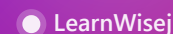


Le dessin des cellules du DataGridView

Guide de la leçon · Guide du lab / de l'examen



Guide de la leçon

Dessiner à l'intérieur de la grille : UserPaint à la portée d'une cellule ou d'une colonne, DataGridViewCellPaintEventArgs avec RowIndex et ColumnIndex, des cellules de progression et de sparkline qui lisent les lignes à bas coût, et la comparaison honnête avec un contenu de cellule AllowHtml sur une grille de mille lignes. Ce que cela signifie pour un développeur .NET expérimenté qui construit déjà des écrans Wisej.NET et a désormais besoin de graphiques personnalisés — et comment aborder ce module.



Objectif en production

Dessiner à l'intérieur de la grille : UserPaint à la portée d'une cellule ou d'une colonne, DataGridViewCellPaintEventArgs avec RowIndex et ColumnIndex, des cellules de progression et de sparkline qui lisent les lignes à bas coût, et la comparaison honnête avec un contenu de cellule AllowHtml sur une grille de mille lignes.

1. Le problème en production

Le Module 2 a doté VisualOperationsStudio d'un **TelemetryGauge** : un contrôle, une machine, quelques centaines de redessins par heure. La grille d'exploitation est d'une autre nature. Un chef de poste ouvre une liste de mille machines et veut voir d'un coup d'œil lesquelles approchent de leur limite et lesquelles dérivent depuis le matin. Des nombres dans une colonne n'y suffiront pas, et une jauge par ligne ferait un millier de contrôles côté serveur dans une seule session. C'est la grille elle-même qui doit dessiner, une fois par cellule visible, et s'arrêter dès que la cellule sort de l'écran.

2. Concepts clés

Concept	Pourquoi c'est important
<code>UserPaint</code>	L'interrupteur explicite : <code>DataGridViewCell.UserPaint = true</code> pour une cellule, <code>DataGridViewColumn.UserPaint = true</code> pour une colonne entière. Sans lui, la grille ne vous demande jamais de dessiner.
<code>CellPaint</code>	L'événement de niveau grille déclenché pour les cellules dessinées par votre code lorsqu'elles doivent être tracées ; vous vous abonnez une fois sur la grille, pas une fois par ligne, et vous aiguillez dans le gestionnaire.
<code>DataGridViewCellPaintEventArgs</code>	Dérive de <code>PaintEventArgs</code> , il porte donc <code>Graphics</code> et <code>ClipRectangle</code> , et ajoute <code>RowIndex</code> et <code>ColumnIndex</code> pour qu'un seul gestionnaire serve plusieurs colonnes.
pas de helpers WinForms	Les arguments d'événement documentés de Wisej.NET n'exposent pas les méthodes de confort WinForms <code>PaintBackground()</code> et <code>PaintContent()</code> . Votre gestionnaire est propriétaire de tout le résultat visuel de la cellule.
<code>e.ClipRectangle</code> comme surface	Le rectangle de destination local de la cellule. Les colonnes peuvent être redimensionnées et la grille mise en page de façon responsive : chaque coordonnée que vous dessinez doit donc en être dérivée, et non d'une constante.
helpers de rendu purs	Des méthodes de la forme <code>DrawProgressCell(Graphics g, Rectangle rect, double value)</code> , pour que le gestionnaire devienne du code d'aiguillage et que le même helper puisse plus tard dessiner dans un bitmap hors écran.
contenu de cellule <code>AllowHtml</code>	L'alternative native au navigateur : icônes, spans, libellés et CSS rendus par le client, et la raison pour laquelle la documentation réserve <code>CellPaint</code> aux visuels trop complexes pour du HTML.

3. Le fil de la leçon

Transformer une cellule en surface de dessin

Rien, dans un `DataGridView` de Wisej.NET, n'est dessiné par votre code par défaut. Vous y adhérez cellule par cellule ou colonne par colonne en mettant `UserPaint` à `true`, et la grille ne déclenche `CellPaint` pour cette surface qu'à ce moment-là. Oublier l'interrupteur est la raison la plus fréquente pour laquelle un gestionnaire d'apparence correcte ne s'exécute jamais : gardez donc la ligne d'activation et la ligne d'abonnement côte à côte dans votre méthode de configuration des colonnes.

