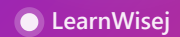


Theme Builder und der interne Aufbau von Theme-JSON

Lektionsleitfaden · Lab- und Prüfungsleitfaden



Lektionsleitfaden

Ein Theme im Theme Builder aus einem Basis-Theme erstellen, die JSON-Struktur aus Farben, Schriften, Bildern, Appearances, Zuständen, Styles, Eigenschaften und Komponenten, Vererbung von Appearances und Reihenfolge der Zustände sowie benannte Farb- und Schrift-Tokens als Design-Assets. Was das für Wisej.NET-Entwickler bedeutet – und wie Sie dieses Modul lesen.



Fokus: Produktivbetrieb

Ein Theme im Theme Builder aus einem Basis-Theme erstellen, die JSON-Struktur aus Farben, Schriften, Bildern, Appearances, Zuständen, Styles, Eigenschaften und Komponenten, Vererbung von Appearances und Reihenfolge der Zustände sowie benannte Farb- und Schrift-Tokens als Design-Assets.

1. Das Problem im Produktivbetrieb

Modul 1 hat der Adaptive Operations Console eine gedockte Shell gegeben, doch Farben und Schriften bestimmt noch vollständig das Basis-Theme. Die Abkürzung wäre, sie Control für Control mit `BackColor`, `Font`, `CssStyle`-Strings oder CSS gegen das interne DOM der Widgets zu branden. Jeder dieser Wege verstreut denselben Hex-Wert über Dutzende Stellen, kämpft gegen die Theme-Engine, die bereits jeden Zustand rendert, und bricht beim nächsten Upgrade. Ein Theme ist eine einzige JSON-Datei, die die Identität der Standard-Controls bestimmt; mit dem Theme Builder erstellen und pflegen Sie es, und in diesem Modul bauen Sie damit das AdaptiveOps-Theme.

2. Zentrale Konzepte

Konzept	Warum es wichtig ist
Basis-Theme	Das integrierte Theme, das der Theme Builder kopiert, um Ihres zu erstellen; so existieren Hunderte Appearances, Komponenten, Zustände, Metriken, Bilder und Schriften, bevor Sie einen einzigen Wert ändern.
benannte Farb- und Schrift-Tokens	Einträge in den Abschnitten colors und fonts (<code>brandPrimary</code> , <code>danger</code> , <code>heading</code>), auf die Appearances über den Namen verweisen, sodass eine Markenänderung eine einzige Bearbeitung ist.
Appearance und <code>AppearanceKey</code>	Eine Appearance ist der Theme-Block für eine Art von Widget; der <code>AppearanceKey</code> eines Controls wählt aus, welcher Block gilt – so entsteht eine semantische Variante wie ein Aktions-Button.
Vererbung von Appearances	Ein <code>inherit</code> -Eintrag erlaubt einer spezialisierten Appearance, eine Basis-Appearance wiederzuverwenden und nur wenige Zustände zu überschreiben, sodass <code>action-button</code> überall sonst ein echter Button bleibt.
Reihenfolge der Zustände	Zustände wie default, hovered, pressed, focused, disabled und invalid werden in Listenreihenfolge angewendet; verschiebt man einen, ändert sich, welche Styles gewinnen. Die Reihenfolge ist also Verhalten.
Child-Komponente	Ein innerer Teil eines Widgets, etwa die Titelleiste eines Panels, ein Spaltenkopf im Grid oder ein Editor-Button, mit eigenem Appearance-Pfad und eigenen Zuständen.
Styles und Eigenschaften	Styles sind CSS-ähnliche Werte, die das Theme in dynamische CSS-Klassen umwandelt und selbst verwaltet; Eigenschaften sind Widget-Eigenschaften, die zu Inline-Styles oder Verhalten werden können und von der Anwendung überschrieben werden dürfen.

3. Der rote Faden der Lektion

Von einem Basis-Theme ausgehen und in der richtigen Reihenfolge ändern

Beginnen Sie ein Wisej.NET-Theme nie mit einer leeren Datei. Der Theme Builder erstellt ein neues Theme, indem er ein bestehendes kopiert, und diese Struktur ist riesig: Jedes Control hat eine Appearance, die meisten haben Child-Komponenten mit eigenen Zuständen, und auch Metriken, Bilder und Schriften für den Designer stecken darin. Die gespeicherte Kopie ist das vollständige Theme; die Anwendung hängt zur Laufzeit nicht vom Original ab.

