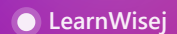


Theme Builder and Theme JSON Internals

Lesson Guide · Lab / Exam Guide



Lesson Guide

Creating a theme from a base theme in the Theme Builder, the JSON structure of colors, fonts, images, appearances, states, styles, properties and components, appearance inheritance and state order, and named color and font tokens as design assets. What this means for a Wisej.NET developer — and how to read this module.



Production focus

Creating a theme from a base theme in the Theme Builder, the JSON structure of colors, fonts, images, appearances, states, styles, properties and components, appearance inheritance and state order, and named color and font tokens as design assets.

1. The production problem

Module 1 gave the Adaptive Operations Console a docked shell, but the base theme still owns every colour and font. The shortcut is to brand it control by control with `BackColor`, `Font`, `CssStyle` strings or CSS against the widgets' internal DOM. Each scatters one hex value across dozens of places, fights the theme engine that already renders every state, and breaks on the next upgrade. A theme is one JSON file that owns the identity of the standard controls; the Theme Builder is how you produce and maintain it, and this module builds the AdaptiveOps theme with it.

2. Key concepts

Concept	Why it matters
base theme	The built-in theme the Theme Builder copies to create yours, so hundreds of appearances, components, states, metrics, images and fonts exist before you change a value.
named color and font tokens	Entries in the colors and fonts sections (<code>brandPrimary</code> , <code>danger</code> , <code>heading</code>) that appearances reference by name, so a brand change is one edit.
appearance and <code>AppearanceKey</code>	An appearance is the theme block for one kind of widget; a control's <code>AppearanceKey</code> selects which block applies, which is how a semantic variant such as an action button exists.
appearance inheritance	An <code>inherit</code> entry lets a specialised appearance reuse a base one and override only a few states, so <code>action-button</code> stays a real button everywhere else.
state order	States such as default, hovered, pressed, focused, disabled and invalid are applied in list order; moving one changes which styles win, so order is behaviour.
child component	An inner part of a widget, such as a panel caption bar, a grid column header or an editor button, with its own appearance path and states.
styles versus properties	Styles are CSS-like values the theme turns into dynamic CSS classes and owns; properties are widget properties that may become inline style or behaviour and can be overridden by the application.

3. Lesson narrative

Start from a base theme and change it in the right order

Never start a Wisej.NET theme from an empty file. The Theme Builder creates a new theme by copying an existing one, and that structure is enormous: every control has an appearance, most have child components with their own states, and the designer's metrics, images and fonts are all in there. The saved copy is the whole theme; the application does not depend on the original at runtime.

The professional workflow has a fixed order. Start from the base theme you recorded in Module 1. Save the copy as `AdaptiveOps.theme` in the project's Themes folder. Change named colors and fonts first, because most appearances already reference them and the whole preview moves at once. Change individual component appearances second, only where a token did not reach. Add custom appearances last, and only for a real variant such as the primary command, so every change stays small and readable in a diff.

Loading the theme is a separate decision. The application picks its startup theme from the `theme` entry in `Default.json` or the `Wisej.DefaultTheme` setting in `Web.config` ; changing the theme shown in the Visual Studio designer does not change what the browser receives. At runtime `Application.Theme` exposes the current session's theme, used in Module 3 to switch themes per session.

The Theme Builder: tree, property grid, preview, live update, Last Clicked

The main window is four tools working together. The tree is the theme file itself: colors, images, fonts, appearances and, inside each appearance, states, components, styles and properties; here you add, rename, delete and move elements. The property grid edits the selected element as typed fields. The preview renders real widgets, and with live update on every edit is visible immediately; disable it for a group of related edits, then re-enable it.

The most useful habit in the lab is the Last Clicked workflow. Click a widget in the preview and the Theme Builder reports the appearance path of what you clicked, for example the caption bar component inside the panel appearance or the column header inside the grid. Jump to that path in the tree and you edit exactly what the user sees. Discover the path first, then edit.

One tree operation deserves respect: moving states. Because matching states are applied in order, dragging a state changes the result, not just the list. Keep default first, and when a control shows the wrong colour while both hovered and pressed apply, check the order before you touch the colours.

Tokens: named colors and fonts as design assets

Named colors are the theme's design tokens. The AdaptiveOps palette defines `brandPrimary`, `brandAccent`, `surface`, `surfaceAlt`, `textMain`, `textMuted`, `danger`, `warning`, `success` and `focusFrame`, each once, in the colors section. Appearances reference the token name wherever a colour is expected, and a named color can point at another named color, so `focusFrame` can follow `brandPrimary`. The lab rule is simple: no literal hex value inside an appearance where a token exists, so a rebrand is one line in one file.

Fonts work the same way. Define a `default` text font, a `heading` font and a `mono` font for ticket identifiers, and reference them from appearances rather than setting `Font` on individual labels. A web font must be deployed with the application or referenced from a source the target browsers can load; a font the browser cannot find falls back silently and your screenshots will not match production.

Inheritance, state order, components, styles and properties

An appearance can inherit from another with the `inherit` entry. The lab's `action-button` inherits from `button` and overrides only its default, hovered and pressed states, as the JSON below shows. Focused and disabled are not mentioned, so they come from `button` unchanged, and a control gets the variant by setting `AppearanceKey` to `"action-button"`.

```

"action-button": {
  "inherit": "button",
  "states": {
    "default": {
      "styles": { "backgroundColor": "brandPrimary", "textColor": "white" }
    },
    "hovered": {
      "styles": { "backgroundColor": "#1F4A92" }
    },
    "pressed": {
      "styles": { "backgroundColor": "#183B75", "transform": "translate(1px, 1px)" }
    }
  }
}

```

States describe conditions: default, hovered, pressed, focused, disabled, invalid, selected, sorted, opened, active and widget-specific ones. Styles and properties are applied by matching the widget's appearance against the states currently active, in state order, which is why the lab styles the invalid editor on the invalid state of the editor appearance rather than in code. Components are the inner widgets: the panel's caption bar and close button, the grid's header and the editor's drop-down button each have their own states, so a part can be styled differently from its container.

