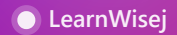


DataGridView meistern

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

BindingSource und explizite Spalten, eigene Editoren und Befehlszellen, Virtual Mode mit CellValueNeeded und serverseitigem Cache sowie Zell-HTML, Zell-Controls und Zell-Painting mit Augenmaß eingesetzt. Was das für Wisej.NET-Entwickler bedeutet – und wie Sie dieses Modul lesen.



Fokus: Produktivbetrieb

BindingSource und explizite Spalten, eigene Editoren und Befehlszellen, Virtual Mode mit CellValueNeeded und serverseitigem Cache sowie Zell-HTML, Zell-Controls und Zell-Painting mit Augenmaß eingesetzt.

1. Das Problem im Produktivbetrieb

DataGridView ist das Control, das am ehesten darüber entscheidet, ob sich eine Geschäftsanwendung ausgereift anfühlt. Aufträge, Rechnungen, Fristen, Logs, Benutzer und Dokumente landen alle in einem Grid, und die Benutzer beurteilen die Anwendung danach, wie dieses Grid lädt, scrollt, Bearbeitungen annimmt und Status meldet. Wisej.NET bietet alles vom manuellen Befüllen bis zu Virtual Mode und Zell-Painting; Meisterschaft heißt, die richtige Technik für die Datenmenge und die Aufgabe des Benutzers zu wählen.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
BindingSource	Ein serverseitiger Vermittler zwischen ViewModel-Zeilen und dem Grid, der ihm eine stabile Datenquelle, eine aktuelle Position und Sortierung gibt, ohne den Bildschirm an die Datenbank zu koppeln.
AutoGenerateColumns	Legt für jede öffentliche Property des gebundenen Typs eine Spalte an; praktisch für einen Prototyp, gefährlich im Produktivbetrieb, weil eine Modelländerung den Bildschirm unbemerkt umstellt.
explizite Spalte	Eine Spalte, die Sie mit Felddbindung, Überschrift, Breite, Ausrichtung, Format, Schreibschutz, Sortiermodus und Editortyp definieren, damit die Oberfläche stabil bleibt, wenn sich das Modell ändert.
Virtual Mode	Mit eingeschaltetem <code>VirtualMode</code> kennt das Grid nur seine Zeilenanzahl und fragt jeden sichtbaren Wert über <code>CellValueNeeded</code> ab, sodass für Zeilen, die niemand sieht, nichts erzeugt wird.
serverseitiger Cache	Ein Puffer in Seitengröße, der <code>CellValueNeeded</code> speist, damit das Scrollen nicht pro Zelle die Datenbank trifft; das <code>DataRead</code> -Event ist der dokumentierte Einstiegspunkt, um ihn aufzubauen und zu pflegen.
eigener Zell-Editor	Ein Control, das der Property <code>Editor</code> einer Spalte zugewiesen ist; es muss seinen Wert in die Zelle zurückschreiben, über <code>CellBeginEdit</code> und <code>CellEndEdit</code> , wenn es kein passendes Change-Event hat.
Zell-HTML vs. Zell-Control	In <code>CellFormatting</code> mit <code>AllowHtml</code> erzeugtes HTML ist reines Anzeige-Markup und nahezu kostenlos; ein eingebettetes Control ist ein echtes Server-Control und Browser-Widget mit Kosten pro Zelle.

3. Der rote Faden der Lektion

Drei Arten zu befüllen, eine für den Produktivbetrieb

Das Grid kann Daten mit oder ohne zugrunde liegende Datenquelle anzeigen. Der direkte Zugriff auf `Rows` und `Columns` ist am einfachsten: Sie fügen Spalten und Zeilen hinzu und verantworten jeden Wert selbst; das passt zu einer kleinen statischen Tabelle wie einer Status-Nachschlagetabelle. Binding über `DataSource` und `DataMember` befüllt das Grid aus einer Liste, einer `BindingSource` oder einem DataSet und ist der Modus für Geschäftsdaten. Der dritte Modus ist das dynamische Erzeugen von Spalten: Ist `AutoGenerateColumns` auf true gesetzt, baut das Grid seine Spalten bei jeder Änderung der Datenquelle neu auf.