Préférez la portée colonne. Dans le lab, vous ajoutez des colonnes explicites à `gridOperations` et mettez `UserPaint = true` sur les seules colonnes Health et Trend, puis vous vous abonnez une fois à `CellPaint` sur la grille. Cela vous donne un abonnement pour tout l'écran au lieu d'un par ligne, et cela laisse toutes les autres colonnes — identifiant, nom de machine, site, dernière mesure — en contenu de grille ordinaire, que le navigateur peut sélectionner, rechercher et mettre à l'échelle. Activer le dessin par

l'utilisateur pour la grille entière est la décision inverse : elle vous remet des cellules que vous n'aviez jamais l'intention de dessiner et demande au gestionnaire de les filtrer à chaque redessin.

Ce que l'événement vous remet, et ce qu'il ne vous remet pas

Comme `DataGridViewCellPaintEventArgs` dérive de `PaintEventArgs`, la partie dessin est celle qu'enseignait le Module 2 : `e.Graphics` est la surface, `e.ClipRectangle` est votre zone, et vous libérez chaque pen, brush et path que vous créez en laissant `e.Graphics` tranquille. La nouveauté, c'est l'aiguillage. `e.RowIndex` et `e.ColumnIndex` demandent tous deux une garde avant que vous ne touchiez à quoi que ce soit : un `RowIndex` négatif n'est pas une ligne de données, et une colonne à laquelle vous n'avez pas adhéré doit provoquer une sortie immédiate.

La partie compacte de l'API est celle qui prend les gens au dépourvu. Les arguments d'événement de Wisej n'exposent ni `PaintBackground()` ni `PaintContent()` : un exemple WinForms qui les appelle doit donc être réécrit plutôt que compilé, et c'est vous qui décidez de l'apparence du fond et du contenu de la cellule. Voyez-y un atout : quand le gestionnaire produit toute la cellule, il n'y a aucun état à moitié dessiné sur lequel raisonner.

```
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));
}
```

Cellules de progression, d'état et de sparkline

Trois formes couvrent la plupart des grilles d'exploitation. Une cellule de progression remplit une piste inactive d'un rectangle actif dont la largeur est proportionnelle à une valeur normalisée, et ne dessine un nombre que si la colonne est assez large. Une cellule d'état est une simple ellipse pleine, parfois assortie d'un court libellé ; c'est le visuel utile le moins coûteux de la grille, et souvent plus clair qu'une icône. Une sparkline normalise une petite série au rectangle de la cellule et relie les points avec `DrawLines`, comme le fait la colonne Trend dans le lab : son rôle est de résumer, jamais de reproduire une série de dix mille points dans quarante pixels.

Gardez les calculs hors du gestionnaire. Donnez à chaque visuel un helper qui prend un `Graphics`, un `Rectangle` et les valeurs dont il a besoin, et le gestionnaire se réduit à une garde et deux appels. Le moteur de rendu hors écran du Module 6 pourra alors appeler la même méthode pour placer la sparkline identique dans un PNG exporté. N'oubliez pas les états dont la grille est propriétaire : une cellule dessinée par votre code doit rester lisible quand sa ligne est sélectionnée ; choisissez donc des couleurs conscientes de la sélection, ou gardez le visuel sobre et laissez parler la mise en évidence de la ligne. Ne laissez jamais

la couleur être le seul porteur d'un état critique ; la même valeur doit aussi exister en texte ou en info-bulle.

Quand le dessin est le mauvais outil

Wisej.NET rend aussi du HTML dans les cellules grâce à `AllowHtml`, et la documentation positionne `CellPaint` comme la réponse pour un contenu trop complexe pour du HTML, ou pour du code de dessin .NET existant que vous voulez réutiliser. Choisissez le HTML lorsque le visuel peut s'exprimer avec du texte, des icônes, des spans et du CSS, lorsque la recherche, la sélection, l'accessibilité ou la mise à l'échelle du navigateur profitent d'un vrai contenu du DOM, ou lorsqu'il y a plusieurs milliers de cellules. Choisissez le dessin lorsque vous avez besoin d'une géométrie au pixel près, lorsque le visuel est compact et change rarement, ou lorsque le HTML finirait plus compliqué que le code de dessin.

