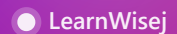


ClientProfile and Responsive Properties

Lesson Guide · Lab / Exam Guide



Lesson Guide

How ClientProfiles.json is matched top to bottom into Application.ActiveProfile, designer-assigned responsive property values per profile, ResponsiveProfileChanged for behaviour that responsive properties cannot express, and profile design and testing rules. What this means for a Wisej.NET developer — and how to read this module.



Production focus

How ClientProfiles.json is matched top to bottom into Application.ActiveProfile, designer-assigned responsive property values per profile, ResponsiveProfileChanged for behaviour that responsive properties cannot express, and profile design and testing rules.

1. The production problem

The Adaptive Operations Console now has a themed shell built from Flow, Table and Flex containers that survives a resize without a Resize handler, but it shows the same screen to everyone: on a 390-pixel phone the navigation rail, the ticket grid and the details editor compete for the same width. CSS media queries cannot fix this, because the controls live on the server: hiding a panel, switching a button to icon-only or re-docking a region is a server-side property change, and a phone user who loses the details panel needs a different way to edit a ticket, not a smaller copy of the desktop one. This module adds the mechanism Wisej.NET owns for that: client profiles, responsive properties and the `ResponsiveProfileChanged` event.

2. Key concepts

Concept	Why it matters
ClientProfile	A named set of device-related matching rules (browser width, screen width, device type, user agent, landscape) that describes how a class of client is used, not a marketing device name.
ClientProfiles.json	The file Wisej.NET reads; profiles are evaluated from top to bottom and the first one that matches wins, so the order of the entries is part of the configuration.
Application.ActiveProfile	The profile currently selected for this session; its <code>Name</code> is what the status label shows and what the profile handler branches on.
responsive property	A normal .NET property marked with <code>ResponsivePropertyAttribute</code> that holds one value per client profile and applies the matching value whenever the active profile changes.
designer-first adaptation	Responsive values are assigned only in the Visual Studio designer, which keeps them serialized, visible and deterministic; code is for adaptations that depend on data or runtime state.
ResponsiveProfileChanged	The event raised on <code>Application</code> , on containers such as Form, Page, Desktop and UserControl and on DataGridView and ListView when the active profile changes; <code>ResponsiveProfileChangedEventArgs.CurrentProfile</code> carries the new profile.
responsive QA matrix	The test plan listing width, height, orientation, device type, touch, scrollbars, font scaling and the profile name you expect at each size.

3. Lesson narrative

How a profile is matched

Each entry in `ClientProfiles.json` has a `name` and any combination of `device` (Mobile, Tablet or Desktop), `landscape`, `minWidth` and `maxWidth` for the browser viewport, `minScreenWidth` and `maxScreenWidth` for the physical screen, and a `userAgent` match. Wisej.NET reads the file, evaluates the profiles from top to bottom and selects the first match as `Application.ActiveProfile`. Nothing merges: a client matches one profile or none, which is why the pattern below falls back to "Default" when `ActiveProfile` is null.

Matching can happen when the page loads or when the browser resizes, depending on the property a rule uses: a viewport width rule is re-evaluated as the browser is dragged, while screen size, device type and user agent describe the client itself. Whenever the selected profile changes, Wisej.NET raises `ResponsiveProfileChanged`. For the lab you write six profiles in first-match order: Phone, Phone Landscape, Tablet, Tablet Landscape, Small Desktop and Desktop. The two phone entries share `device` Mobile and differ only in `landscape`; Small Desktop is a Desktop device with `maxWidth` 1024; Desktop is the broad rule at the bottom. Put it at the top and it shadows every profile below it, and nothing warns you.

Responsive properties belong to the designer

Responsive properties are ordinary properties marked with `ResponsivePropertyAttribute` : `Visible` , `Display` , `Size` , `Location` , `Dock` , `Anchor` , `Font` and layout-related values where the control supports them. In the designer such a property holds a different value for each profile in `ClientProfiles.json` , and when the active profile changes the control applies the value that belongs to it. The designer is the only supported way to assign these values, a rule rather than a preference: values assigned there are serialized with the form, visible to the next developer and deterministic at runtime, whereas values set in code are invisible at design time and drift from the designer's picture of the screen.

