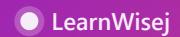


Theming, Resources, Localization & UI Modernization

Go deeper on theming: runtime light/dark switching, reusable status chips via CSSStyle, managing resources, and culture-aware localization.



Applied goal

Modernize the TicketOps operations dashboard end to end: switch a light/dark theme at runtime, build a reusable status-chip control styled through CSSStyle, manage icons and images as resources, and localize labels and formatting for two cultures.

1. Switching themes at runtime with Application.Theme

The TicketOps Console ships a header toggle so operators can move between a light theme for daytime shifts and a dark theme for the overnight operations desk. Assign a theme name to `Application.Theme` and Wisej.NET re-renders every open window; you do not rebuild controls yourself.

Keep the theme names in one place, and persist the operator's choice so the dashboard reopens in the same theme after a session refresh.

```
// TicketOps header: a single toggle drives the whole console
private void themeSwitch_CheckedChanged(object sender, EventArgs e)
{
    // Built-in Wisej.NET theme names shipped with the app
    var theme = themeSwitch.Checked ? "BootstrapDark-4" : "Bootstrap-4";

    Application.Theme = theme;           // re-themes all open windows live
    Application.Session["ticketops.theme"] = theme; // remember per session
}

// On startup, restore the last theme before the dashboard is shown
protected override void OnLoad(EventArgs e)
{
    if (Application.Session["ticketops.theme"] is string saved)
        Application.Theme = saved;

    base.OnLoad(e);
}
```



Production habit

Never scatter theme names as literals across screens. Keep `"Bootstrap-4"` and `"BootstrapDark-4"` in one constant so a rebrand is a one-line change.

2. A reusable status chip styled by CSSStyle, not hard-coded colors

Every Work Order row shows a status: *Open*, *In Progress*, *Blocked*, *Resolved*. Instead of colouring labels by hand on each screen, build one `StatusChip` UserControl. It exposes a `Status` property and applies a `CSSStyle` class token; the actual colours live in the theme, so both light and dark themes render the chip correctly.

```
// StatusChip: a reusable UserControl for the operations dashboard
public partial class StatusChip : UserControl
{
    private WorkOrderStatus _status;

    public WorkOrderStatus Status
    {
        get => _status;
        set
        {
            _status = value;
            labelText.Text = value.ToString();

            // Drive appearance through a CSSStyle token, not inline colors
            // Theme defines .chip-open, .chip-blocked, etc. for each theme
            this.CssClass = "chip chip-" + value.ToString().ToLowerInvariant();
        }
    }
}
```

Because the chip references a class name and the theme supplies the palette, the same control looks right in daylight and at night without a single conditional in your code.

3. Choosing where a style belongs

The most common modernization mistake is putting colour in the wrong layer. Use this table to decide where each visual rule lives before you write it.

Technique	Scope	Use it for	Avoid it for
Theme / Theme Builder	Whole application	Palette, typography, default control look, chip colours	One-off tweaks on a single control
CSSStyle / CssClass	A control or a reusable UserControl	Shared component variants like the status chip	App-wide colour decisions
Inline BackColor / ForeColor	One control instance	Almost nothing in modern screens	Anything you will need to re-theme later



Pitfall: hard-coded colors

A chip set with `chip.BackColor = Color.Red` stays red on the dark theme and breaks contrast. Route the look through the theme or a `CssClass` so both themes stay correct.

4. Managing icons and images as resources

The dashboard toolbar needs a refresh icon, an export icon, and a status glyph per chip. Treat these as managed resources rather than loose file paths. Wisej.NET can resolve theme-aware images, and reusing one resource reference keeps the toolbar consistent across screens.

Resource type	Best for	How it is referenced
Theme image mixin	Icons that differ per theme (light vs dark glyphs)	By theme image name, resolved automatically
Embedded / project resource	Logos and images that never change per theme	Through the generated resource accessor
External file / URL	User-supplied content such as attachments	By path or URL at runtime

```
// Toolbar buttons pull from managed resources, not scattered paths
refreshButton.ImageSource = "resource.wx/TicketOps.Icons.refresh.svg";
exportButton.ImageSource = "resource.wx/TicketOps.Icons.export.svg";

// One reference reused everywhere keeps the toolbar consistent
public static class Glyphs
{
    public const string Refresh = "resource.wx/TicketOps.Icons.refresh.svg";
    public const string Export = "resource.wx/TicketOps.Icons.export.svg";
}
```

5. Localizing labels and formatting for two cultures

TicketOps runs for English and Italian operations teams. Two things must change together: the **text** of labels (via localizable resources) and the **formatting** of dates and currency (via `CultureInfo`). Localizing one without the other produces English words next to Italian-formatted numbers, which looks broken.

Mark form and control text as localizable so translated `.resx` files supply each culture. For values, format explicitly against the operator's culture instead of relying on the server default.

```
// Culture-aware formatting for the Work Order detail panel
using System.Globalization;

// en-US → 7/3/2026, $1,250.00   it-IT → 03/07/2026, 1.250,00 €
var culture = new CultureInfo(operatorCultureCode); // "en-US" or "it-IT"

createdLabel.Text = workOrder.CreatedOn.ToString("d", culture);
costLabel.Text    = workOrder.LaborCost.ToString("C", culture);
slaLabel.Text     = workOrder.DueBy.ToString("g", culture);

// Set the request culture so localizable resources resolve too
Application.CurrentCulture = culture;
Application.CurrentUICulture = culture;
```



Production habit

Set `CurrentUICulture` for text and `CurrentCulture` for formatting together, then re-test the widest label. Italian strings often run longer than English and can clip a fixed-width chip or button.

6. A UI-modernization checklist applied to the dashboard

Walk each operations screen against these questions before calling it modern. They turn the abstract rules from the Lesson Guide into concrete checks on real TicketOps screens.

- ✓ Does every status colour come from the theme or a `CssClass`, with no inline `BackColor` ?
- ✓ Does the light/dark toggle re-theme the whole console, including chips and toolbar icons?
- ✓ Are all toolbar icons resolved from managed resources rather than loose paths?
- ✓ Is every visible label localizable, and does the panel format dates and currency `CultureInfo` by ?
- ✓ Does the widest translated label still fit without clipping in both themes?
- ✓ Could a future rebrand change the palette without editing individual screens?

Key vocabulary

- ✓ **Application.Theme** — assign a theme name to re-render all open windows at runtime.
- ✓ **CssClass / CSSStyle** — attach a class token so a reusable control inherits theme colours.
- ✓ **Status chip** — one `UserControl` expressing Work Order status through a class, not inline colour.
- ✓ **Managed resource** — an icon or image referenced by name so it stays consistent and theme-aware.
- ✓ **Localizable property** — control text sourced from culture-specific `.resx` resources.
- ✓ **CultureInfo** — drives culture-aware date and currency formatting for each operations team.