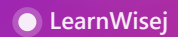


Wisej.Web.Canvas and the HTML5 Canvas 2D Model

Lesson Guide · Lab / Exam Guide



Lesson Guide

Reading browser Canvas 2D documentation and writing the Wisej.NET equivalent: PascalCase members on Wisej.Web.Canvas, degree-based angles where the browser uses radians, Save and Restore around transforms, gradients, patterns, shadows and clipping, and Redraw as the one place a scene is reconstructed. What this means for an experienced .NET developer who already builds Wisej.NET screens and now needs custom graphics — and how to read this module.



Production focus

Reading browser Canvas 2D documentation and writing the Wisej.NET equivalent: PascalCase members on Wisej.Web.Canvas, degree-based angles where the browser uses radians, Save and Restore around transforms, gradients, patterns, shadows and clipping, and Redraw as the one place a scene is reconstructed.

1. The production problem

Everything VisualOperationsStudio has drawn so far was produced on the server: the Module 2 gauge and the Module 3 Health and Trend cells are rendered with `System.Drawing` and reach the user as images. The topology view the application needs next is dozens of small nodes, edges and labels that a user will eventually pan, zoom and select, which is exactly the work a browser canvas does well.

`Wisej.Web.Canvas` hands you that surface from C#, and charges a different failure mode for it. The pixels belong to the browser, nothing on the server remembers them, and a resize can wipe them out. Code that draws once looks perfect in the designer and loses its scene the first time a user drags a window edge.

2. Key concepts

Concept	Why it matters
<code>Wisej.Web.Canvas</code>	A Wisej control exposing the browser's immediate-mode 2D surface to C#; its calls are drawing commands the browser executes, not <code>System.Drawing.Graphics</code> operations.
<code>Redraw</code> event	The hook the surface raises when it needs its content again; because Wisej.NET does not retain the commands, the handler has to reconstruct the whole scene.
PascalCase correspondence	Members map by case — <code>fillRect()</code> to <code>FillRect()</code> , <code>strokeStyle</code> to <code>StrokeStyle</code> — while CSS values become .NET types such as <code>Color</code> , <code>Rectangle</code> , <code>Font</code> and <code>Image</code> .
Degree-based angles	Wisej.NET documents angle arguments in degrees for simplicity where the browser uses radians, which changes every ported <code>Arc()</code> and <code>Rotate()</code> call.
<code>Save()</code> / <code>Restore()</code>	Push and pop the whole rendering state — styles, transform, clip, alpha, line and text settings — so a local change cannot leak into the next shape.
<code>TextFont</code> , <code>TextAlign</code> , <code>TextBaseline</code>	The Wisej.NET stand-in for the browser's <code>font</code> , <code>textAlign</code> and <code>textBaseline</code> ; the font is a real <code>System.Drawing.Font</code> .
<code>LiveUpdate</code>	False by default, so commands travel with the request's normal UI update; true pushes each call at once, which suits a progressive reveal and ruins an animation loop.

3. Lesson narrative

Read the browser documentation, then write the C#

The Wisej.NET Canvas is a server-programmable facade over the browser's 2D context, so you learn it by translating examples. Most of that is mechanical: `beginPath()` becomes `BeginPath()`, `globalAlpha` becomes `GlobalAlpha`, `setLineDash()` becomes `SetLineDash()`. Where the browser expects a CSS string, Wisej.NET expects a .NET value, so a colour is a `Color`, a font is a `Font` and an image is an `Image` or an image source string.

The half a browser reference cannot tell you is that the documented Wisej.NET member list is not the whole modern Canvas API. `measureText`, `getImageData`, `roundRect`, `ellipse` and `createConicGradient` do not appear in it. Translating therefore includes one extra step: confirm the member exists in the version you build against. When it does not, the honest answer is a Widget or client JavaScript, never a guessed method name.

Redraw is the only place your scene exists

Canvas drawing calls are not retained by Wisej.NET. The commands are issued, the browser paints them, and the server's involvement ends. When the surface is resized or refreshed the bitmap can be cleared, and no cached copy exists to put it back. That single fact separates working Canvas code from code that only appears to work.