Une seconde alternative n'a rien à voir avec le HTML : les types de cellules propres à la grille et sa capacité à héberger des contrôles. Un bouton, une case à cocher, un sélecteur de date ou une liste déroulante doivent rester de vraies cellules interactives. Le dessin est un modèle de rendu, pas un modèle d'interaction, et un bouton dessiné vous oblige à reconstruire à la main le retour au pointeur, l'accès clavier, le focus et les commandes. C'est pourquoi le lab construit l'indicateur d'état deux fois et vous demande d'écrire lequel vous livreriez.

Faire défiler mille lignes

Un `DataGridView` gère bien les grands jeux de données grâce à la liaison et aux modes virtuels, et un gestionnaire `CellPaint` négligent annule cet avantage en une ligne. Le défilement rappelle le gestionnaire pour chaque cellule nouvellement visible : tout ce qui est coûteux se multiplie donc par la vitesse à laquelle l'utilisateur fait glisser l'ascenseur.

- Ne faites jamais d'E/S, d'appel à une base de données ou à un service dans le gestionnaire.
- Ne chargez pas d'images et ne créez pas de polices par cellule ; construisez-les une fois et réutilisez-les.
- Dessinez le plus petit ensemble de colonnes possible, et réduisez le nombre de points des sparklines.
- Gardez la récupération d'état en $O(1)$: indexez la liste que vous avez liée, ou lisez la valeur de la cellule via la grille, mais ne parcourez jamais une collection par cellule.
- Mesurez avec un nombre de lignes réaliste — cent lignes ne prouvent rien pour mille.

Les mille lignes générées du lab existent pour ce dernier point. Liez-les, faites glisser l'ascenseur de haut en bas, et comparez ce que coûtent les deux colonnes dessinées face à la colonne HTML voisine. C'est cette mesure, et non une préférence, qui a sa place dans vos notes.



Réflexe dessin : la surface d'abord

Choisissez la surface de rendu avant d'écrire le moindre appel de dessin. Demandez-vous où les pixels sont produits et ce qui transite sur le réseau : un gestionnaire Paint de contrôle s'exécute sur le serveur et expédie une image, CellPaint fait de même à l'intérieur d'une cellule de grille, Canvas envoie des commandes de dessin que le navigateur exécute et ne conserve rien après un redimensionnement, et System.Drawing.Managed produit des octets d'image sans aucun contrôle. Gardez le modèle sémantique dans des objets C#, faites de chaque moteur de rendu une fonction déterministe de ce modèle, n'invalidiez que lorsqu'une dépendance visuelle a changé, et laissez toujours un équivalent textuel de tout ce qui compte et n'existe que sous forme de pixels.



Objectifs d'apprentissage

À la fin de ce module, vous saurez :

- ✓ Activer le dessin à la portée d'une cellule ou d'une colonne avec UserPaint.
- ✓ Utiliser DataGridViewCellPaintEventArgs en toute sûreté.
- ✓ Construire des cellules de progression, d'état et de sparkline.
- ✓ Comparer le dessin par l'utilisateur avec le HTML de cellule, les types de cellules intégrés, les contrôles en cellule et les images.
- ✓ Concevoir le dessin de cellules pour de grands jeux de données sans transformer le défilement en goulot d'étranglement du rendu.

Guide du lab / de l'examen

Ce que vous allez construire dans le lab pratique, l'approche suggérée et les livrables attendus.



Lab pratique

Donnez à VisualOperationsStudio une grille d'exploitation avec deux colonnes dessinées par votre code. Ajoutez des colonnes explicites à `gridOperations`, mettez `UserPaint = true` sur les seules colonnes `Health` et `Trend`, et gérez `CellPaint` une seule fois : sortez immédiatement lorsque `e.RowIndex` est négatif ou que `e.ColumnIndex` ne désigne aucune des deux colonnes visées, puis dessinez une barre de progression colorée selon les seuils pour `Health` et une sparkline en polyligne pour `Trend` à l'intérieur de `e.ClipRectangle`. Lisez les valeurs de chaque ligne depuis l'objet `MachineStatus` lié plutôt qu'en parcourant une liste dans le gestionnaire, respectez les couleurs de la ligne sélectionnée, et gardez le gestionnaire exempt d'effets de bord. Liez 1 000 lignes générées, faites défiler, puis construisez le même indicateur d'état une seconde fois sous forme de cellule `AllowHtml` afin de pouvoir écrire noir sur blanc lequel vous livreriez.

