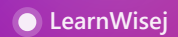


Maîtrise du DataGridView

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



Guide de la leçon

BindingSource et colonnes explicites, éditeurs personnalisés et cellules de commande, mode virtuel avec CellValueNeeded et un cache côté serveur, et HTML de cellule, contrôles de cellule et dessin de cellule utilisés à bon escient. Ce que cela signifie pour un développeur Wisej.NET — et comment aborder ce module.



Objectif en production

BindingSource et colonnes explicites, éditeurs personnalisés et cellules de commande, mode virtuel avec CellValueNeeded et un cache côté serveur, et HTML de cellule, contrôles de cellule et dessin de cellule utilisés à bon escient.

1. Le problème en production

DataGridView est le contrôle qui décide le plus souvent si une application métier paraît mature.

Commandes, factures, échéances, journaux, utilisateurs et documents finissent tous dans une grille, et les utilisateurs jugent l'application à la façon dont cette grille se charge, défile, s'édite et signale son état. Wisej.NET offre tout, du remplissage manuel au mode virtuel et au dessin de cellule ; la maîtrise consiste à choisir la bonne technique pour le volume de données et la tâche de l'utilisateur.

2. Concepts clés

Concept	Pourquoi c'est important
BindingSource	Un intermédiaire côté serveur entre les lignes du view model et la grille, qui lui fournit une source de données stable, une position courante et le tri sans coupler l'écran à la base de données.
AutoGenerateColumns	Crée une colonne pour chaque propriété publique du type lié ; pratique pour un prototype, dangereux en production, car un changement de modèle réorganise l'écran à votre insu.
colonne explicite	Une colonne que vous définissez avec liaison de champ, texte d'en-tête, largeur, alignement, format, lecture seule, mode de tri et type d'éditeur, pour que l'interface reste stable quand le modèle change.
mode virtuel	Avec <code>VirtualMode</code> activé, la grille ne connaît que son nombre de lignes et demande chaque valeur visible via <code>CellValueNeeded</code> , si bien que rien n'est créé pour les lignes que personne ne voit.
cache côté serveur	Un tampon de la taille d'une page qui alimente <code>CellValueNeeded</code> pour que le défilement ne sollicite pas la base à chaque cellule ; l'événement <code>DataRead</code> est le point d'accroche documenté pour le construire et l'entretenir.
éditeur de cellule personnalisé	Un contrôle affecté à la propriété <code>Editor</code> d'une colonne ; il doit renvoyer sa valeur dans la cellule, via <code>CellBeginEdit</code> et <code>CellEndEdit</code> s'il n'a pas d'événement de changement adapté.
HTML de cellule ou contrôle de cellule	Le HTML produit dans <code>CellFormatting</code> avec <code>AllowHtml</code> est un markup d'affichage quasi gratuit ; un contrôle intégré est un vrai contrôle serveur et un widget navigateur, avec un coût par cellule.

3. Le fil de la leçon

Trois façons de remplir, une pour la production

La grille peut afficher des données avec ou sans source de données sous-jacente. L'accès direct à `Rows` et `Columns` est le plus simple : vous ajoutez colonnes et lignes et maîtrisez chaque valeur ; cela convient à une petite table statique comme une liste de statuts. La liaison `DataSource` et `DataMember` remplit la grille à partir d'une liste, d'une `BindingSource` ou d'un jeu de données, et c'est le mode des données métier. Le troisième mode est la génération dynamique des colonnes : avec `AutoGenerateColumns` à true, la grille reconstruit ses colonnes chaque fois que la source de données change.

Les colonnes générées conviennent à un prototype de dix minutes et piègent un écran livré : ajoutez une propriété à `OrderRow`, renommez-en une autre, et la grille suit le modèle plutôt que la conception. La grille Orders d'Operations Console définit donc chaque colonne explicitement : `DataPropertyName`, `HeaderText`, `Width`, alignement et format dans `DefaultCellStyle`, `ReadOnly` quand la valeur est informative, `SortMode` et le type de colonne qui porte le bon éditeur. La liaison elle-même reste courte :

