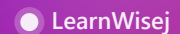


Dominar el evento Paint de los controles

Guía de la lección · Guía del laboratorio / examen



Guía de la lección

Controles dibujados en el servidor, bien hechos: PaintEventArgs.Graphics y ClipRectangle, la geometría separada del dibujo, liberar todos los pens, brushes y fuentes que creas, setters de propiedades que llaman a Invalidate() exactamente una vez, y el presupuesto de coste de dibujado que decide cuándo un control con tema es la mejor respuesta. Qué significa esto para un desarrollador .NET con experiencia que ya construye pantallas Wisej.NET y ahora necesita gráficos personalizados, y cómo leer este módulo.



Enfoque de producción

Controles dibujados en el servidor, bien hechos: PaintEventArgs.Graphics y ClipRectangle, la geometría separada del dibujo, liberar todos los pens, brushes y fuentes que creas, setters de propiedades que llaman a Invalidate() exactamente una vez, y el presupuesto de coste de dibujado que decide cuándo un control con tema es la mejor respuesta.

1. El problema en producción

El Módulo 1 puso una lectura de telemetría en cuatro superficies para que vieras dónde se produce cada píxel. VisualOperationsStudio quiere ahora esa lectura en seis sitios, y copiar un controlador de `Paint` en seis paneles es la forma de acabar con seis indicadores ligeramente distintos, seis pens que nunca se liberan y un redibujado cada vez que cambia un campo que no tiene nada que ver. La solución es dejar de escribir controladores y escribir un control: un único `TelemetryGauge` cuyas propiedades públicas sean lo único capaz de cambiar su imagen, y cuyo coste puedas expresar en kilobytes.

2. Conceptos clave

Concepto	Por qué importa
<code>Wisej.Web.PaintEventArgs</code>	El argumento del evento <code>Paint</code> ; te entrega un <code>System.Drawing.Graphics</code> que no has creado y un rectángulo dentro del cual dibujar, y tu código se ejecuta en el servidor.
<code>ClipRectangle</code>	La región que Wisej.NET ha reservado para esta operación de dibujo; el diseño puede consultar el tamaño de cliente del control, pero el dibujo sigue acotado por el rectángulo que se te ha dado.
Ayudante <code>GaugeGeometry</code>	Una función pura que va de un <code>Rectangle</code> más un valor normalizado a los límites del anillo, los ángulos de barrido, los puntos de la aguja y un rectángulo de etiqueta, de modo que los cálculos se pueden comprobar sin un control vivo.
<code>Invalidate()</code>	La única llamada que pide a Wisej.NET volver a renderizar la superficie del control; el setter de una propiedad la hace exactamente una vez, y solo cuando el cambio puede alterar un píxel.
Propiedad de los recursos	El controlador libera todos los <code>Pen</code> , <code>Brush</code> , <code>Font</code> , <code>GraphicsPath</code> y <code>Bitmap</code> que crea, y nunca libera el <code>Graphics</code> que ha proporcionado el framework.
Presupuesto de coste de dibujado	Tres números por control dibujado: la CPU de servidor que se gasta dibujando, la memoria de servidor para la imagen temporal y los bytes que el navegador tiene que descargar y decodificar.
<code>AccessibleDescription</code>	Lleva la misma lectura como texto, de modo que un valor que existe como un arco dibujado siga estando disponible para quien nunca ve ese arco.

3. Desarrollo de la lección

El controlador, la superficie y el rectángulo que se te ha dado

Un controlador de `Paint` de Wisej.NET resulta familiar porque el objeto de dibujo es realmente `System.Drawing.Graphics` , pero el entorno de ejecución que lo rodea no es un escritorio. Tu controlador se ejecuta en el servidor y la imagen que produce se convierte en el fondo que muestra el navegador. El modelo mental útil es que estás dibujando un mapa de bits del lado del servidor, que es también por lo que un redibujado tiene un precio en red.

El evento proporciona un `Wisej.Web.PaintEventArgs` , y dos miembros llevan todo lo que necesitas. `Graphics` es la superficie activa: no la has reservado tú y no debes liberarla. `ClipRectangle` es la región que Wisej.NET ha reservado para esta operación. Razona sobre el diseño con el rectángulo de cliente del control, pero mantén el dibujo acotado al rectángulo que tienes en las manos.

