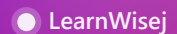


Zellenzeichnen in der DataGridView

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

Zeichnen im Grid: UserPaint auf Zellen- oder Spaltenebene, DataGridViewCellPaintEventArgs mit RowIndex und ColumnIndex, Fortschritts- und Sparkline-Zellen, die Zeilen sparsam auslesen, und der ehrliche Vergleich mit AllowHtml-Zellinhalten in einem Grid mit tausend Zeilen. Was das für erfahrene .NET-Entwickler bedeutet, die bereits Wisej.NET-Oberflächen bauen und nun eigene Grafiken brauchen – und wie Sie dieses Modul lesen.



Fokus: Produktivbetrieb

Zeichnen im Grid: UserPaint auf Zellen- oder Spaltenebene, DataGridViewCellPaintEventArgs mit RowIndex und ColumnIndex, Fortschritts- und Sparkline-Zellen, die Zeilen sparsam auslesen, und der ehrliche Vergleich mit AllowHtml-Zellinhalten in einem Grid mit tausend Zeilen.

1. Das Problem im Produktivbetrieb

Modul 2 hat VisualOperationsStudio ein `TelemetryGauge` gegeben: ein Control, eine Maschine, ein paar hundert Neuzeichnungen pro Stunde. Das Betriebs-Grid ist ein anderes Kaliber. Eine Schichtleiterin öffnet eine Liste mit tausend Maschinen und will auf einen Blick sehen, welche nahe an ihrer Grenze sind und welche den ganzen Vormittag abgedriftet sind. Zahlen in einer Spalte leisten das nicht, und ein Gauge pro Zeile wären tausend serverseitige Controls in einer einzigen Sitzung. Das Grid selbst muss zeichnen, einmal pro sichtbarer Zelle, und in dem Moment aufhören, in dem die Zelle aus dem Bild scrollt.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
<code>UserPaint</code>	Der ausdrückliche Schalter: <code>DataGridViewCell.UserPaint = true</code> für eine einzelne Zelle, <code>DataGridViewColumn.UserPaint = true</code> für eine ganze Spalte. Ohne ihn fordert das Grid Sie nie zum Zeichnen auf.
<code>CellPaint</code>	Das Event auf Grid-Ebene, das für selbst gezeichnete Zellen ausgelöst wird, wenn sie gezeichnet werden müssen; Sie abonnieren es einmal am Grid, nicht einmal pro Zeile, und verzweigen im Handler.
<code>DataGridViewCellPaintEventArgs</code>	Leitet sich von <code>PaintEventArgs</code> ab, trägt also <code>Graphics</code> und <code>ClipRectangle</code> , und ergänzt <code>RowIndex</code> und <code>ColumnIndex</code> , damit ein Handler mehrere Spalten bedienen kann.
keine WinForms-Helfer	Die dokumentierten Wisej.NET-Ereignisargumente bieten die WinForms-Komfortmethoden <code>PaintBackground()</code> und <code>PaintContent()</code> nicht. Ihr Handler besitzt das gesamte visuelle Ergebnis dieser Zelle.
<code>e.ClipRectangle</code> als Fläche	Das lokale Zielrechteck der Zelle. Spalten lassen sich in der Breite ändern und das Grid kann responsiv angeordnet werden, also muss jede gezeichnete Koordinate daraus abgeleitet sein und nicht aus einer Konstanten.
seiteneffektfreie Render-Helfer	Methoden in der Form <code>DrawProgressCell(Graphics g, Rectangle rect, double value)</code> , sodass der Handler zu Verteilercode wird und derselbe Helfer später in eine Off-Screen-Bitmap rendern kann.
<code>AllowHtml</code> -Zellinhalt	Die browsereigene Alternative: Icons, Spans, Labels und CSS, die der Client darstellt – und der Grund, warum die Dokumentation <code>CellPaint</code> für Darstellungen reserviert, die für HTML zu komplex sind.

