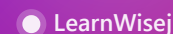


Maîtriser l'événement Paint des contrôles

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



Guide de la leçon

Les contrôles dessinés par le serveur, faits correctement : `PaintEventArgs.Graphics` et `ClipRectangle`, la géométrie séparée du dessin, la libération de chaque pen, brush et font que vous créez, des setters de propriétés qui appellent `Invalidate()` exactement une fois, et le budget de coût de dessin qui décide du moment où un contrôle thémé est la meilleure réponse. 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

Les contrôles dessinés par le serveur, faits correctement : `PaintEventArgs.Graphics` et `ClipRectangle`, la géométrie séparée du dessin, la libération de chaque pen, brush et font que vous créez, des setters de propriétés qui appellent `Invalidate()` exactement une fois, et le budget de coût de dessin qui décide du moment où un contrôle thémé est la meilleure réponse.

1. Le problème en production

Le Module 1 a placé une même mesure de télémétrie sur quatre surfaces pour vous montrer où chaque pixel est produit. VisualOperationsStudio veut désormais cette mesure à six endroits, et recopier un gestionnaire `Paint` dans six panneaux, c'est ainsi qu'un écran se retrouve avec six jauges légèrement différentes, six pens jamais libérés et un redessin dès qu'un champ sans rapport change. La solution est d'arrêter d'écrire des gestionnaires et d'écrire un contrôle : un `TelemetryGauge` unique dont les propriétés publiques sont la seule chose capable de changer l'image, et dont vous pouvez énoncer le coût en kilo-octets.

2. Concepts clés

Concept	Pourquoi c'est important
<code>Wisej.Web.PaintEventArgs</code>	L'argument de l'événement <code>Paint</code> ; il vous remet un <code>System.Drawing.Graphics</code> que vous n'avez pas créé et un rectangle à l'intérieur duquel dessiner, et votre code s'exécute sur le serveur.
<code>ClipRectangle</code>	La région que Wisej.NET a prévue pour cette opération de dessin ; la mise en page peut consulter la taille cliente du contrôle, mais le dessin reste borné par le rectangle qui vous a été remis.
Helper <code>GaugeGeometry</code>	Une fonction pure d'un <code>Rectangle</code> et d'une valeur normalisée vers les limites de l'anneau, les angles de balayage, les points de l'aiguille et un rectangle de libellé, afin que les calculs puissent être vérifiés sans contrôle vivant.
<code>Invalidate()</code>	Le seul appel qui demande à Wisej.NET de refaire le rendu de la surface du contrôle ; un setter de propriété l'émet exactement une fois, et uniquement lorsque le changement peut modifier un pixel.
Propriété des ressources	Le gestionnaire libère chaque <code>Pen</code> , <code>Brush</code> , <code>Font</code> , <code>GraphicsPath</code> et <code>Bitmap</code> qu'il crée, et ne libère jamais le <code>Graphics</code> fourni par le framework.
Budget de coût de dessin	Trois nombres par contrôle dessiné : le CPU serveur consacré au dessin, la mémoire serveur de l'image temporaire, et les octets que le navigateur doit récupérer et décoder.
<code>AccessibleDescription</code>	Porte la même mesure sous forme de texte, pour qu'une valeur qui n'existe que comme un arc dessiné reste disponible à un utilisateur qui ne verra jamais cet arc.

3. Le fil de la leçon

Le gestionnaire, la surface et le rectangle qui vous est remis

Un gestionnaire `Paint` de Wisej.NET paraît familier parce que l'objet de dessin est bel et bien un `System.Drawing.Graphics` , mais le runtime autour de lui n'est pas un bureau. Votre gestionnaire s'exécute sur le serveur et l'image qu'il produit devient l'arrière-plan que le navigateur affiche. Le bon modèle mental est que vous peignez un bitmap côté serveur, et c'est aussi pourquoi un redessin porte un prix réseau.

