusLabel`, `editorsButton`), und setzen Sie auf den Navigations-Buttons bewusst einen `TabIndex`.
- ✓ Erstellen Sie pro Abschnitt (Editors, Layouts, Lists and Trees, DataGridView, Dashboard, Widgets) ein Platzhalter-UserControl mit einem Titel-Label, jeweils in einem eigenen Ordner, damit das Abschlussprojekt es wiederverwenden kann.
- ✓ Schreiben Sie eine einzige Methode `Navigate(Control content, string title)`, die innerhalb eines try / catch das Inhalts-Panel leert, die Seite mit Fill andockt, sie hinzufügt und das Status-Label aktualisiert; leiten Sie jeden Navigations-Button darüber.
- ✓ Fügen Sie dem Projekt `ControlSelection.md` hinzu und halten Sie für jeden Bereich und jede Platzhalterseite fest, welche Control-Familie Sie gewählt und welche einfachere native Alternative Sie verworfen haben.

Geforderte Ergebnisse

- ✓ Haupt-Shell mit Navigations-, Inhalts-, Befehls- und Statusbereich
- ✓ Platzhalterseiten für Editors, Layouts, Lists and Trees, DataGridView, Dashboard und Widgets
- ✓ `ControlSelection.md` mit der für jeden Bereich gewählten Control-Familie
- ✓ Klare Control-Namen und ein Statusbereich, der sich bei der Navigation aktualisiert
- ✓ Screenshot und eine kurze Notiz, die die Control-Entscheidungen erklärt

Wiederholungsfragen

- ✓ Wenn Sie in der Navigationsmethode `statusLabel.Text` setzen: Was wird tatsächlich an den Browser übertragen, und warum unterscheidet sich das vom Schreiben von HTML?
- ✓ Welchen Bereich der Shell würden Sie als Erstes in ein UserControl umwandeln, und wie sähe seine öffentliche Schnittstelle aus?
- ✓ Wenn ein späteres Modul einen einklappbaren Navigationsbereich braucht: Welcher native Container ersetzt das angedockte Panel, und warum ist das einem eigenen Widget vorzuziehen?



Gewohnheit für die Control-Bibliothek

Wählen Sie das einfachste native Control, das die Aufgabe des Benutzers abbildet, bevor Sie eigenen Code schreiben. Native Controls bringen Designer-Unterstützung, Theming, Events und Serverzustand bereits mit; greifen Sie als Nächstes zu einem Extender oder einem UserControl und bauen Sie ein JavaScript-Widget erst, wenn kein natives Control passt.