3. Der rote Faden der Lektion

Eine Zelle in eine Zeichenfläche verwandeln

An einer `DataGridView` von Wisej.NET wird standardmäßig nichts selbst gezeichnet. Sie entscheiden sich pro Zelle oder pro Spalte dafür, indem Sie `UserPaint` auf `true` setzen – und erst dann löst das Grid für diese Fläche `CellPaint` aus. Den Schalter zu vergessen ist der häufigste Grund dafür, dass ein korrekt aussehender Handler nie läuft; halten Sie die aktivierende Zeile und die Abonnementzeile also in Ihrer Methode zur Spaltenkonfiguration beieinander.

Bevorzugen Sie den Spaltenumfang. Im Lab fügen Sie `gridOperations` explizite Spalten hinzu, setzen `UserPaint = true` ausschließlich für die Spalten Health und Trend und abonnieren `CellPaint` einmal am Grid. Das ergibt ein Abonnement für den gesamten Bildschirm statt eines pro Zeile, und es belässt jede andere Spalte — Kennung, Maschinenname, Standort, letzter Messwert — als gewöhnlichen Grid-Inhalt, den der Browser markieren, durchsuchen und skalieren kann. Das eigene Zeichnen für das gesamte Grid

einzuschalten ist die gegenteilige Entscheidung: Sie bekommen Zellen, die Sie nie zeichnen wollten, und der Handler muss sie bei jedem Neuzeichnen herausfiltern.

Was das Event Ihnen übergibt – und was nicht

Weil `DataGridViewCellPaintEventArgs` von `PaintEventArgs` abgeleitet ist, ist der Zeichenteil genau das, was Modul 2 gelehrt hat: `e.Graphics` ist die Oberfläche, `e.ClipRectangle` ist Ihr Bereich, und Sie geben jeden selbst erzeugten Pen, Brush und Path frei, während Sie `e.Graphics` in Ruhe lassen. Neu ist das Verzweigen. `e.RowIndex` und `e.ColumnIndex` brauchen beide eine Prüfung, bevor Sie irgendetwas anfassen: Ein negativer `RowIndex` ist keine Datenzeile, und eine Spalte, für die Sie sich nicht entschieden haben, sollte sofort zurückkehren.

Der knappe Teil der API ist der, über den viele stolpern. Die Ereignisargumente von Wisej bieten weder `PaintBackground()` noch `PaintContent()`, also muss ein WinForms-Beispiel, das sie aufruft, umgeschrieben statt kompiliert werden – und Sie entscheiden selbst, wie Hintergrund und Inhalt der Zelle aussehen. Betrachten Sie das als Vorteil: Wenn der Handler die gesamte Zelle erzeugt, gibt es keinen halb gezeichneten Zustand, über den man nachdenken müsste.

```
private void gridOperations_CellPaint(object sender, DataGridViewCellPaintEventArgs e)
{
    if (e.RowIndex < 0)
        return;

    var rect = Rectangle.Inflate(e.ClipRectangle, -4, -5);

    if (e.ColumnIndex == healthColumn.Index)
        DrawHealthBar(e.Graphics, rect, Convert.ToInt32(
            gridOperations.GetValue(e.ColumnIndex, e.RowIndex) ?? 0));
    else if (e.ColumnIndex == trendColumn.Index)
        DrawSparkline(e.Graphics, rect, TrendOf(e.RowIndex));
}
```

Fortschritts-, Status- und Sparkline-Zellen

Drei Formen decken die meisten Betriebs-Grids ab. Eine Fortschrittszelle füllt eine inaktive Skala mit einem aktiven Rechteck, dessen Breite proportional zu einem normalisierten Wert ist, und zeichnet eine Zahl nur, wenn die Spalte breit genug ist. Eine Statuszelle ist eine einzelne gefüllte Ellipse, manchmal mit einer kurzen Beschriftung; sie ist die günstigste nützliche Darstellung im Grid und oft klarer als ein Icon. Eine Sparkline normalisiert eine kleine Reihe auf das Zellenrechteck und verbindet die Punkte mit `DrawLines`, so wie es die Trend-Spalte im Lab tut: Ihre Aufgabe ist es zusammenzufassen, nie eine Reihe aus zehntausend Punkten in vierzig Pixeln abzubilden.