Eso hace que las dos primeras líneas de un controlador de producción sean deliberadamente aburridas: comprueba que el rectángulo es lo bastante grande para dibujar algo y después pásaselo al ayudante de geometría. Un control dispuesto con ancho cero, o comprimido durante una redistribución adaptable,

simplemente sale. Ninguna ruta de dibujo es tan importante como para justificar que una excepción llegue a la pantalla de un usuario.

Formas, trazados, texto y contenido transformado

`Graphics` es una API de dibujo inmediata y con estado, y las familias que necesita el indicador son pocas. Los contornos salen de `DrawArc`, `DrawLine` y `DrawPolygon`; la geometría rellena, de `FillPie`, `FillEllipse` y `FillRectangle`; el texto, de `DrawString` con una `Font`, un `Brush` y un `StringFormat`; las formas compuestas y las curvas, de un `GraphicsPath` en `System.Drawing.Drawing2D`. El recorte y las transformaciones completan el conjunto.

Las transformaciones son el punto en el que los controladores suelen fallar. Aplicar `TranslateTransform` y `RotateTransform` para girar la aguja es fácil; deshacerlas antes de dibujar el rótulo, no. Abre un contenedor gráfico con `BeginContainer()`, dibuja dentro de un `try` y pon `EndContainer(state)` en el `finally`: la implementación de dibujo gestionado que incluye Wisej.NET no tiene la pareja `Graphics.Save / Restore`, y el contenedor es su equivalente. La misma disciplina reaparece en el Módulo 4 con la pareja `Save()` y `Restore()` del Canvas, que sí lleva esos nombres.

El texto merece el mismo cuidado: `MeasureString` y la alineación de `StringFormat` sobreviven a un rótulo trasladado; un desplazamiento fijo en el código, no. Las imágenes necesitan una regla adicional, porque `Paint` toma el control del fondo. Una marca de agua se dibuja de forma explícita con `DrawImage`, a partir de una imagen decodificada una sola vez y propiedad del control, nunca desde un archivo abierto dentro del controlador.

Un control reutilizable cuyas propiedades lo invalidan correctamente

En el laboratorio, `TelemetryGauge` pasa a ser un control de verdad y no un controlador atornillado a un panel. El renderizado vive dentro de la clase y quien lo consume establece propiedades de dominio: `Value`, `Minimum`, `Maximum`, `WarningThreshold`, `CriticalThreshold`. Wisej.NET expone además un enganche protegido `OnWebPaint(Graphics)` en `Control` para quienes escriben bibliotecas, mientras que el evento público `Paint` sigue siendo el mecanismo de composición más sencillo.