Generierte Spalten taugen für einen Zehn-Minuten-Prototyp und werden zur Falle, sobald ein Bildschirm ausgeliefert wird: Fügen Sie `OrderRow` eine Property hinzu, benennen Sie eine andere um, und das Grid folgt dem Modell statt dem Entwurf. Das Orders-Grid der Operations Console definiert deshalb jede Spalte explizit: `DataPropertyName`, `HeaderText`, `Width`, Ausrichtung und Format in `DefaultCellStyle`,

`ReadOnly`, wo der Wert nur informativ ist, `SortMode` und den Spaltentyp, der den richtigen Editor mitbringt. Das Binding selbst bleibt klein:

```
private readonly BindingSource ordersSource = new BindingSource();

private void BindOrders(IEnumerable<OrderRow> rows)
{
    this.ordersGrid.AutoGenerateColumns = false;    // Spalten werden entworfen, nicht
    ermittelt
    this.ordersSource.DataSource = rows.ToList();
    this.ordersGrid.DataSource = this.ordersSource;
}

private void ordersGrid_CellFormatting(object sender, DataGridViewCellFormattingEventArgs e)
{
    if (ordersGrid.Columns[e.ColumnIndex].Name != "StatusColumn") return;
    var status = System.Net.WebUtility.HtmlEncode(Convert.ToString(e.Value));
    e.Value = $"<span class='status-badge'>{status}</span>";
    e.FormattingApplied = true;
}
```

Schalten Sie `AutoGenerateColumns` ab, sobald der Bildschirm stabil ist, und binden Sie eine ViewModel-Collection statt einer Entität: Ein ViewModel ist auf die Anzeige zugeschnitten und enthält Statustext, eine formatierte Summe und eine ID, ohne Persistenzdetails preiszugeben.

Eingebaute Spalten, eigene Editoren und Befehlszellen

Typisierte Werte verdienen typisierte Spalten. Spaltentypen für Text, Kontrollkästchen, ComboBox, Button, Link und Bild decken den Großteil des Orders-Grids ab, und Wisej.NET ergänzt Spalten auf Basis der Datums-, Zahlen- und Masken-Editoren aus Modul 2. Eine Befehlszelle ist eine Button- oder Link-Spalte: Die Spalte *Ship* aus dem Lab löst `CellContentClick` aus, der Handler liest die ID der Zeile aus dem gebundenen Element und ruft den Service auf. Nichts vom Auftrag selbst lebt in der Zelle.

Wenn kein eingebauter Editor passt, weisen Sie der Property `Editor` der Spalte ein Control zu; das klassische Beispiel ist ein `MonthCalendar` auf einer Zelle mit Fälligkeitsdatum. Die wichtige Warnung: Der Editor muss den Zellwert korrekt aktualisieren. Das Grid erwartet `TextChanged` oder ein gleichwertiges Change-Event; löst Ihr Control keines aus, behandeln Sie `CellBeginEdit`, um den Zellwert in den Editor zu kopieren, und `CellEndEdit`, um den bearbeiteten Wert zurückzukopieren. Testen Sie den Hin- und Rückweg mit Tab, Enter und einem Klick an eine andere Stelle: Ein Editor, der nie übernimmt, ist der klassische Fehler bei eigenen Editoren.

Virtual Mode, CellValueNeeded und der Cache

Große Datenmengen verlangen einen anderen Ansatz. 100.000 Aufträge zu laden, weil die Datenbank sie liefern kann, ist keine UI-Strategie: Niemand kann so viele Zeilen durchsehen, und jede kostet Serverspeicher und Rendering im Browser. Filtern Sie zuerst, bieten Sie serverseitige Suche und seitenweises Laden an und reservieren Sie den Virtual Mode für den Pfad, in dem der Benutzer wirklich durch ein sehr großes Ergebnis scrollen muss. Dort setzen Sie `VirtualMode` auf true, geben dem Grid

seine `RowCount` und seine Spalten und behandeln `CellValueNeeded`, um jeden Wert auf Anfrage zu liefern.