So the scene lives in C# objects, one method rebuilds it in full, and `Redraw` calls that method. In the lab `DrawPlayground` opens with `ClearRect` across the whole surface and re-issues every command, which is why it gives the same result on first display, after a value change and after a resize.

```
private void canvasPlayground_Redraw(object sender, EventArgs e) => DrawPlayground();

private void DrawPlayground()
{
    canvasPlayground.ClearRect(0, 0, canvasPlayground.Width, canvasPlayground.Height);

    canvasPlayground.Save();
    canvasPlayground.Translate(240, 160);
    canvasPlayground.Rotate(45);           // browser: ctx.rotate(Math.PI / 4)
    canvasPlayground.FillStyle = Color.Coral;
    canvasPlayground.FillRect(-60, -24, 120, 48);
    canvasPlayground.Restore();
}
```

Two habits break this. Drawing in `Load` runs once, so the scene is gone after the first resize. A handler that repaints only what changed is worse: it works while you develop and leaves a half-empty surface in front of a user.

Paths, fills, text, gradients and shadows

A path is built first and painted afterwards. `MoveTo()` and `LineTo()` add geometry but paint nothing; `Stroke()` or `Fill()` does the painting with the current `StrokeStyle`, `LineWidth`, `LineCap` and `LineJoin`. `BeginPath()` starts an independent path, and omitting it is the classic port bug: a later stroke quietly includes the previous shape. `FillRect()`, `StrokeRect()` and `ClearRect()` act at once, while `Rect()` adds to the path you are building.

`CreateLinearGradient()` and `CreateRadialGradient()` return objects you assign to `FillStyle` or `StrokeStyle`, built from an array of colour stops, and `CreatePattern()` takes an `Image` or an image source plus a `CanvasRepeatPattern`. Text goes through `TextFont`, `TextAlign` and `TextBaseline` before `FillText()`; images through the `DrawImage()` overloads. Depth comes from the shadow properties, `GlobalAlpha` and `GlobalCompositeOperation` — compositing is subtle and browser-dependent, so test anything you ship with it.

Save, Restore and the transformation stack

Canvas rendering state is far larger than the transform, and `Save()` pushes all of it while `Restore()` pops it back. `Translate()`, `Rotate()` and `Scale()` modify the current transformation, `Transform()` composes another matrix onto it, and `SetTransform()` resets to identity and applies the one you supply.

Bracket every node: save, position, draw, restore. Without the restore, every later shape inherits the shifted coordinate system and the scene drifts in a way that is maddening to debug. A clip behaves identically, because it becomes part of the current state, so a temporary clip is always save, path, `Clip()`, draw, restore. One `Restore()` puts back:

- `FillStyle`, `StrokeStyle` and `GlobalAlpha`
- the current transformation matrix and the active clipping region

- `LineWidth` , `LineCap` , `LineJoin` and the dash pattern
- the text settings and the shadow properties

Degrees instead of radians, and when `LiveUpdate` earns its place

The Wisej.NET custom-painting documentation calls out one deliberate departure: angle arguments are expressed in degrees. A sample that writes `ctx.rotate(Math.PI / 4)` becomes `Rotate(45)` , and a circle running `0` to `Math.PI * 2` becomes `0` to `360` in `Arc()` . Copying the radian value compiles perfectly and turns the shape by a fraction of a degree, which is why the lab asks for the conversion in a comment beside the ported line.

The last documented difference is delivery. `LiveUpdate` is false by default, so the commands issued during one request travel with that request's UI update: one round trip, one repaint. True pushes each call immediately, which is genuinely useful when a long server-side operation should reveal its steps, and costly everywhere else. It is not a route to server-driven animation. Keep it off for the main render so the cost is something you chose.



Surface-first drawing habit

Choose the rendering surface before you write a single drawing call. Ask where the pixels are produced and what crosses the network: a control Paint handler runs on the server and ships an image, `CellPaint` does the same inside one grid cell, `Canvas` sends drawing commands the browser executes and keeps nothing across a resize, and `System.Drawing.Managed` produces image bytes with no control at all. Keep the semantic model in C# objects, make every renderer a deterministic function of that model, invalidate only when a visual dependency changed, and always leave a textual equivalent of anything important that exists only as pixels.



Learning objectives

By the end of this module you'll be able to:

- ✓ Explain the correspondence between `CanvasRenderingContext2D` and `Wisej.Web.Canvas`.
- ✓ Use `Redraw` as the canonical scene-reconstruction hook.
- ✓ Draw paths, rectangles, text, images, gradients, patterns, shadows and composited content.
- ✓ Use `Save/Restore` and transformation methods safely.
- ✓ Understand documented differences such as C# naming and degree-based angles.