Todos los setters siguen la misma receta: acotar a un rango válido, comparar con lo almacenado, salir cuando nada ha cambiado, asignar y después invalidar una sola vez. Un valor fuera de rango se acota en lugar de rechazarse, y un rango invertido se normaliza en lugar de dejar que produzca un barrido negativo. A partir de ahí, el código de dibujo puede dar por supuesto que sus entradas son razonables.

```

public float Value
{
    get => _value;
    set
    {
        var low = Math.Min(Minimum, Maximum); // an inverted range is
        var high = Math.Max(Minimum, Maximum); // normalised, not thrown on
        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 segunda mitad del control es `GaugeGeometry` : un ayudante que recibe un rectángulo y el valor normalizado y devuelve los límites del anillo, un ángulo inicial, un barrido, los puntos de la aguja y el rectángulo de la etiqueta. No toca ningún `Graphics` , así que puedes hacer aserciones sobre su salida en una prueba corriente, y renderizar ese mismo indicador fuera de pantalla en el Módulo 6 no costará prácticamente nada.

Quién es dueño de cada objeto liberable

La propiedad de los recursos es lo que mantiene a un control dibujado fuera de problemas en un servidor compartido. El controlador es dueño de lo que crea —pens, brushes sólidos y de degradado, fuentes, paths, cualquier mapa de bits— y una declaración `using` libera cada uno al final del método. No es dueño de `e.Graphics` , y liberarlo es el error clásico.

Dos hábitos relacionados importan igual. Los estáticos del framework, como `Pens.Red` y `Brushes.Black` , son compartidos y hay que dejarlos en paz. Y un pen mutable aparcado en un campo `static` es peor que una fuga: un servidor Wisej.NET atiende muchas sesiones y puede renderizar para varias a la vez, así que el estado de renderizado se queda local salvo que diseñes a propósito una caché inmutable y segura para hilos. La ruta de dibujo tampoco abre nunca una conexión a la base de datos, ni llama a un servicio, ni inicia un temporizador.

El presupuesto de coste y la lectura que nadie ve

Un control dibujado gasta en tres sitios: CPU de servidor mientras dibuja, memoria de servidor para la imagen temporal, y red más decodificación en el navegador para la imagen enviada. Una barra de estado y un mapa de calor a ventana completa son, para el navegador, simplemente imágenes, y sus costes no son ni de lejos comparables, así que mide los tres cuando una superficie crezca. Eso es lo que registra `PaintCost.md` .

La política de invalidación es la otra mitad del presupuesto. Escribe las dependencias visuales — normalmente el valor, el rótulo, la paleta y el tamaño— y redibuja solo por ellas. Si un valor se mueve diez veces dentro de una misma petición, actualiza el estado diez veces pero busca una única actualización visual al final, y nunca ejecutes un temporizador en el servidor solo para animar una imagen. Cuando los

números dejan de justificar los píxeles, la respuesta honesta es un control con tema, un elemento visual en HTML o una extensión de gráficos.

Por último, un número dibujado no es un número accesible. Quien depende de una tecnología de asistencia recibe una imagen, así que el indicador publica su lectura mediante `AccessibleDescription` y la página mantiene un equivalente textual junto a los gráficos. Añade una alternativa para una fuente o una imagen ausentes y un estado de diagnóstico mínimo en lugar de una excepción, y el control estará listo para entregárselo a otro equipo.



Regla de dibujo: primero la superficie

Elige la superficie de renderizado antes de escribir una sola llamada de dibujo. Pregúntate dónde se producen los píxeles y qué cruza la red: un controlador de Paint de un control se ejecuta en el servidor y envía una imagen, CellPaint hace lo mismo dentro de una celda de cuadrícula, Canvas envía comandos de dibujo que ejecuta el navegador y no conserva nada tras un redimensionado, y `System.Drawing.Managed` produce bytes de imagen sin ningún control de por medio. Mantén el modelo semántico en objetos C#, haz que cada renderizador sea una función determinista de ese modelo, invalida solo cuando haya cambiado una dependencia visual, y deja siempre un equivalente textual de todo lo importante que exista únicamente como píxeles.



Objetivos de aprendizaje

Al final de este módulo podrás:

- ✓ Gestionar el evento Paint y trabajar con `PaintEventArgs.Graphics` y `ClipRectangle`.
- ✓ Dibujar formas, trazados, texto, imágenes, degradados y contenido transformado con `System.Drawing`.
- ✓ Construir un control dibujado personalizado y reutilizable cuyas propiedades públicas lo invaliden correctamente.
- ✓ Aplicar las reglas de tiempo de vida de los recursos para pens, brushes, fuentes, imágenes y objetos gráficos.
- ✓ Reconocer cuándo el coste del dibujo, la transferencia del PNG o la accesibilidad hacen preferible otra técnica.

Guía del laboratorio / examen

Lo que construirás en el laboratorio práctico, el enfoque sugerido y los entregables requeridos.



Laboratorio práctico