Halten Sie die Mathematik aus dem Handler heraus. Geben Sie jeder Darstellung einen Helfer, der ein `Graphics`, ein `Rectangle` und die benötigten Werte entgegennimmt – dann schrumpft der Handler auf eine Prüfung und zwei Aufrufe. Der Off-Screen-Renderer aus Modul 6 kann dann dieselbe Methode aufrufen, um dieselbe Sparkline in ein exportiertes PNG zu setzen. Vergessen Sie die Zustände nicht, die dem Grid gehören: Eine selbst gezeichnete Zelle muss auch lesbar bleiben, wenn ihre Zeile markiert ist – wählen Sie also Farben, die die Markierung berücksichtigen, oder halten Sie die Darstellung zurückhaltend

und lassen Sie die Zeilenhervorhebung sprechen. Lassen Sie Farbe nie der einzige Träger eines kritischen Status sein; derselbe Wert sollte auch als Text oder als Tooltip existieren.

Wann Zeichnen das falsche Werkzeug ist

Wisej.NET stellt über `AllowHtml` auch HTML in Zellen dar, und die Dokumentation positioniert `CellPaint` als Antwort für Inhalte, die für HTML zu komplex sind, oder für vorhandenen .NET-Zeichencode, den Sie wiederverwenden möchten. Wählen Sie HTML, wenn sich die Darstellung mit Text, Icons, Spans und CSS ausdrücken lässt, wenn Suche, Markierung, Barrierefreiheit oder Browser-Skalierung von echtem DOM-Inhalt profitieren, oder wenn es viele Tausend Zellen sind. Wählen Sie das Zeichnen, wenn Sie Geometrie auf Pixelebene brauchen, wenn die Darstellung kompakt ist und sich selten ändert, oder wenn das HTML am Ende komplizierter wäre als der Zeichencode.

Eine zweite Alternative hat mit HTML nichts zu tun: die eigenen Zelltypen des Grids und seine Fähigkeit, Controls aufzunehmen. Ein Button, eine Checkbox, ein Datumsauswähler oder eine Combobox sollten eine echte interaktive Zelle bleiben. Zeichnen ist ein Rendering-Modell, kein Interaktionsmodell, und ein gezeichneter Button zwingt Sie dazu, Zeiger-Rückmeldung, Tastaturzugang, Fokus und Befehle von Hand nachzubauen. Deshalb baut das Lab den Statusindikator zweimal und bittet Sie aufzuschreiben, welchen Sie ausliefern würden.

Tausend Zeilen scrollen

Eine `DataGridView` kommt mit großen Datenmengen über Binding und virtuelle Modi gut zurecht, und ein unachtsamer `CellPaint`-Handler macht diesen Vorteil in einer Zeile zunichte. Beim Scrollen wird der Handler für jede neu sichtbare Zelle erneut aufgerufen, also multipliziert sich alles Teure damit, wie schnell die Benutzerin die Bildlaufleiste zieht.

- Führen Sie im Handler niemals E/A-, Datenbank- oder Serviceaufrufe aus.
- Laden Sie keine Bilder und erzeugen Sie keine Fonts pro Zelle; bauen Sie sie einmal und verwenden Sie sie wieder.
- Zeichnen Sie die kleinstmögliche Menge an Spalten und verringern Sie die Punktzahl der Sparklines.
- Halten Sie den Zustandszugriff bei $O(1)$: Indizieren Sie die gebundene Liste oder lesen Sie den Zellwert über das Grid, aber durchsuchen Sie nie eine Sammlung pro Zelle.
- Messen Sie mit realistischen Zeilenzahlen — hundert Zeilen beweisen nichts über tausend.

Die tausend generierten Zeilen im Lab gibt es genau für diesen letzten Punkt. Binden Sie sie an, ziehen Sie die Bildlaufleiste von oben nach unten und vergleichen Sie, was die beiden gezeichneten Spalten kosten, mit der HTML-Spalte daneben. Diese Messung gehört in Ihre Notizen, nicht eine Vorliebe.