In the lab every deterministic difference in the console is designer work. On both Phone profiles the navigation rail UserControl and the details panel get `Visible` false. The toolbar buttons get an icon-only `Display` on Phone and Tablet so the command bar fits on one row. On the Tablet profiles the details region's `Dock` changes from Right to Bottom so it sits under the ticket grid instead of beside it. None of those changes depends on data, so none belongs in code; code is for adaptations that depend on data, permissions, a runtime calculation or the profile event itself.

ResponsiveProfileChanged for behaviour

Some adaptations are not values. On a phone the details editor cannot be a side panel at all; it has to open as a modal `Form` through `ShowDialog` , and the status label should show and log the profile name so support knows which layout a user sees. That is what `ResponsiveProfileChanged` is for. It fires on `Application` , on ContainerControl types such as Form, Page, Desktop and UserControl, and on selected complex controls such as DataGridView and ListView. Handle it on `Application` for application-wide behaviour and on a key container when the logic is local to that region; the event arguments expose `CurrentProfile` .

The pattern below is the shape the lab expects: subscribe once, apply the profile once at startup from `Application.ActiveProfile` so the initial layout is right, route the event through the same `ApplyProfile` method, and unsubscribe in `Dispose` because `Application` outlives the page. Keep `ApplyProfile` cheap and idempotent: a phone rotating between Phone and Phone Landscape fires it repeatedly, and an editor that is already open must be reused, never opened again. Typical event work: load fewer grid columns, change a search debounce interval, switch a details panel into a modal form, apply a different data template.

```
// MainPage.cs – one method applies the profile at startup and on every change
Application.ResponsiveProfileChanged += Application_ResponsiveProfileChanged;
ApplyProfile(Application.ActiveProfile?.Name);

private void Application_ResponsiveProfileChanged(object sender,
ResponsiveProfileChangedEventArgs e)
{
    ApplyProfile(e.CurrentProfile?.Name);
}

private void ApplyProfile(string profileName)
{
    var phone = profileName != null && profileName.StartsWith("Phone");
    this.lblProfile.Text = profileName ?? "Default"; // shown and logged
    if (phone && this.ticketEditor == null) OpenDetailsAsModal(); // side panel → modal,
    once
}
```

Design variants that change behaviour

Profiles should describe behaviour, not devices. Phone, Tablet Landscape, Small Desktop, Kiosk and Touch Workstation each name a way of using the console, and each variant should change what the console does: the phone gets a modal editor and a menu button, the tablet stacks the details region under the grid, the small desktop keeps the shell with an icon-only toolbar, the desktop shows everything. Use width rules for layout, device rules for input assumptions such as touch targets and hidden scrollbars, **userAgent** only when necessary, and landscape rules only when orientation truly changes the UI. A profile that changes nothing but a CSS dimension did not need to exist.

Test both ways. Real browser resizing exercises the width rules; device emulation exercises the device, screen width and user agent rules, and after switching device in the Chrome mobile emulator you must refresh the page so those profiles are re-evaluated. The responsive QA matrix has one row per scenario, from Desktop at 1440 by 900 down to Phone at 390 by 844 and Phone Landscape at 844 by 390, each with the expected profile name and the checks that prove the behaviour: navigation visible, details stacked, modal editor reachable, no horizontal overflow.

- Narrow or specific profiles before broad ones; the first match wins and nothing merges.
- Responsive values in the designer; **ResponsiveProfileChanged** only when a value is not enough, and then cheap and idempotent.
- Show the profile name in test builds and document each hidden control with its mobile alternative.



Common pitfalls

Listing the broad Desktop profile first so the phone and tablet profiles are never matched. Assigning responsive values in code instead of the designer. Putting every adaptation into the event handler, including the ones a **Visible** or **Dock** value would have expressed. Opening the modal editor again on every rotation because the handler is not idempotent. Hiding a panel on phone with no mobile alternative for what it contained.