Finally, styles and properties differ. Styles correspond to CSS values that Wisej.NET converts into dynamic CSS classes; they belong to the theme and are not changed by application code at runtime. Properties are widget properties defined by the JavaScript widget; some become inline DOM style, others alter behaviour. A theme property such as `backgroundColor` can override a style with the same effect, and an application property such as `BackColor` overrides the theme default. The property grid groups related properties, and spacing values such as padding accept a top, right, bottom, left shorthand. When a control ignores your edit, check in this order: an application property overriding it, the wrong `AppearanceKey`, the state order, and whether the running application loaded the theme you edited.



Common pitfalls

Starting from an empty theme and losing every inherited component. Typing the same hex value into twenty states instead of one named color. Moving default below hovered. Expecting the browser to change without `Default.json` pointing at the theme. Setting `BackColor` to fix a colour the theme should own.



Visual-system habit

Use the Wisej.NET mechanism that owns the problem: the theme for the identity of standard controls, `AppearanceKey` for semantic variants, `CssClass` for scoped app-specific CSS, `CssStyle` only for one-off dynamic values, layout containers instead of resize code, responsive properties for designer-set profile values, and `ResponsiveProfileChanged` for behaviour that is not simply visual.



Learning objectives

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

- ✓ Create a new theme from an existing base theme and organize theme changes without breaking inherited behavior.
- ✓ Use the Theme Builder tree, property grid, preview, live update, and Last Clicked workflow to discover appearance paths.
- ✓ Define named colors and font tokens and use them consistently across appearances.
- ✓ Explain appearance inheritance, state order, child components, style/property differences, and property groups.

Lab / Exam Guide

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



Hands-on lab

Create the AdaptiveOps theme from a built-in base theme in the Theme Builder. Define a named color palette with brandPrimary, brandAccent, surface, surfaceAlt, textMain, textMuted, danger, warning, success and focusFrame plus default, heading and mono fonts, then restyle the button, panel, tab, grid header, invalid editor and tooltip appearances using those tokens. Add an action-button appearance that inherits from button and overrides only its default, hovered and pressed states, verify every change in the Theme Builder preview and load the theme in the Adaptive Operations Console.

Starting point

Open the `AdaptiveOperationsConsole` solution from Module 1: the docked shell with toolbar, navigation, workspace, details and status panels runs on the base theme you chose and recorded, and `VisualInventory.md` already names the colour and font tokens this lab turns into a real theme. You need the Theme Builder installed and a browser to run the console; no theme file exists yet.

Suggested approach

- ✓ In the Theme Builder create a new theme from the base theme you chose in Module 1, name it `AdaptiveOps`, save it as `AdaptiveOps.theme` in the project's Themes folder and commit it before changing anything, so every later edit is a readable diff against the inherited structure.
- ✓ Add the named colors `brandPrimary`, `brandAccent`, `surface`, `surfaceAlt`, `textMain`, `textMuted`, `danger`, `warning`, `success` and `focusFrame` and the `default`, `heading` and `mono` fonts in the tree, then watch the preview move as inherited appearances pick the tokens up; disable live update while you enter a group of values and re-enable it to check the result.
- ✓ Click each target in the preview and read Last Clicked to find its appearance path, then restyle button, panel (including its caption bar component), tab, the grid's column header component, the invalid state of the editor appearance and the tooltip from tokens only; never type a hex value where a token exists.
- ✓ Add an `action-button` appearance that inherits from `button`, override only its default, hovered and pressed states, confirm default is the first state in the list and verify in the preview that focused and disabled still come from `button`.
- ✓ Point `Default.json` at `AdaptiveOps`, run the console, set `AppearanceKey` to `"action-button"` on the primary command, report `Application.Theme.Name` in the status label, take the preview and browser screenshots and write the note on state order and inheritance.

Required deliverables

- ✓ `AdaptiveOps.theme` created from a base theme and loaded by the application
- ✓ Named color tokens and default, heading and mono font tokens defined once and reused
- ✓ Button, panel, tab, grid header, invalid editor and tooltip appearances restyled from the tokens

- ✓ action-button appearance inheriting from button with overridden default, hovered and pressed states
- ✓ Theme Builder preview screenshots and a note on state order and inheritance decisions

Student review questions

- ✓ Marketing changes brandPrimary to a new hex value tomorrow. Which files and which lines change in your solution, and what would the answer have been if you had set BackColor on each button instead?
- ✓ You moved the pressed state above hovered in the action-button appearance. What does a user see while holding the mouse button down, and why does the order rather than the colour decide it?
- ✓ A TextBox in the details editor stays grey when it is invalid even though you styled the invalid state. Which three things do you check, in which order, before editing the theme again?



Visual-system habit

Use the Wisej.NET mechanism that owns the problem: the theme for the identity of standard controls, AppearanceKey for semantic variants, CssClass for scoped app-specific CSS, CssStyle only for one-off dynamic values, layout containers instead of resize code, responsive properties for designer-set profile values, and ResponsiveProfileChanged for behaviour that is not simply visual.