Gewohnheit: erst die Oberfläche, dann das Zeichnen

Wählen Sie die Rendering-Oberfläche, bevor Sie einen einzigen Zeichenaufruf schreiben. Fragen Sie, wo die Pixel entstehen und was über das Netzwerk geht: Der Paint-Handler eines Controls läuft auf dem Server und liefert ein Bild aus, CellPaint tut dasselbe innerhalb einer Grid-Zelle, Canvas schickt Zeichenbefehle, die der Browser ausführt, und behält über eine Größenänderung hinweg nichts, und System.Drawing.Managed erzeugt Bild-Bytes ganz ohne Control. Halten Sie das semantische Modell in C#-Objekten, machen Sie jeden Renderer zu einer deterministischen Funktion dieses Modells, invalidieren Sie nur, wenn sich eine visuelle Abhängigkeit geändert hat, und lassen Sie immer ein textliches Äquivalent für alles Wichtige zurück, das nur als Pixel existiert.



Lernziele

Am Ende dieses Moduls können Sie:

- ✓ Das Zeichnen auf Zellen- oder Spaltenebene mit UserPaint einschalten.
- ✓ DataGridViewCellPaintEventArgs sicher verwenden.
- ✓ Fortschritts-, Status- und Sparkline-Zellen bauen.
- ✓ Eigenes Zeichnen mit Zell-HTML, eingebauten Zelltypen, Controls in Zellen und Bildern vergleichen.
- ✓ Zellenzeichnen für große Datenmengen so entwerfen, dass das Scrollen nicht zum Rendering-Engpass wird.

Lab- und Prüfungsleitfaden

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



Praktisches Lab

Geben Sie VisualOperationsStudio ein Betriebs-Grid mit zwei selbst gezeichneten Spalten. Fügen Sie gridOperations explizite Spalten hinzu, setzen Sie UserPaint = true ausschließlich für die Spalten Health und Trend und behandeln Sie CellPaint an einer einzigen Stelle: Kehren Sie sofort zurück, wenn e.RowIndex negativ ist oder e.ColumnIndex keine der beiden Zielspalten trifft, und zeichnen Sie dann innerhalb von e.ClipRectangle einen schwellenwertgefärbten Fortschrittsbalken für Health und eine Sparkline als Linienzug für Trend. Lesen Sie die Werte jeder Zeile aus dem gebundenen MachineStatus-Objekt, statt im Handler eine Liste zu durchsuchen, respektieren Sie die Farben der markierten Zeile und halten Sie den Handler frei von Seiteneffekten. Binden Sie 1.000 generierte Zeilen an, scrollen Sie und bauen Sie denselben Statusindikator anschließend ein zweites Mal als AllowHtml-Zelle, damit Sie schriftlich begründen können, welche Variante Sie ausliefern würden.

Ausgangspunkt

Sie öffnen VisualOperationsStudio in dem Stand, den Modul 2 hinterlassen hat: Die

VisualOperationsPage trägt das Control TelemetryGauge und seinen Helfer GaugeGeometry, und die Seite hat Platz für ein Grid, aber noch keine Daten. Fügen Sie eine Klasse MachineStatus mit Kennung, Name, Standort, einem Prozentwert Health und einer kurzen Liste der letzten Messwerte für den Trend hinzu, dazu einen Generator, der tausend davon erzeugen kann.