L'événement fournit un `Wisej.Web.PaintEventArgs` , et deux membres portent tout ce dont vous avez besoin. `Graphics` est la surface active : vous ne l'avez pas allouée et vous ne devez pas la libérer. `ClipRectangle` est la région que Wisej.NET a prévue pour cette opération. Raisonniez sur la mise en page avec le rectangle client du contrôle, mais gardez le dessin borné par le rectangle que vous avez en main.

Les deux premières lignes d'un gestionnaire de production sont donc délibérément ennuyeuses : vérifiez que le rectangle est assez grand pour dessiner quoi que ce soit, puis passez-le au helper de géométrie. Un contrôle mis en page à une largeur nulle, ou comprimé pendant une recomposition responsive, se

contente de sortir. Aucun chemin de dessin n'est assez important pour justifier qu'une exception atteigne l'écran d'un utilisateur.

Formes, tracés, texte et contenu transformé

`Graphics` est une API de dessin immédiate et à état, et les familles dont la jauge a besoin sont peu nombreuses. Les contours viennent de `DrawArc`, `DrawLine` et `DrawPolygon` ; la géométrie pleine de `FillPie`, `FillEllipse` et `FillRectangle` ; le texte de `DrawString` avec une `Font`, un `Brush` et un `StringFormat` ; les formes composées et les courbes d'un `GraphicsPath` de `System.Drawing.Drawing2D`. Le découpage et les transformations complètent l'ensemble.

C'est sur les transformations que les gestionnaires dérapent le plus souvent. Appliquer

`TranslateTransform` et `RotateTransform` pour faire pivoter l'aiguille est facile ; les défaire avant de dessiner la légende l'est moins. Ouvrez un conteneur graphique avec `BeginContainer()`, dessinez dans un `try`, et placez `EndContainer(state)` dans le `finally` : l'implémentation de dessin managé fournie par Wisej.NET ne possède pas la paire `Graphics.Save / Restore`, et le conteneur en est l'équivalent. La même discipline réapparaît au Module 4 sous la forme de la paire `Save()` et `Restore()` du Canvas, qui porte bien ces noms.

Le texte mérite le même soin : `MeasureString` et l'alignement par `StringFormat` survivent à une légende tradatée, un décalage codé en dur non. Les images demandent une règle supplémentaire, parce que `Paint` prend le pas sur l'arrière-plan. Un filigrane se dessine explicitement avec `DrawImage`, à partir d'une image décodée une fois et possédée par le contrôle, jamais depuis un fichier ouvert dans le gestionnaire.

Un contrôle réutilisable dont les propriétés l'invalident correctement

Dans le lab, `TelemetryGauge` devient un véritable contrôle plutôt qu'un gestionnaire greffé sur un panneau. Le rendu vit à l'intérieur de la classe et les consommateurs définissent des propriétés métier — `Value`, `Minimum`, `Maximum`, `WarningThreshold`, `CriticalThreshold`. Wisej.NET expose aussi sur `Control` un point d'accroche protégé `OnWebPaint(Graphics)` destiné aux auteurs de bibliothèques, tandis que l'événement public `Paint` reste le mécanisme de composition le plus simple.

Chaque setter suit une même recette : borner dans une plage légale, comparer à ce qui est stocké, sortir si rien n'a changé, affecter, puis invalider une fois. Une valeur hors plage est bornée plutôt que rejetée, et une plage inversée est normalisée plutôt que laissée produire un balayage négatif. Ensuite, le code de dessin peut supposer que ses entrées sont saines.

```

public float Value
{
    get => _value;
    set
    {
        var low = Math.Min(Minimum, Maximum); // une plage inversée est
        var high = Math.Max(Minimum, Maximum); // normalisée, pas rejetée
        var next = Math.Clamp(value, low, high);
        if (Math.Abs(_value - next) < 0.001f)
            return; // nothing visual changed
        _value = next;
        AccessibleDescription = $"{Caption}: {_value:0.#}";
        Invalidate(); // exactly once per real change
    }
}

```

La seconde moitié du contrôle, c'est `GaugeGeometry` : un helper qui prend un rectangle et la valeur normalisée et renvoie les limites de l'anneau, un angle de départ, un balayage, les points de l'aiguille et le rectangle du libellé. Il ne touche à aucun `Graphics`, vous pouvez donc faire des assertions sur sa sortie dans un test ordinaire — et rendre la même jauge hors écran au Module 6 ne coûte alors presque rien.