Das Event allein genügt nicht. Ein Handler, der innerhalb von `CellValueNeeded` die Datenbank abfragt, läuft einmal pro sichtbarer Zelle, sodass ein einziges Scrollen Hunderte von Abfragen erzeugen kann. Der `OrderCache` aus dem Lab hält die aktuelle Seite auf dem Server: `CellValueNeeded` fragt ihn nach Zeile und Spalte, er bedient einen Treffer aus dem Speicher und holt bei einem Miss eine ganze Seite. Die Wisej.NET-Dokumentation nennt das `DataRead`-Event als Ort, um diesen Cache aufzubauen und zu pflegen, weil es Ihnen mitteilt, welchen Zeilenblock das Grid gleich lesen wird. Profilen Sie mit realistischen Mengen, nicht mit den zwanzig Zeilen des Demo-Services, und fragen Sie sich, was bei zehnfacher Datenmenge zuerst bricht.

Zell-HTML, Zell-Controls, Zell-Painting und das zusammengesetzte Grid

Zell-HTML ist die beste Zwischenlösung für visuelle Vielfalt. Braucht ein Status ein Badge, ein Icon, einen Fortschrittsbalken oder mehrere Textfragmente, setzen Sie `AllowHtml` und erzeugen das Markup in `CellFormatting`, wie im obigen Codeausschnitt; der rohe Status bleibt im Modell, und das HTML dient nur der Anzeige. Behandeln Sie es auch so: Kodieren Sie Benutzertext, bevor er ins Markup gelangt, damit ein Kundenname nie zu ausführbarem HTML wird, und halten Sie Geschäftsvalidierung aus dem Formatierungs-Handler heraus.

Zell-Controls sind mächtiger und gefährlicher. Wisej.NET kann ein echtes Control in eine Zelle setzen, sogar über mehrere Zellen hinweg: einen Button in einer Zeile, einen Schieberegler über mehrere Spalten, eine kleine Vorschau. Jedes davon ist ein Server-Control und ein Browser-Widget, daher sind Controls in jeder Zelle eines Grids mit 10.000 Zeilen 10.000 Controls. Setzen Sie sie für die ausgewählte Zeile, Summenzeilen, einen Befehlsbereich oder ein kleines Grid ein. Zell-Painting liegt am äußersten Ende der Skala: Aktivieren Sie das benutzerdefinierte Zeichnen für die Zelle und behandeln Sie `CellPaint`, wenn die Anzeige eigenes Zeichnen braucht oder vorhandenen .NET-Zeichencode wiederverwendet, nicht für Gestaltung, die Themes, Zellstile oder HTML bereits abdecken.

Schließlich verhält sich die zusammensetzbare `DataGridView` wie ein Container: Sie nimmt verbundene Controls wie ein Filter-Panel, eine Statusleiste, einen Navigator oder einen Baum auf, die innerhalb oder um das Grid angedockt und mit `BringToFront` und `SendToBack` geordnet werden. So bleibt die vollständige Grid-API sichtbar, statt hinter einem UserControl zu verschwinden; Filter- und Statusleiste aus dem Lab sind auf diese Weise aufgebaut.

- Explizite Spalten und eine BindingSource oder ein ViewModel für jeden produktiven Bildschirm; `AutoGenerateColumns` aus, sobald der Bildschirm stabil ist.
- Vor dem Anzeigen filtern; Virtual Mode plus Cache für wirklich große Mengen.
- Zell-HTML vor eingebetteten Controls; eingebettete Controls sparsam; Zell-Painting nur, wenn Gestaltung, HTML und normale Controls nicht ausreichen.
- Mit realistischen Daten profilen, nicht mit Beispieldaten.



Typische Fehler

`AutoGenerateColumns` im Produktivbetrieb eingeschaltet lassen. Alle Datensätze beim Start laden. Aus `CellValueNeeded` für jede Zelle die Datenbank abfragen, ohne Cache. Controls in jede Zeile eines riesigen Grids einbetten. Geschäftsvalidierung mit visueller Formatierung vermischen. Ein Grid wie eine Datenbanktabelle behandeln statt als aufgabenorientierte Ansicht.



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:

- ✓ eine DataGridView an eine BindingSource binden und AutoGenerateColumns bewusst steuern.
- ✓ eingebaute Spalten und eigene Editoren für typisierte Werte und Befehlszellen verwenden.
- ✓ Virtual Mode, CellValueNeeded, die Konzepte von DataRead und Caching sowie die Performance-Grenzen bei großen Datenmengen umsetzen.
- ✓ Zell-HTML, Zell-Controls, Zell-Painting und zusammengesetzte Grids mit Augenmaß einsetzen.