Der professionelle Arbeitsablauf hat eine feste Reihenfolge. Gehen Sie von dem Basis-Theme aus, das Sie in Modul 1 festgehalten haben. Speichern Sie die Kopie als `AdaptiveOps.theme` im Themes-Ordner des Projekts. Ändern Sie zuerst benannte Farben und Schriften, denn die meisten Appearances verweisen bereits darauf, und die ganze Vorschau ändert sich auf einmal. Ändern Sie danach einzelne Komponenten-Appearances, nur dort, wo ein Token nicht hingereicht hat. Fügen Sie zuletzt eigene Appearances hinzu, und nur für eine echte Variante wie den primären Befehl, damit jede Änderung klein bleibt und im Diff lesbar ist.

Das Laden des Themes ist eine eigene Entscheidung. Die Anwendung wählt ihr Start-Theme über den Eintrag `theme` in `Default.json` oder die Einstellung `Wisej.DefaultTheme` in `Web.config`; das im Visual-Studio-Designer angezeigte Theme zu ändern, ändert nicht, was der Browser erhält. Zur Laufzeit stellt `Application.Theme` das Theme der aktuellen Session bereit; in Modul 3 nutzen Sie es, um Themes pro Session zu wechseln.

Der Theme Builder: Baum, Eigenschaftenraster, Vorschau, Live-Aktualisierung, Last Clicked

Das Hauptfenster besteht aus vier zusammenarbeitenden Werkzeugen. Der Baum ist die Theme-Datei selbst: Farben, Bilder, Schriften, Appearances und innerhalb jeder Appearance Zustände, Komponenten, Styles und Eigenschaften; hier fügen Sie Elemente hinzu, benennen sie um, löschen und verschieben sie. Das Eigenschaftenraster bearbeitet das ausgewählte Element als typisierte Felder. Die Vorschau rendert echte Widgets, und mit eingeschalteter Live-Aktualisierung ist jede Änderung sofort sichtbar; schalten Sie sie für eine Gruppe zusammengehöriger Änderungen ab und danach wieder ein.

Die nützlichste Gewohnheit im Lab ist der Last-Clicked-Arbeitsablauf. Klicken Sie in der Vorschau auf ein Widget, und der Theme Builder meldet den Appearance-Pfad des angeklickten Elements, etwa die Titelleisten-Komponente innerhalb der Panel-Appearance oder den Spaltenkopf innerhalb des Grids. Springen Sie im Baum zu diesem Pfad, und Sie bearbeiten genau das, was der Benutzer sieht. Erst den Pfad ermitteln, dann bearbeiten.

Eine Operation im Baum verdient Respekt: das Verschieben von Zuständen. Weil zutreffende Zustände der Reihe nach angewendet werden, ändert das Ziehen eines Zustands das Ergebnis, nicht nur die Liste. Lassen Sie default an erster Stelle, und wenn ein Control die falsche Farbe zeigt, während hovered und pressed gleichzeitig gelten, prüfen Sie die Reihenfolge, bevor Sie die Farben anfassen.

Tokens: benannte Farben und Schriften als Design-Assets

Benannte Farben sind die Design-Tokens des Themes. Die AdaptiveOps-Palette definiert `brandPrimary`, `brandAccent`, `surface`, `surfaceAlt`, `textMain`, `textMuted`, `danger`, `warning`, `success` und `focusFrame`, jede einmal, im Abschnitt colors. Appearances verweisen überall dort auf den Token-Namen, wo eine Farbe erwartet wird, und eine benannte Farbe kann auf eine andere benannte Farbe verweisen, sodass `focusFrame brandPrimary` folgen kann. Die Lab-Regel ist einfach: kein literaler Hex-Wert in einer Appearance, wo es ein Token gibt, damit ein Rebranding eine Zeile in einer Datei ist.

Schriften funktionieren genauso. Definieren Sie eine Textschrift `default`, eine Überschriftenschrift `heading` und eine Schrift `mono` für Ticketnummern und verweisen Sie aus den Appearances darauf, statt `Font` an einzelnen Labels zu setzen. Eine Webschrift muss mit der Anwendung ausgeliefert oder aus einer Quelle referenziert werden, die die Zielbrowser laden können; eine Schrift, die der Browser nicht findet, wird stillschweigend ersetzt, und Ihre Screenshots stimmen nicht mit dem Produktivbetrieb überein.