Qui possède quel objet libérable

La propriété des ressources est ce qui épargne les ennuis à un contrôle dessiné sur un serveur partagé. Le gestionnaire possède tout ce qu'il crée — pens, brushes unis et à dégradé, fonts, paths, éventuel bitmap — et une déclaration `using` libère chacun à la fin de la méthode. Il ne possède pas `e.Graphics`, et le libérer est l'erreur classique.

Deux habitudes voisines comptent tout autant. Les statiques du framework comme `Pens.Red` et `Brushes.Black` sont partagées et ne doivent pas être touchées. Et un pen mutable garé dans un champ `static` est pire qu'une fuite : un serveur Wisej.NET gère de nombreuses sessions et peut effectuer plusieurs rendus à la fois, l'état de rendu reste donc local, sauf si vous concevez délibérément un cache immuable et sûr en concurrence. Le chemin de dessin n'ouvre jamais non plus de connexion à une base, n'appelle aucun service et ne démarre aucune minuterie.

Le budget de coût, et la mesure que personne ne voit

Un contrôle dessiné dépense à trois endroits : le CPU serveur pendant le dessin, la mémoire serveur pour l'image temporaire, et le réseau plus le décodage navigateur pour l'image envoyée. Une barre d'état et une carte de chaleur plein écran ne sont toutes deux que des images pour le navigateur, et leurs coûts n'ont rien de comparable : mesurez donc les trois lorsqu'une surface grandit. C'est ce que consigne `PaintCost.md`.

La politique d'invalidation est l'autre moitié du budget. Écrivez noir sur blanc les dépendances visuelles — en général la valeur, la légende, la palette et la taille — et ne redessinez que pour celles-là. Si une valeur bouge dix fois à l'intérieur d'une requête, mettez l'état à jour dix fois mais visez une seule mise à jour visuelle à la fin, et ne faites jamais tourner une minuterie côté serveur dans le seul but d'animer une image. Lorsque les chiffres cessent de justifier les pixels, la réponse honnête est un contrôle thémé, un visuel HTML ou une extension de graphiques.

Enfin, un nombre dessiné n'est pas un nombre accessible. Un utilisateur qui s'appuie sur une technologie d'assistance reçoit une image : la jauge publie donc sa mesure via [AccessibleDescription](#) et la page conserve un équivalent textuel à côté des graphiques. Ajoutez un repli pour une police ou une image manquante, et un état de diagnostic minimal au lieu d'une exception, et le contrôle peut être confié sans risque à une autre équipe.



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 :

- ✓ Gérer l'événement Paint et travailler avec PaintEventArgs.Graphics et ClipRectangle.
- ✓ Dessiner des formes, des tracés, du texte, des images, des dégradés et du contenu transformé avec System.Drawing.
- ✓ Construire un contrôle dessiné personnalisé et réutilisable dont les propriétés publiques l'invalident correctement.
- ✓ Appliquer les règles de durée de vie des ressources pour les pens, brushes, fonts, images et objets graphiques.
- ✓ Reconnaître les cas où le coût du dessin, le transfert du PNG ou l'accessibilité rendent une autre technique préférable.

Guide du lab / de l'examen

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

🎯 Lab pratique

Ajoutez à VisualOperationsStudio un contrôle TelemetryGauge réutilisable qui dessine un anneau, des zones de seuil colorées, une aiguille, une valeur numérique et une légende. Dotez-le de propriétés Value, Minimum, Maximum, WarningThreshold et CriticalThreshold dont les setters bornent la valeur reçue, sortent immédiatement si rien n'a changé et appellent Invalidate() une seule fois. Placez les calculs de coordonnées dans un helper pur GaugeGeometry qui prend un Rectangle et renvoie les limites de l'anneau, les angles de balayage, les points de l'aiguille et le rectangle du libellé, puis dessinez à partir de ces valeurs avec des pens, des brushes et un GraphicsPath portés par des using, sans toucher au Graphics fourni. Gérez sans lever d'exception un contrôle de taille nulle, une plage inversée et une valeur hors plage, et exposez la mesure via AccessibleDescription pour qu'elle n'existe pas uniquement sous forme de pixels.