Añade a VisualOperationsStudio un control TelemetryGauge reutilizable que dibuje un anillo, zonas de umbral coloreadas, una aguja, un valor numérico y un rótulo. Dale propiedades Value, Minimum, Maximum, WarningThreshold y CriticalThreshold cuyos setters acoten la entrada, salgan de inmediato cuando nada haya cambiado y llamen a Invalidate() una sola vez. Pon los cálculos de coordenadas en un ayudante puro GaugeGeometry que reciba un Rectangle y devuelva los límites del anillo, los ángulos de barrido, los puntos de la aguja y el rectángulo de la etiqueta; después dibuja a partir de esos valores con pens, brushes y un GraphicsPath declarados con using, sin tocar el Graphics que se te entrega. Gestiona un control de tamaño cero, un rango invertido y un valor fuera de rango sin lanzar excepciones, y expón la lectura mediante AccessibleDescription para que no exista solo como píxeles.

Punto de partida

Abres VisualOperationsStudio tal como quedó en el Módulo 1, donde VisualOperationsPage ya contiene un único modelo TelemetrySample y muestra su valor en cuatro superficies, entre ellas un Panel con un controlador de Paint incrustado que dibuja una barra. En este laboratorio ese controlador incrustado se sustituye: añades una clase TelemetryGauge al proyecto, colocas tres instancias en la página y enlazas cada una a una lectura del mismo modelo.

Enfoque sugerido

- ✓ Añade una clase TelemetryGauge que derive de un control de Wisej.NET, dale campos de respaldo más Value, Minimum, Maximum, WarningThreshold, CriticalThreshold y Caption, y escribe un setter que acote, compare, salga de inmediato y llame a Invalidate() una sola vez; después copia esa forma a los demás.
- ✓ Escribe GaugeGeometry como ayudante puro: un método que reciba el Rectangle dibujable y el valor normalizado y devuelva los límites del anillo, los ángulos inicial y de barrido, los puntos de la aguja y el rectángulo de la etiqueta, sin ninguna referencia a Graphics en todo el archivo.
- ✓ Asocia el controlador de Paint (o sobrescribe el enganche de dibujo) y dibuja estrictamente a partir del resultado de la geometría: la pista inactiva, las zonas de advertencia y crítica, el arco activo, la aguja construida como un GraphicsPath y, después, el valor numérico y el rótulo medidos con MeasureString y un StringFormat.
- ✓ Envuelve en una declaración using todos los pens, brushes, fuentes y paths que crees, usa los estáticos del framework como Brushes.Black sin liberarlos, y envuelve el RotateTransform de la aguja en Graphics.BeginContainer() y EndContainer(state) para que el rótulo se dibuje en coordenadas sin transformar.
- ✓ Protege los casos límite: sal de inmediato cuando el rectángulo sea demasiado pequeño para dibujar, normaliza un rango invertido, acota un valor que quede fuera de él, establece AccessibleDescription desde el setter y, después, redimensiona el navegador, cambia el valor

desde un botón y registra el tamaño de la imagen y la frecuencia de redibujado en `PaintCost.md`.

Entregables requeridos

- ✓ Control TelemetryGauge con propiedades Value, Minimum, Maximum y de umbral que acotan el valor y llaman a Invalidate() una vez por cambio real
- ✓ Ayudante puro GaugeGeometry que devuelve los límites del anillo, los ángulos de barrido, los puntos de la aguja y el rectángulo de la etiqueta
- ✓ Controlador de Paint que libera todos los pens, brushes, fuentes y paths que crea y nunca libera e.Graphics
- ✓ Comprobaciones para tamaño cero, rango invertido y valores fuera de rango, con un AccessibleDescription que lleve la lectura
- ✓ PaintCost.md que anote el tamaño del redibujado, con qué frecuencia se redibuja y cuándo saldría más barato un control con tema

Preguntas de repaso

- ✓ Un compañero traslada los cálculos del anillo y la aguja directamente al controlador de `Paint` porque «son solo unas líneas». ¿Qué dos cosas dejan de poder hacerse de forma barata a partir de entonces, y de cuál de ellas depende el Módulo 6?
- ✓ Tu indicador está enlazado a un registro de máquina que lleva además el nombre del operario y una fecha de mantenimiento. Si esos campos cambian, ¿debería invalidarse el control? Justifica tu respuesta en términos de dependencias visuales y no en términos del modelo.
- ✓ Ese mismo indicador va a ir a un panel que muestra sesenta de ellos y se actualiza cada dos segundos. ¿Qué medirías primero y qué cambiarías si las mediciones fueran malas?



Regla de dibujo: primero la superficie

Elige la superficie de renderizado antes de escribir una sola llamada de dibujo. Pregúntate dónde se producen los píxeles y qué cruza la red: un controlador de Paint de un control se ejecuta en el servidor y envía una imagen, CellPaint hace lo mismo dentro de una celda de cuadrícula, Canvas envía comandos de dibujo que ejecuta el navegador y no conserva nada tras un redimensionado, y System.Drawing.Managed produce bytes de imagen sin ningún control de por medio. Mantén el modelo semántico en objetos C#, haz que cada renderizador sea una función determinista de ese modelo, invalida solo cuando haya cambiado una dependencia visual, y deja siempre un equivalente textual de todo lo importante que exista únicamente como píxeles.