Point de départ

Vous ouvrez VisualOperationsStudio tel que le Module 2 l'a laissé : la `VisualOperationsPage` porte le contrôle `TelemetryGauge` et son helper `GaugeGeometry`, et la page a de la place pour une grille mais pas encore de données. Ajoutez une classe `MachineStatus` avec un identifiant, un nom, un site, un pourcentage `Health` et une courte liste de mesures récentes pour la tendance, ainsi qu'un générateur capable d'en produire un millier.

Approche suggérée

- ✓ Ajoutez `gridOperations` avec des colonnes explicites pour l'identifiant, la machine, le site, `Health` et `Trend`, liez d'abord dix lignes `MachineStatus`, et vérifiez que les colonnes brutes s'affichent avant qu'aucun dessin n'existe.
- ✓ Mettez `UserPaint = true` sur les seules colonnes `Health` et `Trend`, abonnez-vous une fois à `gridOperations.CellPaint`, et ouvrez le gestionnaire par une garde qui sort lorsque `e.RowIndex` est négatif ou que `e.ColumnIndex` ne correspond à aucune des deux colonnes visées.
- ✓ Écrivez `DrawHealthBar(Graphics, Rectangle, int)` et `DrawSparkline(Graphics, Rectangle, IReadOnlyList<double>)` comme des helpers purs qui dérivent chaque coordonnée du rectangle qu'on leur donne, choisissent la couleur de remplissage d'après les seuils d'avertissement et critique, et libèrent chaque pen et chaque brush qu'ils créent.
- ✓ Faites du gestionnaire un pur aiguillage : rétrécissez `e.ClipRectangle` avec `Rectangle.Inflate`, récupérez les valeurs de la ligne en une seule étape en $O(1)$, passez-les au helper, et gardez la ligne sélectionnée lisible en choisissant des couleurs qui contrastent encore avec le fond de sélection.
- ✓ Liez 1 000 lignes générées, faites glisser l'ascenseur de haut en bas, puis ajoutez une troisième colonne d'état qui rend le même état d'avertissement ou critique via `AllowHtml`, faites défiler à nouveau et consignez les deux résultats.

Livrables attendus

- ✓ gridOperations avec des colonnes explicites et UserPaint = true sur les seules colonnes Health et Trend
- ✓ Un unique gestionnaire CellPaint qui contrôle e.RowIndex et e.ColumnIndex avant de dessiner quoi que ce soit
- ✓ Cellule de progression colorée selon les seuils et cellule sparkline en polyligne dessinées à l'intérieur de e.ClipRectangle
- ✓ Valeurs de ligne lues depuis l'objet MachineStatus lié, sans recherche, allocation ni création de contrôle par cellule
- ✓ Une colonne d'état AllowHtml construite en parallèle, plus une note comparant les deux sur un défilement de 1 000 lignes

Questions de révision

- ✓ Un collègue colle un exemple WinForms de dessin de cellule qui commence par appeler `e.PaintBackground( ... )`. Expliquez ce que les arguments d'événement de Wisej.NET vous donnent à la place, et ce que le gestionnaire doit désormais décider lui-même.
- ✓ Votre helper de sparkline code en dur une largeur de tracé de 100 pixels et rend bien sur votre écran. Décrivez deux actions de l'utilisateur qui le cassent, et ce que le helper devrait lire à la place.
- ✓ La colonne d'état HTML et la colonne Health dessinée signalent toutes deux une machine critique. Défendez celle que vous livreriez pour une grille que le support recherche et lit avec un lecteur d'écran, et dites ce que vous perdriez dans un cas comme dans l'autre.



Réflexe dessin : la surface d'abord

Choisissez la surface de rendu avant d'écrire le moindre appel de dessin. Demandez-vous où les pixels sont produits et ce qui transite sur le réseau : un gestionnaire Paint de contrôle s'exécute sur le serveur et expédie une image, CellPaint fait de même à l'intérieur d'une cellule de grille, Canvas envoie des commandes de dessin que le navigateur exécute et ne conserve rien après un redimensionnement, et System.Drawing.Managed produit des octets d'image sans aucun contrôle. Gardez le modèle sémantique dans des objets C#, faites de chaque moteur de rendu une fonction déterministe de ce modèle, n'invalidiez que lorsqu'une dépendance visuelle a changé, et laissez toujours un équivalent textuel de tout ce qui compte et n'existe que sous forme de pixels.