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

private void BindOrders(IEnumerable<OrderRow> rows)
{
    this.ordersGrid.AutoGenerateColumns = false;    // columns are designed, not discovered
    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;
}
```

Désactivez `AutoGenerateColumns` dès que l'écran est stable, et liez une collection de view models plutôt qu'une entité : un view model est taillé pour l'affichage et porte le texte du statut, un total formaté et un ID sans divulguer de détails de persistance.

Colonnes intégrées, éditeurs personnalisés et cellules de commande

Les valeurs typées méritent des colonnes typées. Les types de colonnes texte, case à cocher, liste déroulante, bouton, lien et image couvrent l'essentiel de la grille Orders, et Wisej.NET ajoute des colonnes fondées sur les éditeurs de date, numériques et masqués du Module 2. Une cellule de commande est une colonne de boutons ou de liens : la colonne *Ship* du lab déclenche `CellContentClick`, le gestionnaire lit l'ID de la ligne dans l'élément lié et appelle le service. Rien de la commande ne réside dans la cellule.

Quand aucun éditeur intégré ne convient, affectez un contrôle à la propriété `Editor` de la colonne ; l'exemple classique est un `MonthCalendar` sur une cellule de date d'échéance. L'avertissement important : l'éditeur doit mettre à jour correctement la valeur de la cellule. La grille attend `TextChanged` ou un événement de changement équivalent ; si votre contrôle n'en déclenche pas, gérez `CellBeginEdit` pour copier la valeur de la cellule dans l'éditeur et `CellEndEdit` pour recopier la valeur modifiée. Testez l'aller-retour avec Tab, Entrée et un clic ailleurs : un éditeur qui ne valide jamais est le bug classique des éditeurs personnalisés.

Mode virtuel, CellValueNeeded et le cache

Les gros volumes exigent une autre approche. Charger 100 000 commandes parce que la base peut les renvoyer n'est pas une stratégie d'interface : personne ne peut examiner autant de lignes, et chacune coûte de la mémoire serveur et du rendu navigateur. Filtrez d'abord, proposez une recherche et une pagination côté serveur, et réservez le mode virtuel au cas où l'utilisateur doit réellement faire défiler un très grand résultat. Là, vous définissez `VirtualMode` à true, donnez à la grille son `RowCount` et ses colonnes, et gérez `CellValueNeeded` pour fournir chaque valeur à la demande.

L'événement seul ne suffit pas. Un gestionnaire qui interroge la base dans `CellValueNeeded` s'exécute une fois par cellule visible, si bien qu'un seul défilement peut produire des centaines de requêtes.

L'`OrderCache` du lab conserve la page courante sur le serveur : `CellValueNeeded` lui demande une ligne

et une colonne, il sert depuis la mémoire les valeurs présentes et récupère une page entière en cas d'absence. La documentation Wisej.NET désigne l'événement `DataRead` comme l'endroit où construire et entretenir ce cache, car il vous indique quel bloc de lignes la grille s'apprête à lire. Profilez avec des volumes réalistes, pas avec les vingt lignes du service de démonstration, et demandez-vous ce qui casse en premier avec dix fois plus de données.

HTML de cellule, contrôles de cellule, dessin de cellule et grille composée

Le HTML de cellule est la meilleure option intermédiaire pour la richesse visuelle. Si un statut a besoin d'un badge, d'une icône, d'une barre de progression ou de plusieurs fragments de texte, activez `AllowHtml` et produisez le markup dans `CellFormatting`, comme dans l'extrait ci-dessus ; le statut brut reste dans le modèle et le HTML ne sert qu'à l'affichage. Traitez-le comme tel : encodez le texte utilisateur avant de l'insérer dans le markup pour qu'un nom de client ne devienne jamais du HTML exécutable, et gardez la validation métier hors du gestionnaire de mise en forme.

Les contrôles de cellule sont plus puissants et plus dangereux. Wisej.NET peut placer un vrai contrôle dans une cellule, voire sur plusieurs cellules : un bouton dans une ligne, un curseur sur plusieurs colonnes, un mini-aperçu. Chacun est un contrôle serveur et un widget navigateur, donc un contrôle dans chaque cellule d'une grille de 10 000 lignes représente 10 000 contrôles. Réservez-les à la ligne sélectionnée, aux lignes de synthèse, à une zone de commandes ou à une petite grille. Le dessin de cellule se situe à l'extrémité de l'échelle : activez le dessin utilisateur sur la cellule et gérez `CellPaint` quand l'affichage exige un dessin personnalisé ou réutilise du code de dessin .NET existant, pas pour un style que les thèmes, les styles de cellule ou le HTML couvrent déjà.

Enfin, le `DataGridView` composable se comporte comme un conteneur : il héberge des contrôles associés, comme un panneau de filtre, une barre d'état, un navigateur ou un arbre, ancrés à l'intérieur ou autour de lui et ordonnés avec `BringToFront` et `SendToBack`. Toute l'API de la grille reste ainsi visible au lieu d'être cachée derrière un UserControl ; la barre de filtre et la barre d'état du lab sont construites de cette façon.

- Des colonnes explicites et une BindingSource ou un view model pour chaque écran de production ; AutoGenerateColumns désactivé dès que l'écran est stable.
- Filtrer avant d'afficher ; mode virtuel et cache pour les ensembles réellement volumineux.