Lab / Exam Guide

What you'll build in the hands-on lab, the suggested approach, and the required deliverables.

Hands-on lab

Build a Canvas playground page in VisualOperationsStudio that ports a set of browser Canvas 2D examples to the documented Wisej.Web.Canvas API. Drive everything from a DrawPlayground method called by the Canvas Redraw event so the scene returns after a resize, and start it with ClearRect over the full surface. Render a stroked path, filled rectangles, a placed caption, a linear gradient and a radial gradient, then wrap each transformed section in Save() and Restore() and demonstrate Translate, Scale, Rotate and SetTransform. Add a clipped region, an alpha section, a dashed line and a shadow, port one browser example that rotates by $\text{Math.PI} / 4$ into the degree value Wisej expects, and keep LiveUpdate off for the main render while showing it in one small progressive example.

Starting point

Open VisualOperationsStudio as Module 3 left it: the `TelemetryGauge` control paints on the server, `gridOperations` has user-painted Health and Trend columns, and the Module 1 page still shows one telemetry value on four surfaces. Add a new `CanvasPlaygroundPage` holding a single `Wisej.Web.Canvas` docked to fill it, plus a small panel of buttons; nothing on that page draws anything yet.

Suggested approach

- ✓ Write `DrawPlayground()` first and subscribe it to `canvasPlayground.Redraw` in the constructor. Open the method with `ClearRect(0, 0, canvasPlayground.Width, canvasPlayground.Height)`, draw one stroked polyline with `BeginPath`, `MoveTo`, `LineTo` and `Stroke`, run the page and resize the browser before you add anything else.
- ✓ Add the fill section: two `FillRect` shapes, a caption placed with `TextFont`, `TextAlign` and `TextBaseline`, then a bar filled from `CreateLinearGradient` and a sphere filled from `CreateRadialGradient`. Check each member against the Wisej.NET Canvas API before you type it.
- ✓ Build the transform row as four bracketed blocks. Each one is `Save()`, then `Translate`, `Scale`, `Rotate` or `SetTransform`, then a small labelled shape, then `Restore()`. Take one browser example that calls `ctx.rotate(Math.PI / 4)`, write `Rotate(45)`, and put the conversion in a comment on the same line.
- ✓ Add the state section: a clipped region built as a path followed by `Clip()` inside a save/restore pair, a block drawn under a reduced `GlobalAlpha`, a dashed rule using `SetLineDash`, and a card drawn with `ShadowColor`, `ShadowBlur` and the shadow offsets. Comment out one `Restore()` on purpose, watch the rest of the scene shift, then put it back.
- ✓ Leave `LiveUpdate` false for `DrawPlayground`. Give one button a short server-side loop that turns `LiveUpdate` on, draws a progress step per iteration and turns it off again, then write down the traffic difference and the cases where that trade is worth making.

Required deliverables

- ✓ Canvas playground whose entire scene is drawn by a DrawPlayground method called from Redraw
- ✓ Paths, rectangles, a placed caption, a linear gradient and a radial gradient using documented Wisej.Web.Canvas members only
- ✓ Save() and Restore() around every Translate, Scale, Rotate and SetTransform section
- ✓ Clipping, alpha, a dashed line and a shadow, plus one radians-to-degrees port written out in a comment
- ✓ A LiveUpdate demonstration kept separate from the main render, with a note on when it is appropriate

Student review questions

- ✓ Your playground looks right until a colleague resizes the browser, after which only the last shape you drew is visible. What does that tell you about where the rest of the scene was coming from?
- ✓ You need the pixel width of a caption so a badge can be sized to fit it. The Wisej.NET Canvas API does not list a text-measurement member — describe two ways to get the layout you want anyway, and what each one costs.
- ✓ A reviewer asks why the clip in your state section is wrapped in `Save()` and `Restore()` when the shapes after it are drawn somewhere else entirely. Answer them.



Surface-first drawing habit

Choose the rendering surface before you write a single drawing call. Ask where the pixels are produced and what crosses the network: a control Paint handler runs on the server and ships an image, CellPaint does the same inside one grid cell, Canvas sends drawing commands the browser executes and keeps nothing across a resize, and System.Drawing.Managed produces image bytes with no control at all. Keep the semantic model in C# objects, make every renderer a deterministic function of that model, invalidate only when a visual dependency changed, and always leave a textual equivalent of anything important that exists only as pixels.