Lab- und Prüfungsleitfaden

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



Praktisches Lab

Erstellen Sie das Orders-Grid der Operations Console: eine DataGridView, gebunden an eine BindingSource, mit expliziten Spalten und kontrolliertem AutoGenerateColumns, eine Statuszelle, die mit AllowHtml und CellFormatting dargestellt wird, eine eigene Editor- oder Befehlsspalte, eine Variante mit Virtual Mode oder Cache für große Datenmengen sowie eine zusammengesetzte Filter- und Statusleiste, die innerhalb oder um das Grid andockt ist.

Ausgangspunkt

Öffnen Sie die Projektmappe `OperationsConsole` aus Modul 4 und wählen Sie in der Shell die Platzhalterseite DataGridView. Dokument-Explorer, `CustomerEditor` und Navigations-Shell existieren bereits; die Seite Orders ist noch eine leere `TabPage`, und ein kleiner In-Memory- `OrderService`, der `OrderRow` -ViewModels zurückgibt, genügt für jeden Schritt dieses Labs.

Vorgeschlagenes Vorgehen

- ✓ Fügen Sie `ordersGrid` und eine BindingSource `ordersSource` hinzu, setzen Sie `AutoGenerateColumns` auf false und definieren Sie jede Spalte explizit: Auftragsnummer, Kunde, Fälligkeitsdatum, Summe, Status, mit bewusst gewählter Überschrift, Breite, Ausrichtung, Format und Schreibschutz.
- ✓ Markieren Sie die Statusspalte mit `AllowHtml` und behandeln Sie `CellFormatting`, um den rohen Status in ein kodiertes Badge umzuwandeln; lassen Sie den Wert im Modell unverändert und setzen Sie `FormattingApplied`.
- ✓ Fügen Sie entweder über die Property `Editor` der Spalte einen MonthCalendar-Editor für das Fälligkeitsdatum hinzu (mit `CellBeginEdit` / `CellEndEdit`, falls der Wert nicht verlustfrei hin- und zurückkommt) oder eine Button-Spalte, deren `CellContentClick` den Service mit der Auftrags-ID aufruft.
- ✓ Bauen Sie den Pfad für große Datenmengen: `VirtualMode` auf true, `RowCount` aus dem Service und ein `OrderCache`, der `CellValueNeeded` aus einer Seite im Speicher bedient, bei einem Miss eine neue Seite holt und über `DataRead` den Block vorab lädt, den das Grid gleich anzeigen wird.
- ✓ Setzen Sie das Grid zusammen: Docken Sie eine Filterleiste (Suchfeld, Status-ComboBox, Apply-Button) über oder innerhalb des Grids an und darunter eine Statusleiste, die Zeilenanzahl, ausgewählten Auftrag und letzte Aktualisierung zeigt; ordnen Sie sie mit `BringToFront` und `SendToBack`.

Geforderte Ergebnisse

- ✓ Orders-DataGridView mit expliziten Spalten, gebunden an eine BindingSource
- ✓ Statuszelle, dargestellt mit AllowHtml und CellFormatting

- ✓ Eine eigene Editor- oder Befehlsspalte
- ✓ Variante des Grids für große Datenmengen mit Virtual Mode oder Cache
- ✓ Zusammengesetzte Filter- und Statusleiste, innerhalb oder um das Grid angedockt

Wiederholungsfragen

- ✓ Wenn `OrderRow` im nächsten Sprint zwei neue Properties erhält: Was ändert sich auf dem Bildschirm mit generierten Spalten, und was mit Ihren expliziten Spalten?
- ✓ Ihr `CellValueNeeded` -Handler funktioniert mit 200 Aufträgen. Was passiert mit der Datenbank, wenn ein Benutzer den Scrollbalken durch 200.000 zieht, und welcher Teil Ihres Entwurfs fängt das ab?
- ✓ Warum gehört das Status-Badge in `CellFormatting` und nicht in das `OrderRow` -Modell, und was würde schiefgehen, wenn ein Kundenname ohne Kodierung in dieses HTML gelangte?



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.