- Le HTML de cellule avant les contrôles intégrés ; les contrôles intégrés avec parcimonie ; le dessin de cellule seulement quand le style, le HTML et les contrôles normaux ne suffisent pas.
- Profiler avec des données réalistes, pas avec des données d'exemple.



Pièges courants

`AutoGenerateColumns` laissé activé en production. Charger tous les enregistrements au démarrage. Interroger la base depuis `CellValueNeeded` pour chaque cellule, sans cache. Intégrer des contrôles dans chaque ligne d'une énorme grille. Mélanger validation métier et mise en forme visuelle. Traiter une grille comme une table de base de données plutôt que comme une vue orientée tâche.



Réflexe bibliothèque de contrôles

Choisissez le contrôle natif le plus simple qui exprime la tâche de l'utilisateur avant d'écrire du code personnalisé. Les contrôles natifs offrent déjà le support au design, les thèmes, les événements et l'état serveur ; tournez-vous ensuite vers un extender ou un UserControl, et ne créez un widget JavaScript que si aucun contrôle natif ne convient.



Objectifs d'apprentissage

À la fin de ce module, vous saurez :

- ✓ Lier un DataGridView à une BindingSource et maîtriser délibérément AutoGenerateColumns.
- ✓ Utiliser les colonnes intégrées et des éditeurs personnalisés pour les valeurs typées et les cellules de commande.
- ✓ Mettre en œuvre le mode virtuel, CellValueNeeded, les notions de DataRead et de cache, et les limites de performance liées aux gros volumes.
- ✓ Utiliser à bon escient le HTML de cellule, les contrôles de cellule, le dessin de cellule et les grilles composées.

Guide du lab / de l'examen

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



Lab pratique

Créez la grille Orders d'Operations Console : un DataGridView lié à une BindingSource avec des colonnes explicites et AutoGenerateColumns maîtrisé, une cellule de statut rendue avec AllowHtml et CellFormatting, une colonne d'éditeur personnalisé ou de commande, une version en mode virtuel ou adossée à un cache pour le chemin des gros volumes, et une barre de filtre et une barre d'état composées, ancrées dans la grille ou autour.

Point de départ

Ouvrez la solution `OperationsConsole` du Module 4 et sélectionnez la page provisoire DataGridView dans le shell. L'explorateur de documents, le `CustomerEditor` et le shell de navigation existent déjà ; la page Orders est encore une `TabPage` vide, et un petit `OrderService` en mémoire qui renvoie des view models `OrderRow` suffit pour mener chaque étape de ce lab.

Approche suggérée

- ✓ Ajoutez `ordersGrid` et une BindingSource `ordersSource`, définissez `AutoGenerateColumns` à false, et déclarez chaque colonne explicitement (numéro de commande, client, date d'échéance, total, statut) avec texte d'en-tête, largeur, alignement, format et lecture seule choisis délibérément.
- ✓ Marquez la colonne de statut `AllowHtml` et gérez `CellFormatting` pour transformer le statut brut en badge encodé ; laissez la valeur du modèle intacte et définissez `FormattingApplied`.
- ✓ Ajoutez soit un éditeur MonthCalendar sur la colonne de date d'échéance via la propriété `Editor` de la colonne (avec `CellBeginEdit` / `CellEndEdit` si la valeur ne fait pas l'aller-retour correctement), soit une colonne de boutons dont le `CellContentClick` appelle le service avec l'ID de la commande.
- ✓ Construisez le chemin des gros volumes : `VirtualMode` à true, `RowCount` fourni par le service, un `OrderCache` qui sert `CellValueNeeded` depuis une page en mémoire et récupère une nouvelle page en cas d'absence, en utilisant `DataRead` pour précharger le bloc que la grille s'apprête à afficher.
- ✓ Composez la grille : ancrez une barre de filtre (zone de recherche, ComboBox de statut, bouton Apply) au-dessus ou à l'intérieur de la grille et une barre d'état en dessous qui affiche le nombre de lignes, la commande sélectionnée et la dernière actualisation ; ordonnez-les avec `BringToFront` et `SendToBack`.

Livrables attendus

- ✓ DataGridView Orders avec des colonnes explicites liées à une BindingSource
- ✓ Cellule de statut rendue avec AllowHtml et CellFormatting
- ✓ Une colonne d'éditeur personnalisé ou de commande

- ✓ Version de la grille en mode virtuel ou adossée à un cache pour les gros volumes
- ✓ Barre de filtre et barre d'état composées, ancrées dans la grille ou autour

Questions de révision

- ✓ Si `OrderRow` gagne deux nouvelles propriétés au prochain sprint, qu'est-ce qui change à l'écran avec des colonnes générées, et qu'est-ce qui change avec vos colonnes explicites ?
- ✓ Votre gestionnaire `CellValueNeeded` fonctionne avec 200 commandes. Qu'arrive-t-il à la base de données quand un utilisateur fait glisser la barre de défilement sur 200 000, et quelle partie de votre conception absorbe cette charge ?
- ✓ Pourquoi le badge de statut a-t-il sa place dans `CellFormatting` plutôt que dans le modèle `OrderRow`, et que se passerait-il si un nom de client était placé dans ce HTML sans encodage ?



Réflexe bibliothèque de contrôles

Choisissez le contrôle natif le plus simple qui exprime la tâche de l'utilisateur avant d'écrire du code personnalisé. Les contrôles natifs offrent déjà le support au design, les thèmes, les événements et l'état serveur ; tournez-vous ensuite vers un extender ou un UserControl, et ne créez un widget JavaScript que si aucun contrôle natif ne convient.