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:

- ✓ Explain how ClientProfiles.json is read and matched and how Application.ActiveProfile is selected.
- ✓ Use responsive properties in the designer for properties such as Visible, Display, Size, Location, Dock, Anchor, Font, and layout-related values when available.
- ✓ Handle ResponsiveProfileChanged on Application and key containers when profile-specific logic is required.
- ✓ Design phone, tablet, small-desktop, and desktop variants that change UI behavior, not just CSS dimensions.

Lab / Exam Guide

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



Hands-on lab

Add a `ClientProfiles.json` to the Adaptive Operations Console with Phone, Phone Landscape, Tablet, Tablet Landscape, Small Desktop and Desktop profiles ordered narrow to broad. In the designer, use responsive properties to hide the navigation rail and details panel on phone, switch the toolbar buttons to icon-only Display, and re-dock the details region under the grid on tablet. Handle `ResponsiveProfileChanged` to show the active profile name in the status label, log each change and open the details editor as a modal form on phone instead of a side panel.

Starting point

Open the Adaptive Operations Console as Module 5 left it: the shell is nested containers with a Flow metric-card area, a `TableLayoutPanel` ticket editor inside the details `UserControl` and a Flex workspace, the `AdaptiveOps` theme and the scoped stylesheet are loaded, and the status label from Module 1 still reports the browser width. There is no `ClientProfiles.json` yet, so `Application.ActiveProfile` is null for every client and every width shows the desktop shell.

Suggested approach

- ✓ Write `ClientProfiles.json` in the application root with six entries in first-match order (Phone, Phone Landscape, Tablet, Tablet Landscape, Small Desktop, Desktop): the two phone entries use `device` Mobile and differ by `landscape`, the tablet pair does the same with Tablet, Small Desktop is a Desktop device with `maxWidth` 1024 and Desktop is the broad rule last. Run once and confirm the profile names appear in the designer.
- ✓ In the designer, give the navigation rail `UserControl` and the details panel a `Visible` value of false for both Phone profiles, and check that the workspace grid still has its `MinimumSize` so the phone layout does not collapse to zero.
- ✓ Still in the designer, set the toolbar buttons' `Display` to icon-only for the Phone and Tablet profiles, and change the details region's `Dock` from Right to Bottom for both Tablet profiles so it sits under the ticket grid; leave the desktop values untouched.
- ✓ In `MainPage`, subscribe once to `Application.ResponsiveProfileChanged`, write an `ApplyProfile(string profileName)` method that sets `lblProfile.Text` and writes one log line per change, call it at construction with `Application.ActiveProfile?.Name` and from the handler with `e.CurrentProfile?.Name`, and unsubscribe in `Dispose`.
- ✓ For the Phone profiles, make `ApplyProfile` open the ticket editor in a modal `Form` through `ShowDialog`, reusing the form if it is already open; then test by dragging the browser and by device emulation with a refresh after each device change, and capture the profile label at every size.

Required deliverables

- ✓ ClientProfiles.json with Phone, Phone Landscape, Tablet, Tablet Landscape, Small Desktop and Desktop profiles in first-match order
- ✓ Responsive property values assigned in the designer to hide the navigation rail and details panel on phone
- ✓ Icon-only toolbar Display and re-docked details region for tablet set through responsive properties
- ✓ ResponsiveProfileChanged handler that shows and logs the active profile name
- ✓ Details editor opened as a modal form on phone through the profile handler

Student review questions

- ✓ If Small Desktop were moved to the top of ClientProfiles.json, which of the other five profiles could still be matched, and what would a tablet in landscape see?
- ✓ The tablet re-dock lives in the designer and the phone modal editor lives in code. What decides which mechanism owns each change, and what would go wrong if you swapped them?
- ✓ A tester rotated a phone four times and your handler ran four times. Which state must be identical after the fourth run and after the first, and how does your code guarantee it?



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.