Vererbung, Reihenfolge der Zustände, Komponenten, Styles und Eigenschaften

Eine Appearance kann über den Eintrag `inherit` von einer anderen erben. Die `action-button` - Appearance des Labs erbt von `button` und überschreibt nur ihre Zustände default, hovered und pressed, wie das JSON unten zeigt. Focused und disabled werden nicht erwähnt, sie kommen also unverändert von `button`, und ein Control erhält die Variante, indem Sie `AppearanceKey` auf `"action-button"` setzen.

```

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

```

Zustände beschreiben Bedingungen: default, hovered, pressed, focused, disabled, invalid, selected, sorted, opened, active und widgetspezifische. Styles und Eigenschaften werden angewendet, indem die Appearance des Widgets in Zustandsreihenfolge mit den aktuell aktiven Zuständen abgeglichen wird. Deshalb gestaltet das Lab den ungültigen Editor über den Zustand invalid der Editor-Appearance statt im Code. Komponenten sind die inneren Widgets: Titelleiste und Schließen-Button des Panels, der Header des Grids und der Dropdown-Button des Editors haben jeweils eigene Zustände, sodass ein Teil anders gestaltet werden kann als sein Container.

Schließlich unterscheiden sich Styles und Eigenschaften. Styles entsprechen CSS-Werten, die Wisej.NET in dynamische CSS-Klassen umwandelt; sie gehören zum Theme und werden zur Laufzeit nicht vom Anwendungscode geändert. Eigenschaften sind Widget-Eigenschaften, die das JavaScript-Widget definiert; manche werden zu Inline-Styles im DOM, andere verändern das Verhalten. Eine Theme-Eigenschaft wie `backgroundColor` kann einen Style mit derselben Wirkung überschreiben, und eine Anwendungseigenschaft wie `BackColor` überschreibt den Standardwert des Themes. Das Eigenschaftenraster gruppiert zusammengehörige Eigenschaften, und Abstandswerte wie padding akzeptieren die Kurzform oben, rechts, unten, links. Wenn ein Control Ihre Änderung ignoriert, prüfen Sie in dieser Reihenfolge: eine überschreibende Anwendungseigenschaft, den falschen `AppearanceKey`, die Reihenfolge der Zustände und ob die laufende Anwendung das Theme geladen hat, das Sie bearbeitet haben.



Typische Fehler

Mit einem leeren Theme beginnen und dabei jede geerbte Komponente verlieren. Denselben Hex-Wert in zwanzig Zustände tippen statt in eine benannte Farbe. default unter hovered verschieben. Erwarten, dass sich der Browser ändert, ohne dass `Default.json` auf das Theme verweist. `BackColor` setzen, um eine Farbe zu korrigieren, für die das Theme zuständig sein sollte.



Gewohnheit für das visuelle System

Nutzen Sie den Wisej.NET-Mechanismus, der für das Problem zuständig ist: das Theme für die Identität der Standard-Controls, AppearanceKey für semantische Varianten, CssClass für bereichsbezogenes anwendungsspezifisches CSS, CssStyle nur für einmalige dynamische Werte, Layout-Container statt Resize-Code, responsive Eigenschaften für im Designer gesetzte Profilwerte und ResponsiveProfileChanged für Verhalten, das nicht rein visuell ist.



Lernziele

Am Ende dieses Moduls können Sie:

- ✓ ein neues Theme aus einem bestehenden Basis-Theme erstellen und Theme-Änderungen so organisieren, dass geerbtes Verhalten nicht bricht.
- ✓ Baum, Eigenschaftenraster, Vorschau, Live-Aktualisierung und den Last-Clicked-Arbeitsablauf des Theme Builders nutzen, um Appearance-Pfade zu ermitteln.
- ✓ benannte Farben und Schrift-Tokens definieren und sie in allen Appearances konsistent verwenden.
- ✓ Vererbung von Appearances, Reihenfolge der Zustände, Child-Komponenten, den Unterschied zwischen Styles und Eigenschaften sowie Eigenschaftsgruppen erklären.

Lab- und Prüfungsleitfaden

Was Sie im praktischen Lab bauen, das vorgeschlagene Vorgehen und die geforderten Ergebnisse.



Praktisches Lab

Erstellen Sie im Theme Builder das AdaptiveOps-Theme aus einem integrierten Basis-Theme. Definieren Sie eine benannte Farbpalette mit `brandPrimary`, `brandAccent`, `surface`, `surfaceAlt`, `textMain`, `textMuted`, `danger`, `warning`, `success` und `focusFrame` sowie die Schriften `default`, `heading` und `mono`, und gestalten Sie anschließend die Appearances für Button, Panel, Tab, Grid-Header, ungültigen Editor und Tooltip mit diesen Tokens neu. Fügen Sie eine `action-button`-Appearance hinzu, die von `button` erbt und nur ihre Zustände `default`, `hovered` und `pressed` überschreibt, prüfen Sie jede Änderung in der Vorschau des Theme Builders und laden Sie das Theme in der Adaptive Operations Console.