Point de départ

Vous ouvrez VisualOperationsStudio tel que le Module 1 l'a laissé : VisualOperationsPage détient déjà un unique modèle TelemetrySample et affiche sa valeur sur quatre surfaces, dont un Panel muni d'un gestionnaire Paint écrit en ligne qui dessine une barre. Dans ce lab, ce gestionnaire en ligne est remplacé : vous ajoutez une classe TelemetryGauge au projet, posez trois instances sur la page, et liez chacune à une mesure issue du même modèle.

Approche suggérée

- ✓ Ajoutez une classe TelemetryGauge dérivant d'un contrôle Wisej.NET, donnez-lui ses champs de stockage ainsi que Value, Minimum, Maximum, WarningThreshold, CriticalThreshold et Caption, et écrivez un setter qui borne, compare, sort immédiatement et appelle Invalidate() une fois — puis reproduisez cette forme sur les autres.
- ✓ Écrivez GaugeGeometry comme un helper pur : une méthode qui prend le Rectangle dessinable et la valeur normalisée et renvoie les limites de l'anneau, les angles de départ et de balayage, les points de l'aiguille et le rectangle du libellé, sans aucune référence à Graphics nulle part dans le fichier.
- ✓ Attachez le gestionnaire Paint (ou surchargez le point d'accroche de dessin) et dessinez strictement à partir du résultat de la géométrie : la piste inactive, les zones d'avertissement et critique, l'arc actif, l'aiguille construite en GraphicsPath, puis la valeur numérique et la légende mesurées avec MeasureString et un StringFormat.
- ✓ Enveloppez chaque pen, brush, font et path que vous créez dans une déclaration using, utilisez les statiques du framework comme Brushes.Black sans les libérer, et encadrez le RotateTransform de l'aiguille par Graphics.BeginContainer() et EndContainer(state) pour que la légende soit dessinée en coordonnées non transformées.
- ✓ Protégez les cas limites : sortez immédiatement lorsque le rectangle est trop petit pour dessiner, normalisez une plage inversée, bornez une valeur qui en sort, définissez

`AccessibleDescription` depuis le setter, puis redimensionnez le navigateur, changez la valeur depuis un bouton et consignez la taille de l'image et la fréquence des redessins dans `PaintCost.md`.

Livrables attendus

- ✓ Contrôle TelemetryGauge avec des propriétés Value, Minimum, Maximum et de seuil qui bornent la valeur et appellent Invalidate() une fois par changement réel
- ✓ Helper pur GaugeGeometry renvoyant les limites de l'anneau, les angles de balayage, les points de l'aiguille et le rectangle du libellé
- ✓ Gestionnaire Paint qui libère chaque pen, brush, font et path qu'il crée et ne libère jamais e.Graphics
- ✓ Garde-fous pour une taille nulle, une plage inversée et des valeurs hors plage, avec une AccessibleDescription portant la mesure
- ✓ PaintCost.md indiquant la taille du redessin, sa fréquence et les cas où un contrôle théme coûterait moins cher

Questions de révision

- ✓ Un collègue déplace les calculs de l'anneau et de l'aiguille directement dans le gestionnaire `Paint` parce que « ce n'est que quelques lignes ». Quelles deux choses deviennent ensuite impossibles à faire à bas coût, et laquelle des deux le Module 6 exige-t-il ?
- ✓ Votre jauge est liée à une fiche machine qui porte aussi un nom d'opérateur et une date de maintenance. Si ces champs changent, le contrôle doit-il s'invalider ? Répondez en termes de dépendances visuelles, et non en termes de modèle.
- ✓ La même jauge va figurer sur un tableau de bord qui en affiche soixante et se rafraîchit toutes les deux secondes. Que mesureriez-vous en premier, et que changeriez-vous si les mesures étaient mauvaises ?



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.