Vorgeschlagenes Vorgehen

- ✓ Fügen Sie gridOperations mit expliziten Spalten für Kennung, Maschine, Standort, Health und Trend hinzu, binden Sie zuerst zehn MachineStatus-Zeilen an und prüfen Sie, dass die schlichten Spalten dargestellt werden, bevor überhaupt gezeichnet wird.
- ✓ Setzen Sie UserPaint = true ausschließlich für die Spalten Health und Trend, abonnieren Sie gridOperations.CellPaint einmal und beginnen Sie den Handler mit einer Prüfung, die zurückkehrt, wenn e.RowIndex negativ ist oder e.ColumnIndex keine der beiden Zielspalten trifft.
- ✓ Schreiben Sie DrawHealthBar(Graphics, Rectangle, int) und DrawSparkline(Graphics, Rectangle, IReadOnlyList<double>) als seiteneffektfreie Helfer, die jede Koordinate aus dem übergebenen Rechteck ableiten, die Füllfarbe aus der Warn- und der kritischen Schwelle wählen und jeden erzeugten Pen und Brush freigeben.
- ✓ Machen Sie den Handler zu reinem Verteilercode: Verkleinern Sie e.ClipRectangle mit Rectangle.Inflate, holen Sie die Werte der Zeile in einem O(1)-Schritt, übergeben Sie sie an den Helfer und halten Sie die markierte Zeile lesbar, indem Sie Farben wählen, die sich weiterhin vom Markierungshintergrund abheben.
- ✓ Binden Sie 1.000 generierte Zeilen an, ziehen Sie die Bildlaufleiste von oben nach unten, fügen Sie dann eine dritte Statusspalte hinzu, die denselben Warn- oder Kritischzustand über AllowHtml darstellt, scrollen Sie erneut und halten Sie beide Ergebnisse fest.

Geforderte Ergebnisse

- ✓ gridOperations mit expliziten Spalten und UserPaint = true nur für die Spalten Health und Trend
- ✓ Ein einziger CellPaint-Handler, der e.RowIndex und e.ColumnIndex prüft, bevor irgendetwas gezeichnet wird
- ✓ Schwellenwertgefärbte Fortschrittszelle und eine Sparkline-Zelle als Linienzug, gezeichnet innerhalb von e.ClipRectangle
- ✓ Zeilenwerte, die aus dem gebundenen MachineStatus-Objekt gelesen werden, ohne Suche, Allokation oder Control-Erzeugung pro Zelle
- ✓ Eine parallel aufgebaute AllowHtml-Statusspalte samt Notiz, die beide beim Scrollen über 1.000 Zeilen vergleicht

Wiederholungsfragen

- ✓ Ein Kollege fügt ein WinForms-Beispiel zum Zellenzeichnen ein, das mit einem Aufruf von `e.PaintBackground( ... )` beginnt. Erklären Sie, was die Ereignisargumente von Wisej.NET Ihnen stattdessen geben und was der Handler nun selbst entscheiden muss.
- ✓ Ihr Sparkline-Helfer verdrahtet eine Plotbreite von 100 Pixeln fest und sieht auf Ihrem Bildschirm gut aus. Beschreiben Sie zwei Benutzeraktionen, die ihn zerstören, und was der Helfer stattdessen auslesen sollte.
- ✓ Die HTML-Statusspalte und die gezeichnete Health-Spalte zeigen beide eine kritische Maschine. Begründen Sie, welche Sie für ein Grid ausliefern würden, das Support-Mitarbeitende durchsuchen und mit einem Screenreader lesen, und was Sie jeweils verlieren.



Gewohnheit: erst die Oberfläche, dann das Zeichnen

Wählen Sie die Rendering-Oberfläche, bevor Sie einen einzigen Zeichenaufruf schreiben. Fragen Sie, wo die Pixel entstehen und was über das Netzwerk geht: Der Paint-Handler eines Controls läuft auf dem Server und liefert ein Bild aus, CellPaint tut dasselbe innerhalb einer Grid-Zelle, Canvas schickt Zeichenbefehle, die der Browser ausführt, und behält über eine Größenänderung hinweg nichts, und System.Drawing.Managed erzeugt Bild-Bytes ganz ohne Control. Halten Sie das semantische Modell in C#-Objekten, machen Sie jeden Renderer zu einer deterministischen Funktion dieses Modells, invalidieren Sie nur, wenn sich eine visuelle Abhängigkeit geändert hat, und lassen Sie immer ein textliches Äquivalent für alles Wichtige zurück, das nur als Pixel existiert.