Ausgangspunkt

Öffnen Sie die Solution `AdaptiveOperationsConsole` aus Modul 1: Die gedockte Shell mit Toolbar-, Navigations-, Arbeitsbereichs-, Details- und Status-Panel läuft mit dem Basis-Theme, das Sie gewählt und dokumentiert haben, und `VisualInventory.md` nennt bereits die Farb- und Schrift-Tokens, die dieses Lab in ein echtes Theme umsetzt. Sie brauchen den installierten Theme Builder und einen Browser, um die Konsole auszuführen; eine Theme-Datei gibt es noch nicht.

Vorgeschlagenes Vorgehen

- ✓ Erstellen Sie im Theme Builder ein neues Theme aus dem Basis-Theme, das Sie in Modul 1 gewählt haben, nennen Sie es `AdaptiveOps`, speichern Sie es als `AdaptiveOps.theme` im Themes-Ordner des Projekts und committen Sie es, bevor Sie etwas ändern, damit jede spätere Änderung ein lesbarer Diff gegenüber der geerbten Struktur ist.
- ✓ Fügen Sie im Baum die benannten Farben `brandPrimary`, `brandAccent`, `surface`, `surfaceAlt`, `textMain`, `textMuted`, `danger`, `warning`, `success` und `focusFrame` sowie die Schriften `default`, `heading` und `mono` hinzu und beobachten Sie, wie sich die Vorschau ändert, während geerbte Appearances die Tokens übernehmen; schalten Sie die Live-Aktualisierung ab, während Sie eine Gruppe von Werten eingeben, und wieder ein, um das Ergebnis zu prüfen.
- ✓ Klicken Sie jedes Ziel in der Vorschau an und lesen Sie unter Last Clicked seinen Appearance-Pfad ab; gestalten Sie dann Button, Panel (einschließlich seiner Titelleisten-Komponente), Tab, die Spaltenkopf-Komponente des Grids, den Zustand `invalid` der Editor-Appearance und den Tooltip ausschließlich über Tokens neu; tippen Sie nie einen Hex-Wert, wo es ein Token gibt.
- ✓ Fügen Sie eine `action-button`-Appearance hinzu, die von `button` erbt, überschreiben Sie nur ihre Zustände `default`, `hovered` und `pressed`, stellen Sie sicher, dass `default` der erste Zustand in der Liste ist, und prüfen Sie in der Vorschau, dass `focused` und `disabled` weiterhin von `button` kommen.
- ✓ Lassen Sie `Default.json` auf `AdaptiveOps` verweisen, starten Sie die Konsole, setzen Sie `AppearanceKey` beim primären Befehl auf `"action-button"`, melden Sie

`Application.Theme.Name` im Status-Label, machen Sie die Screenshots von Vorschau und Browser und schreiben Sie die Notiz zu Zustandsreihenfolge und Vererbung.

Geforderte Ergebnisse

- ✓ AdaptiveOps.theme aus einem Basis-Theme erstellt und von der Anwendung geladen
- ✓ Benannte Farb-Tokens sowie die Schrift-Tokens default, heading und mono einmal definiert und wiederverwendet
- ✓ Appearances für Button, Panel, Tab, Grid-Header, ungültigen Editor und Tooltip anhand der Tokens neu gestaltet
- ✓ action-button-Appearance, die von button erbt und die Zustände default, hovered und pressed überschreibt
- ✓ Screenshots der Theme-Builder-Vorschau und eine Notiz zu den Entscheidungen bei Zustandsreihenfolge und Vererbung

Wiederholungsfragen

- ✓ Das Marketing ändert brandPrimary morgen auf einen neuen Hex-Wert. Welche Dateien und welche Zeilen ändern sich in Ihrer Solution, und wie hätte die Antwort gelautet, wenn Sie stattdessen BackColor an jedem Button gesetzt hätten?
- ✓ Sie haben den Zustand pressed in der action-button-Appearance über hovered verschoben. Was sieht ein Benutzer, während er die Maustaste gedrückt hält, und warum entscheidet darüber die Reihenfolge und nicht die Farbe?
- ✓ Eine TextBox im Details-Editor bleibt grau, wenn sie ungültig ist, obwohl Sie den Zustand invalid gestaltet haben. Welche drei Dinge prüfen Sie in welcher Reihenfolge, bevor Sie das Theme erneut bearbeiten?



Gewohnheit für das visuelle System

Nutzen Sie den Wisej.NET-Mechanismus, der für das Problem zuständig ist: das Theme für die Identität der Standard-Controls, AppearanceKey für semantische Varianten, CssClass für bereichsbezogenes anwendungsspezifisches CSS, CssStyle nur für einmalige dynamische Werte, Layout-Container statt Resize-Code, responsive Eigenschaften für im Designer gesetzte Profilwerte und ResponsiveProfileChanged für Verhalten, das nicht rein visuell ist.